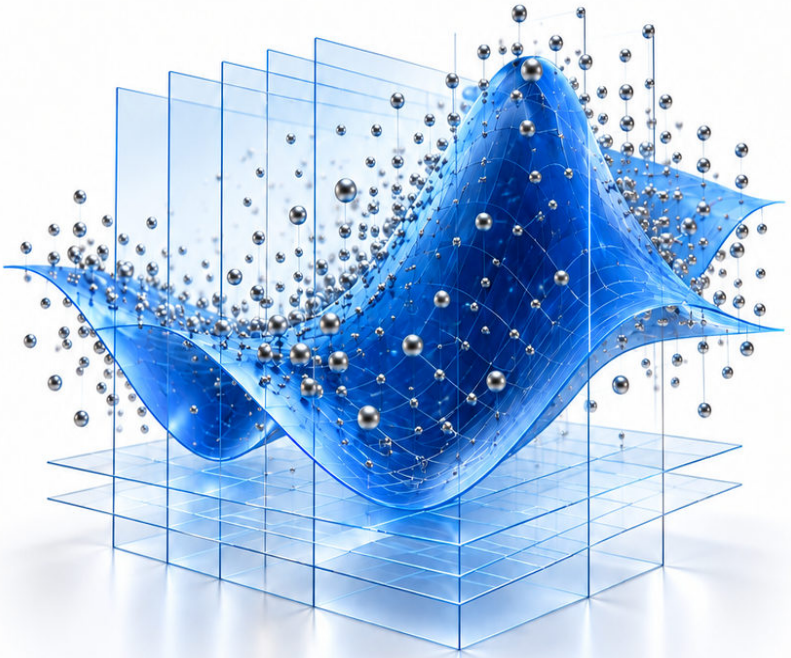
ArtificialGate

Modern

Machine Learning

Practical Course Notes



Dr. Barak Or

2027 EDITION

Preface

I wrote Modern Machine Learning to connect mathematical reasoning with the decisions we make when building models. My aim is to help readers formulate useful problems, understand why methods work, recognize their limits and judge whether an apparent improvement is real.

This book is for students, researchers and practitioners with basic Python, linear algebra, probability and differentiation. It supports independent study while developing the mathematical depth needed for advanced work.

Classical methods are useful tools in their own right and a foundation for deep learning. Across prediction, representation, uncertainty, neural networks and text, explanations and proofs sit alongside explicit algorithms, examples and experiments on established datasets. Companion notebooks and simulators turn these ideas into controlled investigations.

Read actively: trace a derivation, question an assumption, reproduce an example and explain what changes under new conditions. A favorable metric is only a beginning; the goal is evidence you can interpret and defend. I hope this book helps you develop that habit and carry it into your own work.

Dr. Barak Or

About The Author



Dr. Barak Or is an AI researcher, entrepreneur and lecturer specializing in machine learning, deep learning, autonomous systems and Physical AI.

He founded ArtificialGate and is founder and CEO of STATE16. He is Academic Director of the AI Model Development and Deep Learning program at the Google-Reichman Tech School and lectures at the Technion and Reichman University. He holds a University of Haifa Ph.D. (2022) and three Technion degrees: an M.Sc. in Aerospace Engineering (2018), a B.Sc. in Aerospace Engineering (2016), and a B.A. with honors in Economics and Management (2016).

Full biography: artificialgate.ai/barak-or/

How To Study This Book

Start with each chapter's problem and capabilities. Follow the Essential path in the reading map, reproduce the worked example and try the lab task. Add Recommended sections and Advanced proofs as needed.

Use the mathematical refresher in Chapter 1 to review vectors, probability and differentiation. Chapters 1 and 2 develop the guarantees for generalization and optimization. Read supporting proofs within the chapter that introduces each result. Case studies and references close each chapter.

Exercises, questions, prerequisite diagnostics, solutions and video scripts are in the separate aids. Attempt Practice before opening Solution. Implementation assertions check a contract, not your explanation or ability to solve a changed task. Follow the guide's sequence and required evidence for all three capabilities.

The 24 notebook exercises and 12 real-data case studies have matched solutions. Thirty-six short reading lessons orient the work.

Case Studies report measured runs, not promised scores; splits, seeds and software can change results. This 2027 edition records selected source and implementation checks through 14 October 2026, not future coverage. Independent academic review and learner-outcome validation remain outstanding.

Mathematical refresher in Chapter 1 | Open the separate course aids

Keep the supplied book and aids folders together. If the companion is missing, request it from the sender. The public course hub may contain an earlier practice release.

Displayed numerical results use at most three decimal places. Calculations and comparisons use unrounded values, so rounded components need not recombine exactly. Small nonzero quantities use scientific notation.

2027 edition. All rights reserved. Third-party software, models and datasets retain their respective licenses.

Reading levels

Essential Read first **Recommended** Explore **Advanced** Go deeper

Chapters 10-12 each have an Essential path for their topic.

Reading Map And Conventions

Machine learning infers structure from data. This book develops prediction, representation, uncertainty and experimental judgment, with classical methods preparing the transition to deep learning. It is not a systematic treatment of reinforcement learning, causal inference, structured sequence prediction or large-scale generative-model training. For those follow-on subjects, carry forward the same questions: what information is available, which objective is optimized, and what evidence supports generalization?

Basic Python, linear algebra, probability and differentiation are assumed. Applications develop graduate-level understanding; independent-study guidance follows.

Chapters 1-4 cover risk, preprocessing, optimization, prediction and evaluation; 5-8 develop representations, neighbors, kernels, ensembles and clustering; 9-10 address diagnosis, uncertainty and data acquisition. Chapter 11 applies classical methods to text. Chapter 12 develops neural-network foundations and a compact complete training example.

Sections explain concepts, define notation and give examples; theoretical results include assumptions and proofs or derivations. Main equations are numbered within chapters; T-series equations develop theory. The B-series adds bridge derivations; the C-series adds worked calculations. Independent formulas have separate displays and numbers; long identities may span aligned lines at unchanged type size.

Reading key

◇ Theory: assumptions, statement and mathematical result

The diamond marks a theorem, proposition or derivation; assumptions delimit its claim.

□ Proof: the argument supporting a result

An open square introduces proof or derivation; a final small square marks completion.

▶ Example: a concrete use of the idea

The triangle marks examples. Algorithms specify inputs, outputs and checks in numbered, indented pseudocode. Reading markers never replace mathematical conditions.

Contents

Select a chapter, section title or page number to go directly to that part of the book.

About The Author 2

How To Study This Book 3

Reading Map And Conventions 4

Notation and conventions 10

01. ML Problems, Data, And Generalization 15

1.1 Define the prediction problem before selecting an algorithm 16
1.2 Population risk, empirical risk and the best squared-error prediction 18
1.3 Generalization requires a disciplined data lifecycle 22
1.4 Dimensionality changes geometry and data requirements 26
1.5 Bias, variance and inductive bias 28
From A Finite List To A Continuous Linear Model 37
Iris 42
Case Study: Iris: write the prediction contract before chasing accuracy 43
Diabetes progression 45
Laboratory 01 | ML Problems, Data, And Generalization 46
Mathematical Refresher 48
Finite-Class Generalization With Bounded Loss 51
Local Coverage In A Uniform Hypercube 56

02. Regression, Optimization, And Regularization 59

2.1 Preprocessing is part of the fitted model 60
2.2 Least squares and its Gaussian likelihood interpretation 64
2.3 Gradient descent as a discretized gradient flow 70
2.4 Regularization controls the fitted solution 80
2.5 Report errors in a form the application can use 88
Case Study: Diabetes: connect the ridge objective to held-out error 92
Laboratory 02 | Regression, Optimization, And Regularization 94
Smooth Descent, Convex Progress And Nonconvex Limits 96

03. Classification And Imbalanced Data 102
3.1 Logistic regression models a conditional probability 104
3.2 Bernoulli likelihood leads to binary cross-entropy 105
3.3 Confusion matrices and ranking metrics answer different questions 114
3.4 Thresholds connect predictions to decisions 128
3.5 Handle imbalance without rewriting the evaluation population 134
Wisconsin Diagnostic Breast Cancer (WDBC) 142
Case Study: WDBC: separate ranking, probability quality and decision cost 143
What Zero Observed Misses Can Still Hide 145
Laboratory 03 | Classification And Imbalanced Data 146
ROC Area Is Pairwise Ranking Accuracy With Half-Credit For Ties 148

04. Validation And Model Selection 152
4.1 Holdout, K-fold, stratification and leave-one-out 153
4.2 Group and temporal splits must match the intended use 158
4.3 The validation boundary surrounds the entire pipeline 160
4.4 Search selects a model; nested validation evaluates that selection 162
4.5 Uncertainty, reproducibility and the final test lock 170
California Housing 178
Case Study: California Housing: random interpolation is not northward transfer 179
Wine Recognition 181
Laboratory 04 | Validation And Model Selection 182
The Estimand And Dependence Structure Of Cross-Validation 184

05. Representations, PCA, And PLS 187
5.1 Representations, scale, and numerical conditioning 189
5.2 PCA as variance maximization and reconstruction 192
5.3 Choosing dimension for reconstruction or prediction 202
5.4 Partial least squares uses target information 204

PLS1 From Residual Directions To New Predictions 207
Red Wine Quality 212
5.5 t-SNE is a neighborhood visualization 213
Case Study: Red Wine Quality: reconstruction variance versus predictive signal 219
Laboratory 05 | Representations, PCA, And PLS 221
The First PLS Covariance Direction Is A Singular-Vector Problem 223

06. Nearest Neighbors and Support Vector Machines 225
6.1 Nearest neighbors and the meaning of distance 227
6.2 Maximum margins and soft constraints 229
6.3 Duality, KKT conditions, and support vectors 233
Example: Preserve The Dual Constraint 239
6.4 Kernels and nonlinear support-vector classification 241
From Binary Margins To A Multiclass SVC 247
Digits · scikit-learn/UCI 8×8 copy 247
6.5 Support-vector regression and practical model choice 250
Case Study: Digits: verify a margin, then compare local and kernel prediction 257
MNIST handwritten digits 259
From Error Counts To Paired Evidence 260
Laboratory 06 | Nearest Neighbors and Support Vector Machines 262
Deriving The Soft-Margin Dual And Its Box Constraints 264

07. Trees And Ensemble Learning 266
7.1 Regression trees as recursive partitions 267
7.2 Classification impurity and complexity control 272
Example: Grow, Then Prune The Weakest Link 276
7.3 Bagging and random forests 278
7.4 Gradient boosting follows the loss gradient 281
7.5 Voting and leakage-free stacking 289

Adult Census Income 295
Case Study: Compare tree ensembles on historical Adult census records 296
Laboratory 07 | Trees And Ensemble Learning 297

08. Clustering And Anomaly Detection 300
8.1 K-means and alternating optimization 301
8.2 Evaluating cluster structure without inventing labels 309
8.3 Hierarchical clustering and linkage choices 311
8.4 DBSCAN and density connectivity 314
8.5 Anomaly scores, thresholds, and drift 320
Case Study: Discover chemical groups without choosing them from cultivar labels 324
Laboratory 08 | Clustering And Anomaly Detection 326
Ward Linkage Is An Exact Increase In Squared Distortion 328

09. Probability Calibration And Model Diagnosis 330
9.1 Proper scores and probability information 331
9.2 Estimating and correcting calibration 334
Temperature Scaling: What Changes And What Does Not 341
9.3 Error slices and conditional reliability 343
9.4 Interpreting predictive dependence 346
9.5 Distribution shift and the limits of uncertainty guarantees 349
Case Study: Audit probability claims on Wisconsin diagnostic measurements 360
Laboratory 09 | Probability Calibration And Model Diagnosis 362

10. Gaussian Processes And Experiment Selection 364
10.1 A distribution over related function values 365
10.2 RBF and Matérn kernels encode smoothness 368
10.3 Conditioning and interpreting posterior uncertainty 371
Example: Two Observations, Triangular Solves, And One New Label 375
10.4 Active learning asks which label improves prediction 377

10.5 Bayesian optimization asks where the objective is high 383
Case Study: Replay a limited-label GP study on real diabetes measurements 392
Laboratory 10 | Gaussian Processes And Experiment Selection 394
Expected Improvement From Its Defining Integral 396
The Marginal Likelihood Balances Residual Fit And Covariance Volume 398

11. Machine Learning for Text 401
11.1 From documents to count vectors 402
11.2 TF-IDF reweights lexical evidence 404
11.3 Naive Bayes, logistic regression, and linear SVMs 413
11.4 Optional bridge: fixed pretrained representations 418
11.5 Compare representations fairly and inspect failures 419
20 Newsgroups · by-date distribution 423
Case Study: Classify real newsgroup topics without metadata shortcuts 424
Laboratory 11 | Machine Learning for Text 426
SMS Spam Collection 426
Additive Smoothing Is A Posterior Predictive Calculation 429

12. Neural Networks: Foundations and Practice 432
12.1 From linear models to a neural network 433
12.2 Activation functions and learned features 436
12.3 From prediction error to backpropagation 443
12.4 Checking derivatives independently 450
12.5 From a correct gradient to a trained predictor 454
Case Study: Verify a handwritten network on actual MNIST threes and eights 459
Laboratory 12 | Neural Networks: Foundations and Practice 461

Conceptual Glossary 463

Subject Index 468

Notation and conventions

Definitions precede use. Section-opening definitions introduce symbols needed in the prose; equation definitions precede their display; proof notation is introduced before the assumptions and statement. Algorithms and figures have their own introductions where their notation is distinct. The same letter can have different roles in different declared scopes, so a familiar glyph does not establish its meaning.

The convention table defines generic operands. Local definitions specify each data or model quantity, its dimensions, index ranges and units. A definition introduces its left-hand symbol.

Shared chapter notation remains in force within its explicitly stated scope. Local definition blocks introduce new or changed meanings; unchanged shared definitions are not printed again before every display.

Common mathematical conventions

$\mathbb{R},\ \mathbb{N},\ \mathbb{Z}$	The real numbers, positive integers, and all integers, respectively. A range explicitly beginning at zero includes zero regardless of the natural-number convention.
n, d, i, j	Generic integer dimensions and indices in this convention table: n and d are positive sizes, while i and j select valid entries. Later sections define their local roles and ranges before use.
$a, b, c,\ \theta$	Generic scalar operands and a generic parameter in the convention table; these placeholders acquire a model-specific meaning only when explicitly introduced in a later section.
$v, w,\ A, B$	Generic column vectors v and w, and generic matrices A and B, of compatible dimensions. Capitalization alone does not assign a role: local definitions control every later use.
X, Y, Z, U, V	Generic random quantities in the probability conventions. A random variable is a function of the experiment's outcome; its observed numerical value is a realization. Later sections specify dimensions and distributions.
$\mathcal{A}, \mathcal{B},\ S$	Generic sets; in probability notation, calligraphic A and B are events. S denotes a set when used in the set and cardinality conventions below.

f, g, J	Generic functions f and g, and a scalar-valued objective J. Their arguments, domains and statistical meanings are specified locally in each application.
$=,\ \neq,\ \approx,\ \equiv$	Equality, inequality, numerical approximation, and identity or explicitly stated equivalence. Approximation is not an exact mathematical identity.
$+,\ -,\ \times,\ /$	Addition, subtraction or unary negation, multiplication, and division. Division requires a nonzero denominator. Matrix dimensions joined by a multiplication sign specify rows by columns rather than a product to evaluate.
$a^b,\ \sqrt{a},\ \lvert a\rvert$	A scalar power, the nonnegative square root of a nonnegative scalar, and absolute value. Parentheses determine the scope of operations; exponents can instead label an iterate when this is stated locally.
$<,\ \leq,\ >,\ \geq$	Scalar strict and non-strict order comparisons. An array inequality is componentwise only where that convention is explicitly stated.
$\in,\ \notin,\ \subseteq,\ \varnothing$	Membership, non-membership, set inclusion allowing equality, and the empty set.
$\cup,\ \cap,\ \setminus,\ \lvert S\rvert$	Set union, set intersection, set difference, and the number of members of a finite set S. Vertical bars around a set denote its cardinality, not a numerical absolute value.
$\{a_i\}_{i=1}^{n},\ \forall,\ \exists$	An indexed collection of n objects, followed by the quantifiers 'for every' and 'there exists'. Braces can describe a set by listing elements or by stating a membership condition.
$(a, b),\ [a, b],\ \infty$	Open and closed real intervals with endpoints a and b, and infinity. A bracket includes an endpoint; a parenthesis excludes it. Parentheses also denote tuples or function arguments when the context explicitly identifies them.
$\Rightarrow,\ \Leftrightarrow,\ \to$	Logical implication, equivalence in both directions, and an arrow denoting a map, a limit, or an update direction as specified locally.
$\mathbb{R}^{n\times d},\ \mathbb{R}^{d}$	Real matrices with n rows and d columns, and real column vectors with d components. A function's domain and codomain are its allowed input and output spaces.
$v_i,\ A_{ij},\ A_{:,j}$	Component i of vector v, entry in row i and column j of matrix A, and the whole column j of A. A colon in an array index selects all entries along that axis.

$A^T,\ A^\top,\ A^{-1}$ Two equivalent transpose notations, followed by the inverse of a nonsingular square matrix. An inverse in a formula does not prescribe forming an explicit inverse in numerical code.

$I_d,\ 0,\ \mathbf{1}$ The d-by-d identity matrix, a scalar or conformable array of zeros, and a conformable vector of ones. Local dimensions distinguish the scalar and array forms.

$v^T w,\ \odot$ The Euclidean inner product of equal-length real vectors, and componentwise multiplication of equal-shaped arrays. Ordinary matrix multiplication is written by juxtaposition and requires compatible inner dimensions.

$\|v\|_1,\ \|v\|_2$ The sum of absolute components and the Euclidean norm, respectively. The Euclidean norm is the square root of the sum of squared components. An unqualified vector norm in this book means the Euclidean norm unless a different norm is stated.

$\|A\|_F,\ \operatorname{tr}(A)$ The Frobenius norm, the square root of the sum of squared matrix entries, and the trace, the sum of diagonal entries of a square matrix.

$\det(A),\ \operatorname{rank}(A),\ \operatorname{diag}(v)$ The determinant of a square matrix, the dimension of a matrix's column space, and the diagonal matrix whose diagonal is vector v. Bars around a square matrix conventionally denote its determinant, which can be negative; bars around the scalar determinant denote its absolute value.

$A \succeq 0,\ A \succ 0$ For a real symmetric matrix, positive semidefinite and positive definite: the quadratic form is nonnegative for every vector, or strictly positive for every nonzero vector, respectively.

$\sum_{i=1}^{n},\ \prod_{i=1}^{n},\ \cdots$ Sum and product over the integer indices from one through n, including both endpoints, and an ellipsis continuing a stated pattern. An omitted index range is the range explicitly established for that calculation.

$\min,\ \max,\ \arg\min,\ \arg\max$ Minimum and maximum objective values, and the argument values attaining them. A constrained optimization also states its feasible set. Multiple optimizers can exist; an arg operator need not return a unique value.

$\mathbb{E}[Z],\ \mathrm{Var}(Z)$ Expectation of the random quantity Z, and its variance: the expected squared deviation from its mean when the relevant moments exist. A subscript on expectation identifies the averaging distribution or source of randomness.

$\mathrm{Cov}(U, V)$ Covariance of random quantities U and V. For vectors, entry i,j is the covariance between their corresponding components, with dimensions determined by their vector lengths.

$\mathbb{P}(\mathcal{A} \mid \mathcal{B}),\ \Pr(\mathcal{A})$ Conditional probability of event A given event B, requiring event B to have positive probability, and probability of event A. The probability notation P, blackboard P or Pr has the same operator role when locally identified. The vertical bar denotes conditioning; a semicolon separates a parameter from an argument where stated.

$X \sim P,\ \mathrm{i.i.d.}$ Random quantity X has distribution P. The abbreviation i.i.d. means independent and identically distributed; both independence and the common distribution are assumptions, not consequences of collecting a dataset.

$\mu,\ \Sigma,\ \mathcal{N}(\mu, \Sigma)$ Mean vector mu, covariance matrix Sigma, and a normal distribution with those parameters, in this convention. In one dimension the second normal-distribution argument is a variance, not a standard deviation.

$\mathbf{1}\{\mathcal{A}\},\ \mathbf{1}[\mathcal{A}]$ Equivalent indicator notations: one if the event or statement A holds, and zero otherwise. This differs from an unqualified bold one denoting a vector of ones.

$\log,\ \ln,\ \exp,\ e$ The natural logarithm, its alternative notation ln, the exponential function, and its base e. A different logarithm base is written explicitly. Logarithms of probabilities give dimensionless scores; negative natural log probabilities are measured in nats.

$f'(a),\ \frac{df}{da},\ \frac{\partial J}{\partial \theta_j}$ A univariate derivative evaluated at a, derivative notation, and a partial derivative of J with respect to parameter component j while other components are held fixed. A prime can instead label a second observation only if explicitly defined that way.

$\nabla_\theta J,\ \nabla^2_\theta J$ Gradient, the column vector of first partial derivatives of scalar objective J, and Hessian, the matrix of its second partial derivatives with respect to the parameter vector theta.

$\hat{a},\ \bar{a},\ \tilde{a},\ a^*$ Common modifiers for an estimate, an average, a transformed quantity, and an optimum or distinguished reference value. Each application explicitly states its actual meaning; the mark alone does not define an estimator or guarantee optimality.

$a^{(t)},\ a_{t+1},\ \leftarrow$ An iterate indexed by integer step t, the next indexed value, and algorithmic assignment. Parenthesized iteration superscripts are not exponentiation. An asterisk in programming pseudocode denotes multiplication when not used as a declared symbol modifier.

$\int_a^b f(u)\,du$ The definite integral of function f over the scalar integration variable u between endpoints a and b. The differential du identifies the integration variable; the letter d here is the differential operator, not a feature dimension.

$\propto,\ O(h^2),\ \lim_{h\to 0}$ Proportionality by a factor independent of the argument of interest; an error bounded in magnitude by a constant times h squared for sufficiently small positive h; and a limit as the scalar increment h tends to zero. Local text identifies the variable and limiting regime.

$\lfloor a \rfloor,\ \lceil a \rceil,\ \operatorname{sign}(a)$ The greatest integer no larger than a, the least integer no smaller than a, and the sign function, with values negative one, zero and positive one for negative, zero and positive inputs.

$\pi,\ \%,\ \delta_{ij}$ The circle constant in formulas such as a normal density; percentage as a quantity per hundred; and the Kronecker delta, one when the two indices agree and zero otherwise. A policy, probability or other use of pi is explicitly redefined locally.

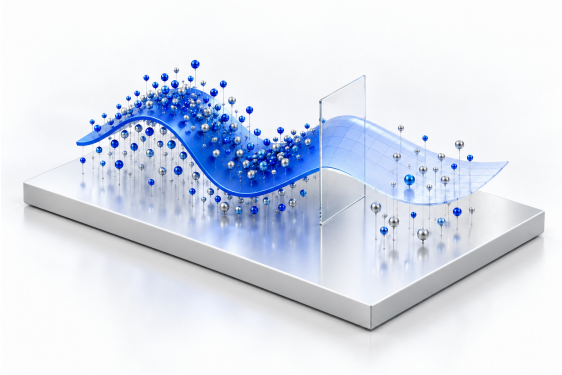
Data and numerical conventions

Training data estimate parameters; validation data guide model selection; a final test evaluates a frozen procedure. Scaling, imputation, vocabulary construction and representation learning are fitted within the relevant training boundary. Loss functions explicitly distinguish sums from means. Library regularization parameters need not use the same numerical convention as the displayed equations.

Features and targets retain the units of the defined application unless transformed. Standardized features are dimensionless. A probability is dimensionless and lies between zero and one. Numerical examples state their assumptions; a good fit under those assumptions is not a physical or causal guarantee.

CHAPTER 01

ML Problems, Data, And Generalization



Reading path

Start with Essential. The other levels offer more depth.

Essential
1.1 1.2 1.3 1.5

Recommended
1.4

Advanced
Proofs 1.2 1.4

Worked example Lab task

A model can fit every recorded example and still answer the wrong question. Begin with what will be predicted, when the information exists, and what future observations should resemble.

What You Will Be Able To Do

- Translate an application into a prediction contract with a target, unit of observation, prediction time and evaluation population.
- Compute empirical risk and explain why it differs from population risk.
- Measure the effects of dimensionality and model flexibility with controlled synthetic experiments.
- Interpret bias, variance and noise without treating training fit as proof of generalization.

Before you begin: Python functions and arrays; Means, variances and conditional probability; Vectors and elementary derivatives.

Define the prediction task, compute risk and separate the data roles. Work through the bias-variance decomposition before the optional coverage proofs.

Algorithm scope: Conceptual experiment protocol: Algorithm 1.1 specifies data roles, selection and a frozen test; Fit_h delegates model fitting. Generalization results retain their stated assumptions.

Shared chapter notation

d	Positive represented-feature count; features need not be independent. In the unit-hypercube example it is the dimension.
r	Dimensionless side length of a fully contained axis-aligned subcube of the unit hypercube; between zero and one.
c	Arbitrary real constant prediction at a fixed input; the candidate optimized under squared loss.
$v(x)$	Conditional variance of a fresh scalar target at input x, where x is the fixed input value; may vary with x and is the irreducible squared-error component relative to that input.

Chapter notation

n	Positive observation count for the dataset under discussion; its training or evaluation role is stated locally.

A learned pattern is only one part of a prediction system. Forecasting services, document classifiers and image recognizers also need a precise target, information available at prediction time, and evaluation that resembles intended use. These requirements apply across industries and dataset sizes, from hundreds of observations to millions.

This chapter separates fitting observed data from performing well on observations not used to make choices. Statistical risk, data boundaries, dimensionality and repeated-training experiments develop that distinction. Simple baselines and controlled comparisons make the reasoning testable; the mathematics does not automatically certify a deployed system.

- *Essential*

1.1 Define the prediction problem before selecting an algorithm

Start with the information available for learning. Supervised learning pairs inputs with observed targets: regression predicts a quantity, such as journey duration; classification predicts a category or its probability, such as whether a message is unsolicited. Unsupervised learning studies structure without a designated target. Other settings add feedback or interaction. The distinction is the training information and objective, not model complexity or how intelligent an application appears.

Before fitting, write a prediction contract: one row represents what; prediction occurs when; the target covers which horizon; deployment concerns which population; and the prediction supports which action. Then mark each feature's availability time and each row's underlying entity. Visits from one person or

orders from one restaurant are not automatically independent evaluation units. These two checks determine which features are admissible and which rows may cross a validation boundary.

Read a column's meaning and availability before its numbers: a category code, identifier or post-outcome field is not automatically a usable measurement. Features need not be statistically independent. The design matrix has n observations as rows and d features as columns. An observation may still be written as a column vector; transpose it to place it in its matrix row.

Vapnik's statistical learning framework [1.1] and the formulation of learning problems by Shalev-Shwartz and Ben-David [1.3] separate the data-generating process, hypothesis class, and loss. This separation makes a prediction task precise before an algorithm is selected.

Notation for equations (1.1), (1.2), (1.3)

D	Observed input-target dataset; records may be dependent or share entities.
$n,\ i$	Observation count and index from one through that count.
x_i	Observation i's real input column, one coordinate per represented feature.
$y_i,\ \hat{y}_i$	Observed target and model prediction for the same observation.
X	Design matrix (n by d); its complete row i,: is the transpose of input column x_i .
$f_\theta,\ \theta$	Prediction function and fitted parameters; maps represented inputs to the target space.
$\mathbb{R}^d$	Real column vectors with d coordinates; two indices instead specify a real matrix space.

$$D = \{(x_i, y_i)\}_{i=1}^{n}, \qquad x_i \in \mathbb{R}^d \tag{1.1}$$

$$X \in \mathbb{R}^{n \times d}, \qquad X_{i,:} = x_i^T \tag{1.2}$$

$$\hat{y}_i = f_\theta(x_i) \tag{1.3}$$

D contains n examples. The transpose of the column vector x_i occupies row i of X. The model parameters theta determine predictions; d describes the representation rather than an independence assumption.

Here X is a design matrix. In the following risk formulas, capital X instead denotes a random input vector.

▶ Example: A delivery estimate with a hidden shortcut

For delivery duration predicted at order acceptance, first list what is known then: distance and quoted preparation time qualify. The actual dispatch timestamp is created later, so exclude it even if it predicts duration accurately. Next choose the evaluation population. Future orders at known restaurants require a time-respecting design; orders at unseen restaurants require withholding restaurants. Repeated customers also need attention if the contract concerns new customers. A random row split answers neither question automatically. The final contract records acceptance time, the duration target, admissible features, the intended population and the action supported.

Interpretation and limits

A feature is not legitimate merely because it appears in the historical table. Check when and how it was generated relative to the prediction event.

Essential

1.2 Population risk, empirical risk and the best squared-error prediction

A loss scores one prediction; risk averages that score across cases. Squared error penalizes the squared numerical difference, whereas absolute error penalizes its magnitude. Choose the loss to reflect the question and mistake costs. Population risk averages it over a specified joint distribution of inputs and targets. That is the problem we want to solve, but the distribution is ordinarily unknown.

Empirical risk replaces a population average by an observed mean. Training observations select the fitted parameters, so their loss measures fit to those observations, not an independent test of prediction. Flexibility can reduce that loss by fitting accidents of the sample. To assess transfer, freeze the learned procedure and evaluate separate observations drawn in a way that matches the contract. Separation prevents reuse; representative sampling addresses relevance. Neither condition substitutes for the other.

At a fixed input, compare any prediction with the conditional mean. Expanding squared error around that mean leaves two terms: conditional variance, which the prediction cannot change, and the nonnegative squared distance from the prediction to the mean. The latter is smallest at zero. Thus the conditional mean minimizes expected squared loss when the relevant second moment exists, without requiring a linear relationship. With a finite conditional first absolute moment, a conditional median minimizes absolute loss. The loss determines which feature to predict.

Hastie, Tibshirani, and Friedman [1.2] develop the decision-theoretic view behind the conditional-mean predictor under squared loss. Vapnik [1.1] distinguishes this population objective from the empirical criterion minimized on a finite sample. For a fixed predictor evaluated on independent observations with a common distribution and bounded loss, Hoeffding's inequality controls the probability that the sample mean loss differs from its population value [1.7]. Unbounded squared loss needs additional assumptions or a different argument; the bounded-loss finite-class proof in Chapter 1 makes that distinction explicit.

Notation for equations (1.4), (1.5)

f	Fixed predictor evaluated by population or sample-average loss.
$X,\ Y$	Jointly drawn random input vector and target from the evaluation population.
P	Joint input-target population distribution; the tilde denotes sampling from it.
L	Numerical penalty taking the observed target and prediction as arguments.
$R(f)$	Expected loss of the fixed predictor under the declared population.
$D,\ n,\ i$	Observed dataset, sample count, and observation index from one through that count.
$x_i,\ y_i$	Realized input and observed target of dataset observation i.
$\widehat{R}_D(f)$	Mean loss on the indicated training or independent-evaluation dataset.
$\mathbb{E}_{(X,Y)\sim P}$	Expectation over a fresh joint input-target draw from the specified population.

$$R(f) = \mathbb{E}_{(X,Y)\sim P}[L(Y, f(X))] \tag{1.4}$$

$$\widehat{R}_D(f) = \frac{1}{n}\sum_{i=1}^{n} L(y_i, f(x_i)) \tag{1.5}$$

R averages loss over the chosen population P. Empirical risk uses a mean over n observed examples. A training average and an independent evaluation average play different roles.

The population distribution must match the stated use case. A score on fitting data is not an independent generalization estimate.

Notation for equation (1.6)

$X,\ x$	Random input vector and fixed input value where predictions are compared.

Y	Scalar target conditional on that input, with finite conditional second moment.
$\mathbb{E}[\cdot \mid X = x]$	Expectation over targets at the fixed input; the dot is an argument placeholder.
$\mathrm{Var}(Y \mid X = x)$	Conditional target variance; independent of the candidate constant prediction.

$$\mathbb{E}[(Y - c)^2 \mid X = x] = \mathrm{Var}(Y \mid X = x) + (c - \mathbb{E}[Y \mid X = x])^2 \tag{1.6}$$

For a fixed x with finite conditional variance, the expression is minimized by c equal to the conditional mean. The remaining conditional variance cannot be removed by choosing a different constant at that x.

The conditional mean minimizes squared loss at the fixed input; no linearity assumption is required.

◇ Theorem 1.1: The conditional mean is the squared-loss Bayes predictor

Notation for this result

X, Y, x	Random input, scalar target, and a fixed input value; the input is not the design matrix here.
$m(x)$	Conditional mean target at the fixed input, also called the regression function.
ε	Target residual after subtracting the conditional mean; its conditional expectation given the input is zero.
$f, R(f)$	Candidate prediction function and its population squared-loss risk under the same joint distribution.

Assumptions. The target is a real random variable with finite second moment. Inputs and targets have a fixed joint distribution. Candidate predictors are measurable functions of the available input and have finite squared prediction risk. Conditional statements hold at inputs where the conditional moments are defined, hence almost surely.

The conditional mean uniquely minimizes conditional squared loss at almost every input. The excess population risk of any other predictor is exactly its mean squared distance from that conditional mean. No Gaussian-noise or linearity assumption is needed.

□ Proof *Recommended*

The proof needs zero residual mean at the input being considered, not merely zero mean across all inputs. Fix that input and subtract its conditional mean from the target. The resulting residual has conditional mean zero by definition. Unconditionally centered noise could still have positive mean at some inputs and negative mean at others; it would not justify this cancellation.

$$m(x) = \mathbb{E}[\,Y \mid X = x] \tag{1.T1}$$

$$\varepsilon = Y - m(X) \tag{1.T2}$$

$$\mathbb{E}[\varepsilon \mid X = x] = 0 \tag{1.T3}$$

$$v(x) = \mathbb{E}[\varepsilon^2 \mid X = x] \tag{1.T4}$$

Hold the input fixed and compare an arbitrary scalar prediction with the conditional mean. Expand their squared-error difference before averaging. The cross term multiplies a fixed prediction difference by a residual whose conditional mean is zero, so it disappears. This cancellation uses conditional centering, not a Gaussian-noise assumption.

$$(\,Y - c)^2 = \varepsilon^2 + 2\varepsilon(m(x) - c) + (m(x) - c)^2 \tag{1.T5}$$

$$\mathbb{E}[(\,Y - c)^2 \mid X = x] = v(x) + (m(x) - c)^2 \tag{1.T6}$$

The variance term does not depend on the prediction. The remaining square is nonnegative and equals zero only at the conditional mean. Applying the law of total expectation integrates this pointwise result over the input population.

$$R(f) = \mathbb{E}[v(X)] + \mathbb{E}[(m(X) - f(X))^2] \tag{1.T7}$$

$$R(f) - R(m) = \mathbb{E}[(f(X) - m(X))^2] \geq 0 \tag{1.T8}$$

Equal risks require the prediction to equal the conditional mean almost surely: exceptions may occupy only a set of total input probability zero. Changing predictions on that set cannot affect expected loss, so risk does not identify those values. The argument uses conditional expectation and expansion of a square, not differentiation under an integral; finite second moments suffice.

□

Separate what should be predicted from what the chosen model can represent. The conditional mean is the squared-loss target, but a linear model may not contain it; running optimization longer cannot enlarge that model class. Likewise, irreducible variance means irreducible given the available inputs. Adding an informative measurement can reduce it, so the term is not a universal physical limit.

▶ Example: Choose a constant using the loss

At one input, targets zero, two and six have probabilities 0.2, 0.5 and 0.3. Their weighted mean is 2.8 and the weighted mean of their squares is 12.8, giving variance 4.96. Predicting two instead gives squared-error risk 5.60. Its excess, 0.64, is exactly the squared distance from two to the mean. Under absolute loss, two is a median and its risk is 1.6. The same declared distribution therefore has different optimal predictions under different losses; these are calculations, not measured model performance.

Interpretation and limits

A random row split is not automatically an independent evaluation: related measurements from one subject can cross the boundary. Match the split to the unit that will be new at prediction time, then state the population to which the score applies.

Essential

1.3 Generalization requires a disciplined data lifecycle

Give each data partition a distinct job: training fits parameters, validation guides development, and testing evaluates the frozen choice. These are roles, not prescribed percentages. Repeated validation choices can adapt to chance features of that set. Test results likewise cease to be a clean final check if they change features, thresholds, architecture or the decision to keep searching.

Choose a baseline for the declared loss: the training mean for squared error, a training median for absolute error, class frequencies for probability forecasts, or a majority-class rule for accuracy. Compare it with the proposed model on identical evaluation examples and metrics. Record the modeling change before comparison. Greater complexity must earn its cost through evidence, but a valid experiment may show no gain; that outcome can expose weak signal, inadequate data or an unsuitable inductive bias.

Distribution mismatch complicates generalization even when the code never touches test labels. Randomly mixing future observations into training can disguise change over time. Sharing users across splits can exaggerate performance on unseen users. Deployment can further change behavior,

measurement systems or target prevalence. The prediction contract therefore governs splitting and interpretation. Keep the test boundary intact while using diagnostics on training and validation data to improve the experiment.

Shalev-Shwartz and Ben-David [1.3] explain why fitting observations and generalizing to new observations are different requirements. The training-validation-test separation used here makes that distinction operational; it does not remove the need for representative sampling. Choosing a predictor after seeing the sample requires more than a concentration statement for one predictor fixed in advance. Vapnik and Chervonenkis studied uniform convergence across classes of events, providing a foundation for controlling such selection in classification under appropriate class-complexity and sampling conditions [1.8].

Why comparing more candidates costs evidence

A holdout score contains sampling fluctuation. Choosing the smallest of several scores can therefore select a favorable fluctuation as well as a good predictor. The proof controls this by bounding every candidate's estimation error simultaneously, for a fixed finite collection, bounded loss, and independent, identically distributed evaluation observations. Comparing the selected candidate with the best candidate uses the allowance twice: once for each score. An imperfect empirical minimizer adds its optimization tolerance.

Check the independence boundary before using the bound. Either choose the candidate collection without the scoring sample or specify a fixed family large enough to include all candidates considered. Counting only the fitted models tried on their own training data does not establish either condition. See the proof for sample size and the refresher for probability.

This guarantee supplements, rather than removes, separate development and final evaluation.

Hoeffding [1.7] treats independent, bounded summands. Read this alongside Shalev-Shwartz and Ben-David [1.3], Section 4.2: a deviation bound for one fixed predictor becomes a finite-class guarantee by controlling all candidates together. Choosing the predictor after seeing the sample changes the question. For this chapter, identify the candidate class and sampling assumptions before invoking a bound; a small training error alone supplies neither independence nor selection control.

Read the finite-class generalization proof

Notation for equations (1.7), (1.8)

D_{train}	Records for parameter fitting and all learned preprocessing in the simple holdout workflow.

$D_{\text{validation}}$	Separate development records for decisions such as model selection and threshold tuning.
D_{test}	Records reserved for final evaluation after development choices are frozen.
$\cap,\ \cup,\ \varnothing$	Intersection, union, and empty set; disjoint partitions share no records.

$$D_{\text{train}} \cap D_{\text{validation}} = \varnothing \tag{1.7}$$

$$D_{\text{test}} \cap (D_{\text{train}} \cup D_{\text{validation}}) = \varnothing \tag{1.8}$$

Disjoint observation sets are necessary for this simple workflow but are not sufficient: shared entities, time overlap and information derived from future outcomes can still create leakage.

Indexing identifies records; it does not certify that their information is independent.

▶ Example: A model-development ledger

Split 1,000 eligible records into 600 training, 200 validation and 200 test observations according to the contract. Fit the mean baseline and polynomial candidates using only the 600; use the next 200 to choose a configuration; record it before opening the final 200. The ledger should identify which decisions each partition influenced. Choosing another polynomial after seeing test error adds a development decision to the test column. Report that reuse and obtain a new independent final evaluation. The counts illustrate roles, not universally recommended fractions; the test boundary protects a frozen procedure, not a preferred score.

Interpretation and limits

Disjoint row indices do not guarantee independent evidence. Duplicated records, related entities and future information can cross a boundary without sharing an index.

Algorithm 1.1. A reproducible baseline experiment

Source and adaptation: Hastie, Tibshirani and Friedman [1.2], Chapter 7, and Shalev-Shwartz and Ben-David [1.3], Chapter 4. This is the book's experimental protocol for fixed data roles, a declared finite comparison and a frozen final test; model fitting remains delegated.

Notation and assumptions

$D = \{(x_i, y_i)\}_{i \in I}$	Eligible indexed observations; x_i is the available input and y_i the target. D_I restricts the data to indices I.

$I_{\mathrm{tr}}, I_{\mathrm{val}}, I_{\mathrm{te}}$	Nonempty, disjoint training, validation and final-test index sets.
$\mathcal{H},\ h,\ \ell$	Nonempty finite set of planned baseline configurations, its member, and the declared per-observation loss.
$f_h,\ \hat{f},\ \widehat{y}_{\mathrm{te}}$	Training-fitted pipeline for configuration h, selected frozen pipeline, and final prediction vector.
$\widehat{R}_I(f) = \lvert I\rvert^{-1} \sum_{i \in I} \ell(y_i, f(x_i))$	Mean loss on a nonempty index set I; the bar-like hat denotes an empirical quantity.
$q_h,\ h^*,\ \mathcal{P}$	Validation loss, selected configuration, and provenance record containing splits, seed, versions, population and limitations.

Fit_h clones and fits every learned transformation and predictor using only the indexed training records. Split respects the declared groups and chronology. Select chooses a minimum validation loss with the predeclared tie rule. Declare the target, positive class when relevant, sampling unit and evaluation population before fitting.

Algorithm 1.1

Input: Eligible data and prediction-time feature contract. D

Loss, split rule and finite comparison plan. $\ell,\ \mathcal{H}$

Seed, software versions and deterministic tie rule.

Output: Frozen fitted pipeline and provenance. $\hat{f},\ \mathcal{P}$

Final mean loss, record count and independent-unit count.

Declare Target, loss, population, available features and planned comparisons.

Split $(I_{\mathrm{tr}}, I_{\mathrm{val}}, I_{\mathrm{te}}) \leftarrow \mathrm{Split}(D)$

Check Feature timing, nonempty partitions, grouping and chronology.

Lock Final-test outcomes until the complete procedure is frozen.

For $h \in \mathcal{H}$

  $f_h \leftarrow \mathrm{Fit}_h(D_{I_{\mathrm{tr}}})$

  $q_h \leftarrow \widehat{R}_{I_{\mathrm{val}}}(f_h)$

End for

$h^* \leftarrow \mathrm{Select}\big(\arg\min_{h \in \mathcal{H}} q_h\big)$

Freeze $\hat{f} \leftarrow f_{h^*}$

$\widehat{y}_{\mathrm{te}} \leftarrow (\hat{f}(x_i))_{i \in I_{\mathrm{te}}}$

Algorithm 1.1 (continued)

12 $\widehat{R}_{\text{te}} \leftarrow \widehat{R}_{I_{\text{te}}}(\hat{f})$

13 **Record** Splits, transforms, versions, row/unit counts and limits.

14 **Return** $\hat{f},\ \mathcal{P},\ \widehat{R}_{\text{te}},\ |I_{\text{te}}|$

Why this works. Retain the selected training fit; do not silently refit on validation. Store the independent-unit count in the provenance when justified by the sampling design. The plan makes finitely many fitting calls. A model change motivated by test outcomes needs new independent evidence for a new final claim.

Recommended

1.4 Dimensionality changes geometry and data requirements

Why can extra features make nearby examples harder to find? In a unit hypercube with independent uniform coordinates, a smaller aligned cube of side length r contains a volume fraction r raised to d. To retain the same fraction as d grows, its neighborhood must widen. This simplified geometry explains a difficulty for local averaging when many coordinates matter; additional features may still reveal valuable signal.

The example is not a universal theorem that high-dimensional learning fails. Real datasets may concentrate near a lower-dimensional structure, have sparse relevant variables, or admit representations with useful similarity. Strong regularization and appropriate models can work when d exceeds n. Conversely, a large sample can still be insufficient for a badly chosen representation. Effective dimensionality, noise, coverage and the task matter alongside the number of columns.

Treat feature addition as an experimental intervention. Ask whether a column is available, informative and stable; then evaluate its effect without breaking the split. Scaling determines how coordinates contribute to distance, while redundant variables can repeatedly emphasize one underlying factor. Feature selection and dimensionality reduction are themselves learned operations and belong inside the fitting process. Later chapters compare distance methods, principal component analysis (PCA), partial least squares (PLS) and learned representations using this same logic.

Bellman [1.5] identified the dimensionality problem that motivates the geometric examples in this section. Exponential growth in a grid is an

illustrative mechanism, not a claim that every high-dimensional learning problem requires exponentially many observations.

Notation for equation (1.9)

d	Positive integer dimension of the unit hypercube, with independent uniformly distributed coordinates.
q	Fraction of total hypercube volume occupied by the subcube, equal to its probability under uniform sampling.

$$q = r^{\,d} \qquad \Longrightarrow \qquad r = q^{\,1/d}, \quad 0 < q < 1 \tag{1.9}$$

For independent uniform coordinates in a unit hypercube, the aligned subcube occupies fraction r of each coordinate's range. Multiplying those fractions gives its volume fraction q. More coordinates compound the reduction. This illustration concerns uniform geometry; it is not a sample-complexity bound for every model or for data concentrated near a lower-dimensional structure.

All coordinates use unit intervals. This geometric illustration is not a universal sample-complexity bound.

◇ Proposition 1.2: Local coverage in a uniform hypercube

Notation for this result

d, j	Positive input dimension and coordinate index from one through that dimension.
r, q	Subcube side length and captured probability mass, each strictly between zero and one.
A, N_A	Fixed subcube and number of sampled inputs inside it.
n, i, X_i	Sample count, observation index from one through that count, and indexed random input.
$\mathbf{1}\{\cdot\}$	Indicator equal to one when the enclosed event occurs and zero otherwise.
$\Pr$	Probability over independent repeated uniform input samples.
δ	Allowed probability of an empty fixed neighborhood, strictly between zero and one.
$\lceil\cdot\rceil, \log$	Ceiling: smallest integer not below the argument; and natural logarithm, respectively.

Assumptions. The input has independent uniform coordinates on the unit interval. A fixed axis-aligned subcube lies entirely inside the unit hypercube. Training inputs are independent draws from this distribution. The neighborhood is fixed before observing those draws.

A neighborhood with fixed side length captures an exponentially decreasing population fraction as dimension increases. The exact probability of collecting at least one observation in it determines a corresponding sample requirement; this is a geometric statement under the specified sampling model, not a universal impossibility theorem.

Read the complete supporting proof . *Advanced*

At fixed neighborhood width the coverage burden grows exponentially with dimension, while at fixed captured mass the neighborhood becomes less local. Structured inputs, sparse relevant variables, or a learned low-dimensional representation can violate the uniform-cube assumptions and change this conclusion. Adding features is not automatically harmful; the question is whether they supply signal commensurate with their effect on geometry.

▶ Example: A neighborhood containing one tenth of the volume

For an axis-aligned cube inside a unit cube with uniform sampling, multiply its equal side lengths to obtain its volume fraction. Setting that fraction to 0.1 gives side length approximately 0.316 in two dimensions and 0.891 in twenty. Thus the same probability mass requires a region spanning more of each coordinate. Use the unrounded lengths when checking that their powers equal 0.1; rounded labels only approximate this identity. The notebook checks it by Monte Carlo sampling. This calculation concerns uniform full-dimensional geometry, not a claim that structured image data must defeat a classifier.

Interpretation and limits

Do not infer that d greater than n guarantees failure or that every added feature is harmful. Evaluate the model, representation and regularization together on suitable held-out data.

Essential

1.5 Bias, variance and inductive bias

Think of refitting the same procedure on repeated training samples. At a fixed input, predictions vary across samples, and their average may miss the conditional mean target. Under squared loss and finite second moments, expected error separates into squared bias, prediction variance and conditional noise variance. This requires fresh noise with conditional mean zero, conditionally independent of training. The expectation covers repeated training and fresh outcomes, not rows of one fitted model.

Bias describes a systematic mismatch of the average fitted prediction; variance describes sensitivity to the training sample. Model flexibility can affect both, but simplistic rules are unreliable. Additional data, regularization, optimization

choices and representations can change the balance. Interpolation of training examples is not by itself proof of either success or disaster. The relevant question is whether independent performance supports the claim, including any change in population or extrapolation outside observed coverage.

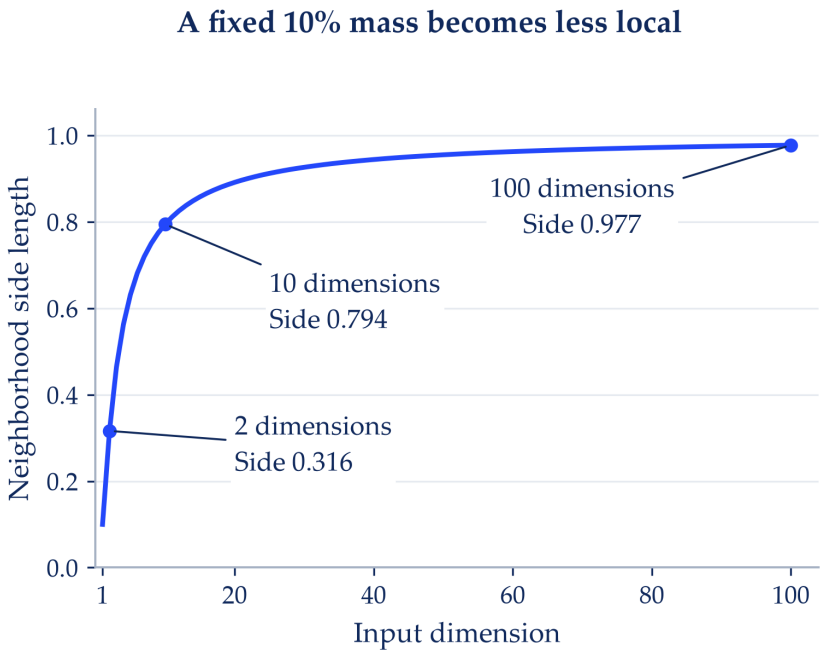


Figure 1.1. Exact side length needed for a fully contained axis-aligned neighborhood to hold 10% of a unit hypercube with independent uniform coordinates. As dimension increases, maintaining the same probability mass forces the neighborhood to span most of each coordinate. This is a geometric coverage calculation, not a universal learning-rate theorem.

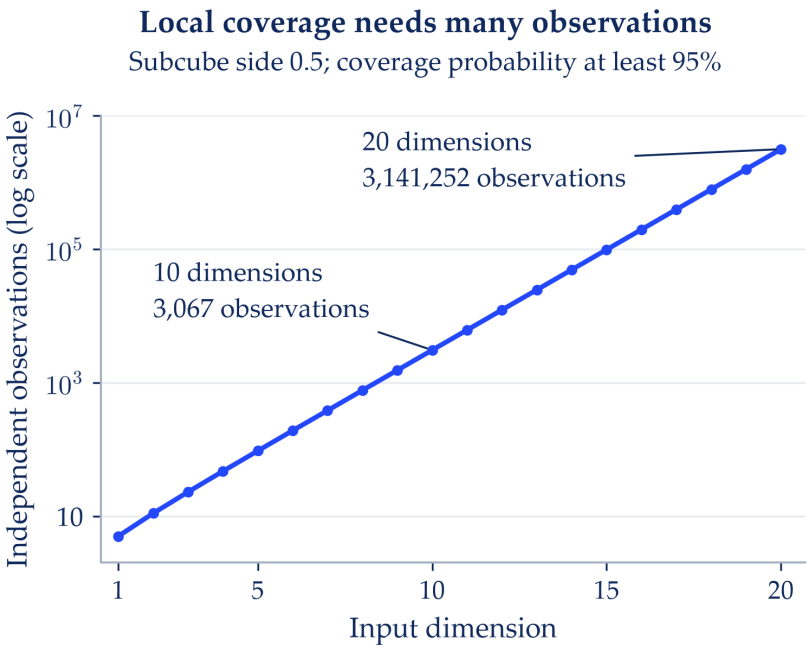


Figure 1.2. Exact smallest independent sample count giving at least 95% probability of one observation in a fully contained subcube of side 0.5. The probability of missing it in every draw is evaluated directly. A single local observation is not enough for accurate regression; the curve describes coverage only.

Inductive bias supplies preferences where the observations leave possibilities unresolved: linearity favors linear relations, locality favors nearby examples, and sparsity favors fewer active variables. Smoothness and shared parameters are further examples. These preferences can help across related tasks without implying failure everywhere else. No-free-lunch conclusions depend on how possible problems are averaged; they neither rank real-world models universally nor remove the need to compare task-appropriate assumptions using independent evidence.

Geman, Bienenstock, and Doursat [1.4] analyze the bias-variance dilemma, while Wolpert [1.6] shows why algorithm rankings cannot be distribution-free under his averaging assumptions. Together these works motivate inductive assumptions that are appropriate to the intended task rather than universally preferred models. Sensitivity to the training sample also has a formal role

beyond the bias-variance decomposition: Bousquet and Elisseeff relate controlled changes in a learning algorithm's loss when a training example is removed to generalization bounds [1.9]. This algorithmic stability is a stated mathematical property, not something established by observing a few similar fitted curves.

Bousquet and Elisseeff [1.9] require uniform loss stability, independent sampling and bounded loss for their generalization bound; a few perturbations cannot certify these premises. Theorem 22 supplies a separate stability result for a squared-norm-regularized kernel method, assuming a bounded kernel diagonal and controlled loss sensitivity. Stronger regularization tightens that stability bound, not necessarily prediction error.

Notation for equation (1.10)

D	Random training dataset redrawn under a fixed sampling protocol before refitting the same learner.
$X,\ x,\ Y$	Random input, fixed evaluation input, and fresh scalar target there.
$\hat{f}_D(x)$	Fixed-input prediction from the model fitted to the random training dataset.
$\mathbb{E}_{D,Y\|x}$	Expectation over repeated training datasets and fresh targets at fixed input.
$\mathrm{Bias}(x)$	Mean fitted prediction over datasets minus the conditional target mean at fixed input.
Var_D	Fixed-input prediction variance over independently repeated training datasets.
$\mathrm{Var}(Y \mid X = x)$	Fresh-target conditional variance: irreducible squared-error variation at that input.
$V(x)$	Variance Var_D of the fitted value at fixed input over repeated training.

$$\begin{aligned}\mathbb{E}_{D,Y|x}[(Y-\hat{f}_D(x))^2] = \mathrm{Bias}(x)^2 \\ + \mathrm{Var}_D(\hat{f}_D(x)) \\ + \mathrm{Var}(Y \mid X = x)\end{aligned} \tag{1.10}$$

Imagine repeatedly drawing a training dataset and refitting the same procedure at a fixed input x. Bias compares the mean of those fitted predictions with the conditional mean target. The new outcome's noise must be conditionally independent of the trained predictor given x. Under that assumption, squared error separates training-sample variability from fresh outcome noise.

Assume finite second moments and conditional independence of the fresh target noise from the trained predictor.

◇ Theorem 1.3: Bias, variance, and fresh-outcome noise

Notation for this result

X, x, Y	Random evaluation input, its fixed value, and a fresh scalar target at that value.
$D, \hat{f}_D(x)$	Random training object including algorithm randomness, and its resulting prediction at the fixed input.
m, a	Conditional target mean and mean fitted prediction across repetitions of the training procedure.
$\mathbb{E}_{D,Y\|x}$	Expectation over repeated training and fresh targets, holding the evaluation input fixed.
$\mathrm{Bias}(x)$, $V(x)$	Systematic mean-prediction error and prediction variance over repeated training datasets at the fixed input.
$\mathrm{Err}(x)$	Expected squared error over the training procedure and an independent fresh target at the fixed input.

Assumptions. Fix an evaluation input. Both the fresh target and the fitted prediction have finite conditional second moments. The fresh target is conditionally independent of the training dataset and any training randomness given the evaluation input. Training randomness is included in the random training object.

Average squared prediction error over repeated fitting and fresh outcomes to obtain squared bias, prediction variance and conditional target variance. Why do the cross terms vanish? Fresh outcome noise is independent of the fit conditional on the input; the fitted prediction minus its training-sample average has mean zero. These are different expectations, and both cancellations need justification.

□ Proof ▪ *Essential*

$$m = m(x) = \mathbb{E}[\,Y \mid X = x] \tag{1.T15}$$

$$a = \mathbb{E}_D[\hat{f}_D(x)] \tag{1.T16}$$

First separate outcome noise from the fitted model's discrepancy from the target mean. Conditional independence lets the mixed expectation factor; the fresh residual has conditional mean zero, so this mixed term disappears.

$$Y - \hat{f}_D(x) = (Y - m) + (m - \hat{f}_D(x)) \tag{1.T17}$$

$$\mathbb{E}_{D,Y|x}[(Y - m)(m - \hat{f}_D(x))] = 0 \tag{1.T18}$$

$$\mathbb{E}_{D,Y|x}[(Y - \hat{f}_D(x))^2] = v(x) + \mathbb{E}_D[(m - \hat{f}_D(x))^2] \tag{1.T19}$$

Now add and subtract the mean fitted prediction. This time no target noise is involved. The centered fitted prediction has expectation zero by definition, which eliminates the second mixed term.

$$m - \hat{f}_D(x) = (m - a) + (a - \hat{f}_D(x)) \tag{1.T20}$$

$$\mathbb{E}_D[(m - \hat{f}_D(x))^2] = (m - a)^2 + \mathbb{E}_D[(\hat{f}_D(x) - a)^2] \tag{1.T21}$$

$$\mathrm{Bias}(x) = a - m \tag{1.T22}$$

$$V(x) = \mathbb{E}_D[(\hat{f}_D(x) - a)^2] \tag{1.T23}$$

$$\mathrm{Err}(x) = \mathrm{Bias}(x)^2 + V(x) + v(x) \tag{1.T24}$$

To pass from the fixed-input identity to population error, average over independent evaluation inputs. Reusing training targets breaks the fresh-noise independence that made the first cross term vanish; the decomposition therefore cannot certify a training score. Even when valid, it is an identity, not a monotonicity theorem: increasing a complexity setting need not always lower bias or raise variance.

□

Repeated-training experiments estimate variability of the learning procedure, not merely variation across test rows for one fixed model. Squared bias and prediction variance depend on the estimator, sample size, and representation. Neither exact interpolation nor a large model proves failure without a valid evaluation experiment.

◇ Proposition 1.4: What a no-information argument actually establishes

Notation for this result

F, x, D	Random binary labeling function, an unobserved domain input, and observed training information including independent algorithm randomness.
$\hat{f}_D(x)$	Learner's zero-or-one prediction at the unobserved input, determined by its available training information.
$\mathbf{1}\{\cdot\}$	Indicator of disagreement between the prediction and hidden binary label.

Assumptions. Consider a finite input domain. Before data are revealed, every input receives an independent fair binary label; equivalently, all binary labeling functions are equally likely. Query locations, stopping and the choice of an unobserved evaluation input are fixed independently of the labeling function, or chosen nonanticipatingly using only previously revealed labels and independent randomness. A learner observes labels at a strict subset of inputs. Its prediction at an unobserved input may depend on this transcript and independent algorithm randomness, but not on a hidden label or a label-dependent sampling rule.

A learner has expected classification error one half at any unobserved input under this deliberately unstructured prior. This does not say that practical algorithms are equivalent on structured tasks or that success in one application forces failure in every other application.

□ Proof *Advanced*

Condition on the observed training information and the learner's random seed. Its prediction at a hidden input is now fixed. Under the assumed unstructured prior, that input's unseen label is still an independent fair coin: predicting either zero or one disagrees with half the remaining labelings. Conditioning cannot reveal a relationship that this prior explicitly excludes.

$$\Pr(F(x) = 1 \mid D) = \Pr(F(x) = 0 \mid D) = \tfrac{1}{2} \tag{1.T25}$$

$$\mathbb{E}[\mathbf{1}\{\hat{f}_D(x) \neq F(x)\} \mid D] = \tfrac{1}{2} \tag{1.T26}$$

Averaging this conditional error over the training information leaves one half unchanged. Averaging over any distribution supported on unobserved inputs and independent of their labels also leaves one half. The argument works because observed and hidden labels have no relationship in the assumed problem distribution. Smoothness, shared features, or other regularities deliberately introduce such relationships and invalidate this particular fair-coin premise.

□

Useful generalization needs a match between a learning procedure's preferences and the structure of the task. Those preferences can help across many related domains. The precise limitation here is inability to infer independent hidden labels, not a universal ban on broadly useful models.

▶ Example: Bias and variance with three possible fits

At one fixed input, let three equally likely training outcomes give predictions 2, 3 and 4. Let an independent fresh target be 1.5 or 2.5 with equal probability. Its mean is 2 and noise variance is 0.25. The mean prediction is 3, so squared bias is 1; prediction variance is $(1 + 0 + 1)/3 = 2/3$. Expected squared error is therefore $1 + 2/3 + 0.25 = 23/12$, about 1.917. Averaging the six possible target-prediction squared errors gives the same value. These stipulated distributions isolate the identity before the notebook estimates it by repeated fitting.

▶ Example: Repeat the whole learning experiment

Generate forty independent samples from the declared noisy sine relationship. Fit a line and a fifth-degree polynomial to each; predict on one fixed grid. At each grid point, average the forty predictions, square the difference from the known noiseless value, and average squared deviations from the prediction mean using divisor forty. These last two terms exactly decompose the average squared prediction error relative to the noiseless value. Add the declared fresh-noise variance to estimate expected test error. Forty fits approximate the training-sample expectation; no model ordering is presumed. Real data do not directly reveal the true function or exact bias.

Interpretation and limits

Variance across cross-validation scores is not the same quantity as prediction variance in the bias-variance identity. A single fitted model does not reveal its population bias.

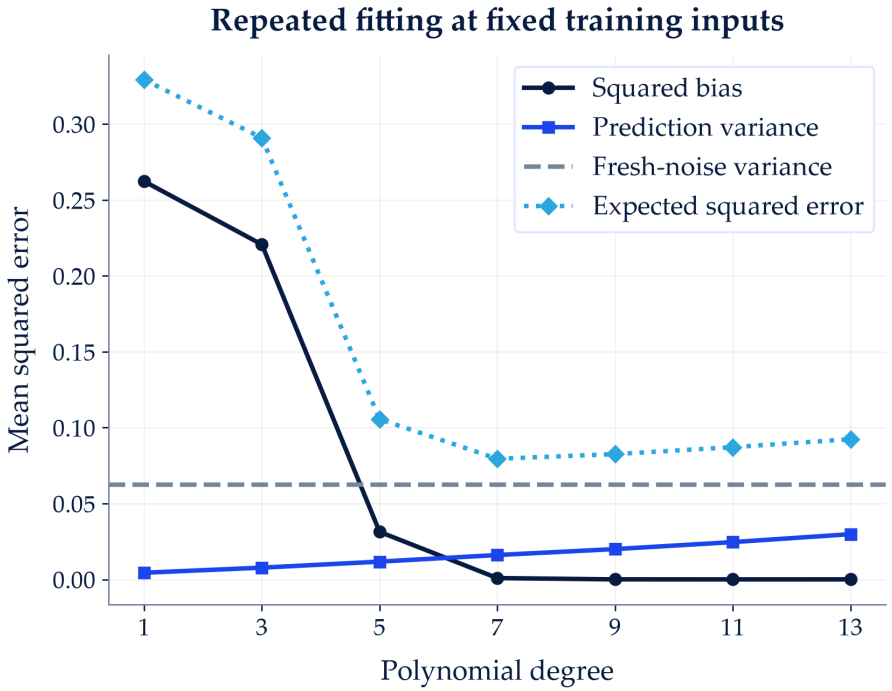


Figure 1.3. Read squared error against polynomial degree: diamonds sum squared bias, prediction variance and fresh-noise variance 0.25 squared. The Monte Carlo experiment refits Legendre least squares on 600 independent noisy target samples at the same 25 inputs, evaluating on 201 inputs. Estimates are conditional on this fixed design, not a universal complexity curve.

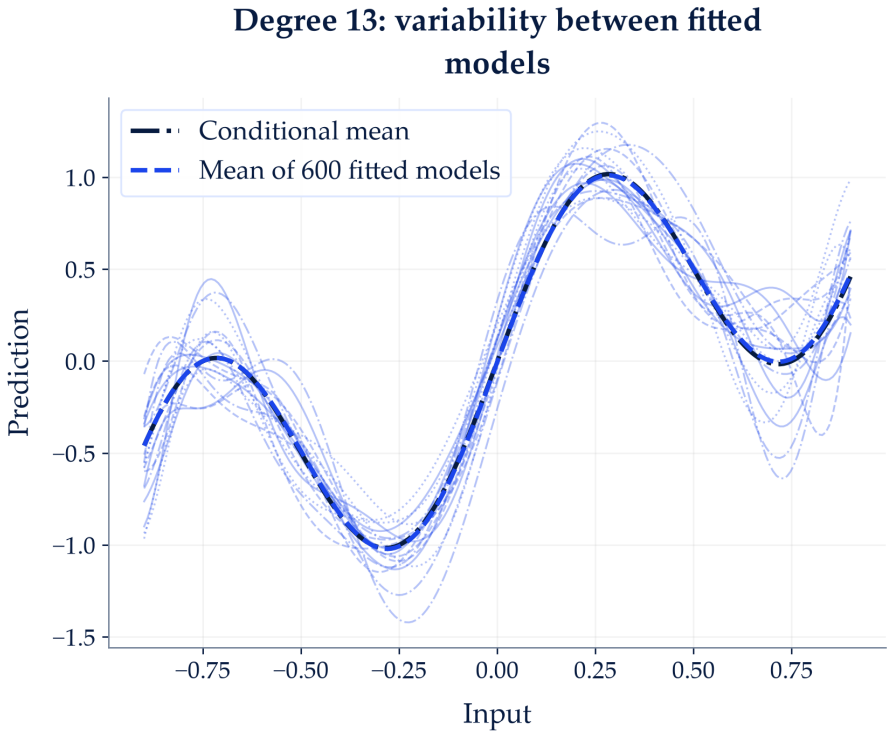


Figure 1.4. Pale curves show 18 of 600 degree-13 least-squares fits. The dashed curve is the mean of all 600 fitted models; the dark curve is the known conditional mean x + 0.75 sin(2 pi x). Variation between fitted predictions is distinct from the bias of their mean.

From A Finite List To A Continuous Linear Model

A linear learner chooses real coefficients, not a member of a short list prepared in advance. This chapter's finite-class theorem can nevertheless control this infinite family if nearby coefficients necessarily produce nearby losses. The following argument makes that connection explicit. It is a conservative finite-dimensional bound, not the sharpest available learning guarantee.

Fix a representation with positive integer dimension d before observing the sample. Let (X, Y) denote a population input-target pair, and let $D = ((X_i, Y_i))_{i=1}^{n}$ contain n independent draws from that same population. Inputs are real column vectors satisfying $\|X\|_2 \leq B$ almost surely, for a fixed

positive bound B. Restrict the coefficient column w to the closed Euclidean ball $\mathcal{W}$ of a fixed positive radius A. The predictor f_w is defined below. If an intercept is needed, append a constant input coordinate and include its coefficient in the norm constraint; the input bound must include that extra coordinate.

$$\mathcal{W} = \{w \in \mathbb{R}^d : \|w\|_2 \leq A\} \tag{1.B1}$$

x One represented input column with d entries; the superscript T denotes transpose.

$$f_w(x) = w^T x \tag{1.B2}$$

Let $\ell(y, s)$ be a measurable loss for target y and real prediction s. Assume it lies in $[0, 1]$ and is K-Lipschitz in its prediction, for a fixed positive constant K: changing the prediction from s to t changes the loss by at most $K|s - t|$. Define population risk R and empirical risk $\widehat{R}_D$ for the same loss. Expectations below are over fresh population pairs, not over a re-used validation score.

$\mathbb{E}$ Expectation under the declared population distribution.

$$R(w) = \mathbb{E}[\ell(Y, w^T X)] \tag{1.B3}$$

i Observation index from one through the positive sample count n.

$$\widehat{R}_D(w) = \frac{1}{n} \sum_{i=1}^{n} \ell(Y_i, w^T X_i) \tag{1.B4}$$

Theorem. Choose a positive coefficient-covering radius $\rho \leq A$ and a failure probability $0 < \delta < 1$ before seeing D. With probability at least $1 - \delta$, the following bound holds simultaneously for every coefficient vector in $\mathcal{W}$, including a vector subsequently fitted using D.

ε_n Uniform risk-deviation allowance for this fixed covering radius; log is the natural logarithm.

$$\varepsilon_n = \sqrt{\frac{d \log(1 + 2A/\rho) + \log(2/\delta)}{2n}} + 2KB\rho \tag{1.B5}$$

sup Least upper bound over every coefficient vector in the stated ball.

$$\sup_{w \in \mathcal{W}} |R(w) - \widehat{R}_D(w)| \leq \varepsilon_n \tag{1.B6}$$

Proof. Construct a maximal collection $\mathcal{N}$ of coefficient vectors in the ball whose pairwise distances exceed ρ. Maximality means every vector in the ball lies within distance ρ of a member of $\mathcal{N}$. The balls of radius $\rho/2$ around the collection are disjoint and lie inside a ball of radius $A + \rho/2$. Comparing their d-dimensional volumes bounds the collection size M. The factor common to every ball volume cancels, so no formula for that factor is needed.

$|\mathcal{N}|$ Number of centers in the fixed finite covering collection, constructed without the data.

$$M = |\mathcal{N}| \leq (1 + 2A/\rho)^d \tag{1.B7}$$

For any two coefficient vectors w and v, the Lipschitz assumption and Cauchy-Schwarz control their loss difference. In particular, replacing w by a covering center within distance ρ changes both its population risk and its sample mean loss by at most $KB\rho$.

v A second coefficient vector in the same ball; y is a possible target and x obeys the input bound.

$$|\ell(y, w^T x) - \ell(y, v^T x)| \leq KB\|w - v\|_2 \tag{1.B8}$$

Apply the bounded-loss finite-class theorem from S2 to all M centers at once. Its deviation allowance is the square-root term in (1.B5), after inserting (1.B7). For any other coefficient vector, move from its population risk to its center's population risk, cross the center's empirical-versus-population gap, then move back to its empirical risk. The two approximation steps contribute $2KB\rho$. This proves (1.B6). Continuity in the coefficients allows the supremum to be taken over a countable dense subset, so this event is measurable.

To translate this into learning, let $\widehat{w}$ minimize empirical risk over $\mathcal{W}$ to within a nonnegative optimization tolerance τ. Let w_* minimize population risk over that ball. Both risks are continuous on a compact set, so their exact minima exist. On the same uniform event, the usual two-deviation argument gives a bound on excess population risk.

$$R(\widehat{w}) - R(w_*) \leq 2\varepsilon_n + \tau \tag{1.B9}$$

Indeed, move from the fitted population risk to fitted empirical risk, use approximate empirical minimization, and move from the comparator's empirical risk back to its population risk. Each statistical move costs at most ε_n. This is why counting only the fitted outputs is unnecessary: the event already covers every eligible coefficient vector, including data-dependent choices.

Example. Use clipped absolute error, $\ell(y, s) = \min\{1, |y - s|\}$, in declared dimensionless target units. This loss has $K = 1$. For $d = 2$, $A = B = 1$, $n = 10000$, $\rho = 0.01$ and $\delta = 0.05$, the covering-size bound is 40401. Substitution gives a uniform deviation below 0.047 and an exact empirical-minimization excess-risk allowance below 0.094. These numbers are mathematical allowances, not measured model errors. Clipping intentionally caps the loss; it must not silently replace a deployment objective whose large errors matter.

Read the bound's scope before applying it. Raw squared error and unbounded-logit log loss violate boundedness; zero-one loss violates the stated continuity control. Counting final coefficients also ignores data-dependent feature construction: fix the representation independently or control the joint family. Coarser covers need fewer sampling comparisons but approximate predictors less accurately, producing the trade-off in (1.B5). A norm restriction may reduce estimation error while increasing approximation error. Finally, penalized minimization is not the empirical-risk minimization assumed in (1.B9); connecting the two requires a separate comparison argument.

Shalev-Shwartz and Ben-David (2014), Understanding Machine Learning, Chapter 4: the finite-class uniform-convergence argument extended here by a Euclidean covering construction. Source

Table 1.1. Match the learning problem to the evidence

Problem	Observed evidence	Output and main limitation
Regression	Inputs with numeric targets	A numeric prediction; error depends on units and the loss.
Classification	Inputs with class labels	A score or probability, then a decision; these require different evaluations.
Clustering	Inputs without task labels	A partition under a geometry or density assumption; not automatically real categories.
Representation learning	Inputs, sometimes with target supervision	A transformed coordinate system; useful variance need not be predictive.

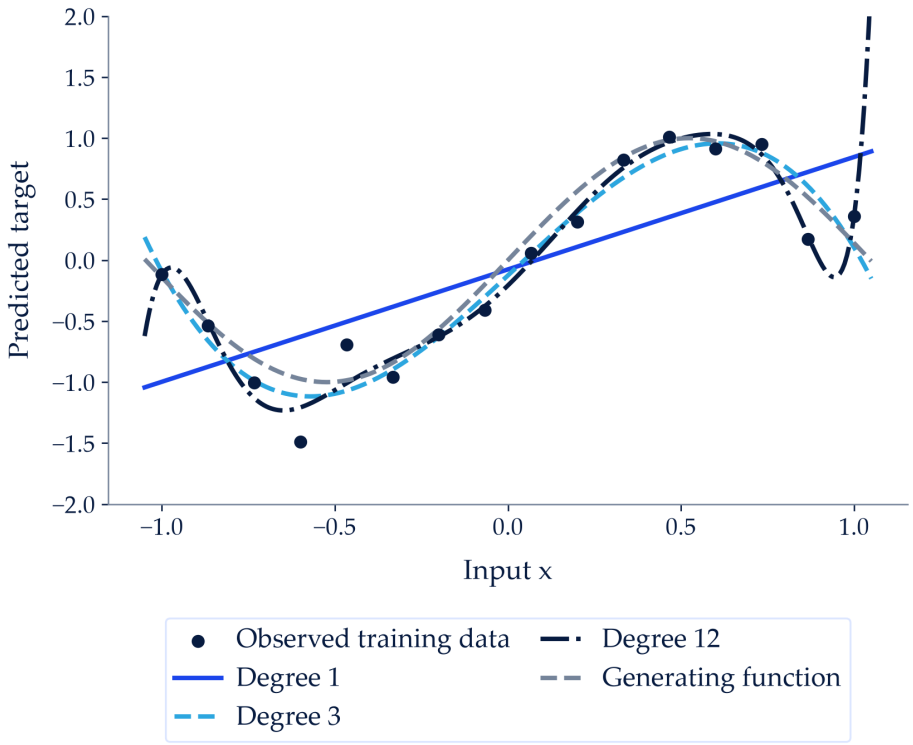


Figure 1.5. A reproducible synthetic sample with 16 observations. All fitted curves use the same training data; the dashed curve is known only because this is a simulation. Training fit alone cannot select the model.

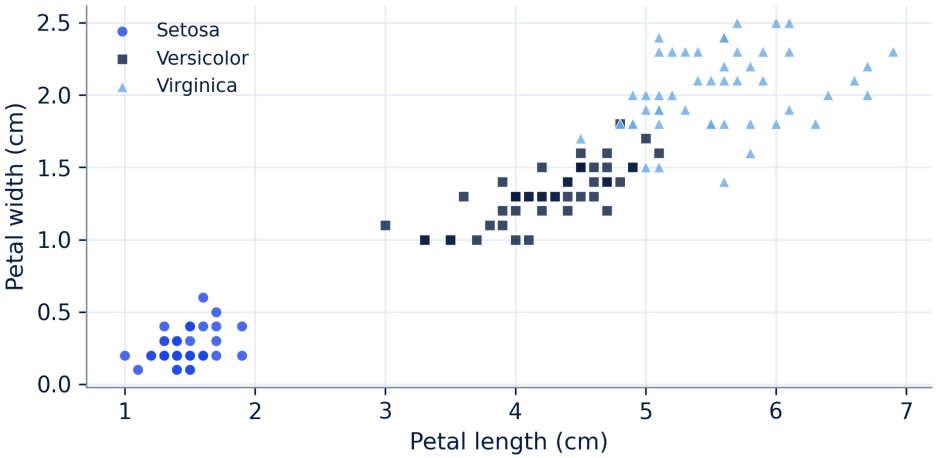
Table 1.2. Chapter synthesis

Concept	Main conclusion	Engineering check
Prediction contract	Specify the target, prediction time and available inputs before modeling.	Can every feature actually be observed when a prediction is needed?
Risk	An empirical average estimates a population objective under sampling assumptions.	Do the sampled cases resemble the intended population?

Concept	Main conclusion	Engineering check
Generalization	Performance on fitting data is not independent evidence.	Keep the final test outside development decisions.
Inductive bias	Restrictions trade flexibility against estimation uncertainty.	Compare an explicit simple baseline before adding capacity.

Iris

150 specimens | **4** measurements | **3** species



All 150 flowers in two of their four measured coordinates. Species overlap is part of the data, not a plotting error.

Iris records four flower measurements in centimetres for 150 specimens: 50 each of setosa, versicolor and virginica. It makes class overlap and feature geometry visible, but this small historical collection does not establish performance on flowers collected at new sites. [1.10]

Here: the scikit-learn corrected copy. The case uses all four measurements; later laboratories sometimes select fewer species or predict a held-out measurement. A two-axis view is not a generalization test.

Fisher / UCI, CC BY 4.0. Chart from the scikit-learn distribution. Source / terms.

Case Study: Iris: write the prediction contract before chasing accuracy

Problem and data

The Iris collection supplies four centimetre measurements for each of 150 specimens, with three recorded species. Formulate a prediction contract for identifying a specimen after all four measurements are available. The target population is comparable specimens from this historical collection, not arbitrary plants or new field conditions. Use zero-one loss and reserve final evaluation observations before inspecting model performance. The question is whether added fitting capacity improves independent predictions, not whether training labels can be reproduced.

Experimental design

Split row identifiers into stratified training, validation and final-test roles using a fixed seed. Fit a majority-class baseline and trees with depths one, three and unrestricted on training observations only. Treat these trees as a preview of capacity control, not a replacement for their later derivation. Record each procedure's empirical training and validation risk, choose by validation error with a fixed simplicity tie rule, and freeze that fitted procedure before comparing it with the baseline on the final test.

Measured Reference Run

The saved run used 90 fitting, 30 validation and 30 test specimens, with stratification and seed 2027. A majority rule and trees of depth one, depth three and unrestricted depth formed the entire search. Validation error selected the already fitted unrestricted tree; no test-driven refitting followed. The table reports error counts, not population risks.

Table 1.3. Measured case-study evidence

Procedure	Fit / 90	Validation / 30	Test / 30
Majority	60	20	20
Depth one	30	10	Not evaluated
Depth three	3	1	Not evaluated
Unrestricted	0	0	1

The unrestricted tree beat both shallower trees on validation. Its frozen-test error count was 1/30, versus 20/30 for the constant reference: empirical risks 0.033 and 0.667. This supports the selected rule on this split, not unrestricted depth generally.

Failure Analysis And Limits

Zero fitting and validation errors did not imply a flawless frozen rule: one final specimen was misclassified. Each test error changes the estimate by 1/30. This is a finite-sample discrepancy, not proof that a particular depth is generally best. Missing collection-site identifiers prevent a new-site generalization claim.

Historical saved-run outputs, not population guarantees. Retain the recorded software versions, seed and source-row split order; the evidence record identifies each output cell.

▶ Example: inspecting the real data

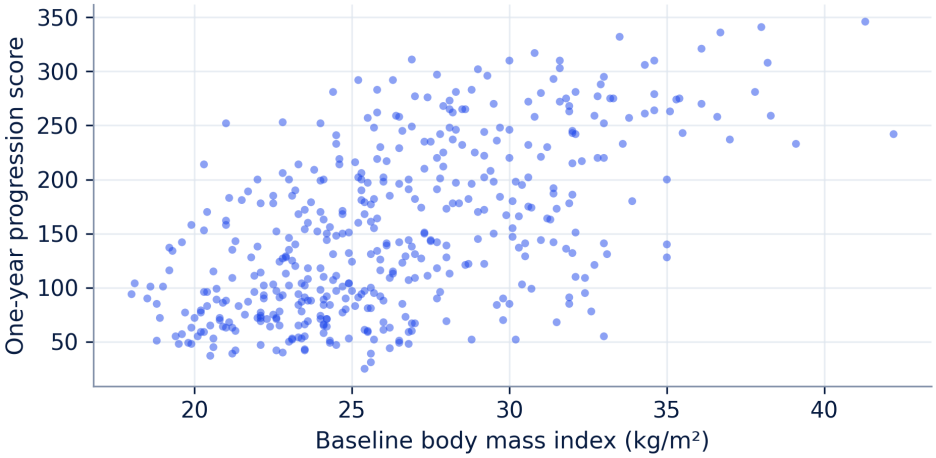
The complete collection contains 50 observations per species. A constant species prediction therefore makes 100 errors among 150 records, an empirical zero-one risk of 100/150 = 2/3. This full-collection arithmetic illustrates a rule; it is not the notebook's untouched test result.

Dataset source and documentation

Companion: Week 01, Real-Data Case Study notebook. Reproduce the declared run, then investigate a changed condition. No fixed benchmark score is promised.

Diabetes progression

442 patients	**10** baseline inputs	**1 year** follow-up



The 442 patients, using unscaled body mass index. This association is descriptive; it is not a causal or clinical decision rule.

This regression benchmark follows 442 patients with diabetes. Ten baseline variables accompany a quantitative disease-progression response one year later. The opening simulator uses only BMI, while later cases use more predictors. The response is a benchmark score, not a diagnosis, probability or treatment effect. [1.11]

Here: the opening laboratory uses raw BMI from this source; later cases use ten baseline inputs. The displayed progression score is not a probability. Fitting-set size depends on the experiment, not the full 442-row source.

Efron, Hastie, Johnstone and Tibshirani; raw scikit-learn copy. Descriptive chart, not clinical validation. Source / terms.

A Controlled Learning Cycle

The Iris case connects the prediction contract to fitted and held-out risk. Week 1 notebooks develop the risk calculations and information-boundary checks; the Diabetes laboratory makes coverage, sample size and polynomial flexibility visible. Compare a baseline with one declared change and record the metric, data roles and explanation. The transfer boundary below limits the comparison to mechanisms, not matching scores or known population truth.

Matched practice and simulator protocol · Packaged edition; launch instructions in the guide.

Laboratory 01 | ML Problems, Data, And Generalization

Coverage, sample size and polynomial flexibility on measured rows.

Measured Diabetes observations reveal how training coverage affects polynomial predictions beyond the observed range.

Open Laboratory 01 | Use the live view for readable controls.

Transfer: Iris species classification → Diabetes BMI-to-progression regression. No known population truth or independent proof of generalization.

Chapter Capability

Write a defensible prediction contract and separate fitting evidence from evidence about new observations.

Week 01: separate practice, solutions and assessment

References

[1.1] Vapnik, V. N. (1998). Statistical Learning Theory. Wiley. Source / DOI

[1.2] Hastie, T.; Tibshirani, R.; Friedman, J. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction. 2nd ed., Springer. Source / DOI

[1.3] Shalev-Shwartz, S.; Ben-David, S. (2014). Understanding Machine Learning: From Theory to Algorithms. Cambridge University Press. Source / DOI

[1.4] Geman, S.; Bienenstock, E.; Doursat, R. (1992). Neural Networks and the Bias/Variance Dilemma. Neural Computation 4(1), 1-58. Source / DOI

[1.5] Bellman, R. (1961). Adaptive Control Processes: A Guided Tour. Princeton University Press. Source / DOI

[1.6] Wolpert, D. H. (1996). The Lack of A Priori Distinctions Between Learning Algorithms. Neural Computation 8(7), 1341-1390. Source / DOI

[1.7] Hoeffding, W. (1963). Probability Inequalities for Sums of Bounded Random Variables. Journal of the American Statistical Association 58(301), 13-30. Source / DOI

[1.8] Vapnik, V. N.; Chervonenkis, A. Ya. (1971). On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities. Theory of Probability & Its Applications 16(2), 264-280. Source / DOI

[1.9] Bousquet, O.; Elisseeff, A. (2002). Stability and Generalization. Journal of Machine Learning Research 2, 499-526. Source / DOI

[1.10] Fisher, R. A. (1936). Iris. UCI Machine Learning Repository. Source / DOI
[1.11] Efron, B.; Hastie, T.; Johnstone, I.; Tibshirani, R. (2004). Least angle regression. The Annals of Statistics, 32(2), 407-451. Source / DOI

Recommended

Mathematical Refresher: The Tools Used In The Proofs

Review probability inequalities before the finite-class proof and vector derivatives before the smooth-descent proof in Chapter 2.

Vectors, Shapes And A Useful Inequality

Vectors are columns. Let p be a positive integer and let u and v be real vectors with p entries. A superscript T means transpose; the subscript j selects coordinate j. The Euclidean inner product measures alignment, and the Euclidean norm measures length. A norm can combine coordinates numerically even when their original physical units differ, but that choice then needs a deliberate scaling convention.

p, j Positive vector dimension and coordinate index from one through p.

u, v Real p-entry column vectors; T denotes transpose.

$$u^T v = \sum_{j=1}^{p} u_j v_j \tag{S1.1}$$

$\|\cdot\|_2$ Euclidean vector norm; the square root is the nonnegative root.

$$\|u\|_2 = \sqrt{u^T u} \tag{S1.2}$$

$$|u^T v| \le \|u\|_2 \|v\|_2 \tag{S1.3}$$

This is the Cauchy-Schwarz inequality. If v is zero it is immediate. Otherwise expand the nonnegative squared norm of u minus a times v, where a is any real scalar, and choose a equal to the inner product divided by the squared norm of v. Rearrangement gives the displayed inequality. S3 uses it to control a gradient's change without inspecting each coordinate separately.

Differentiation Along A Line

Let F be a continuously differentiable real-valued function on all real p-entry vectors. Its gradient is the column of its p partial derivatives. For fixed vectors u and v, define the displacement Delta as v minus u and the scalar function phi as F along the straight segment from u to v. The real scalar s runs from zero to one. The ordinary chain rule becomes an inner product.

$F, \nabla F$ Continuously differentiable scalar objective on real p-entry vectors and its p-entry gradient.

Δ Displacement v minus u; a real p-entry vector.

s, ϕ Scalar segment coordinate in [0,1] and the objective restricted to that segment.

$$\phi(s) = F(u + s\Delta) \tag{S1.4}$$

ϕ' Ordinary derivative with respect to the scalar s.

$$\phi'(s) = \nabla F(u + s\Delta)^T \Delta \tag{S1.5}$$

For example, a design matrix A with n rows and p columns maps a p-entry parameter theta to n predictions. Let y contain the n observed targets and let n be positive. The half-mean-squared residual objective has the following gradient. Its shape is p by one, not n by one. The factor one over n belongs to the mean-loss convention; replacing the mean by a sum changes both this gradient and a safe learning rate.

n, A, y Positive observation count, real n-by-p design matrix, and n-entry target column.

θ Real p-entry parameter column; A may include an intercept column.

$$F(\theta) = \frac{\|A\theta - y\|_2^2}{2n} \tag{S1.6}$$

$$\nabla F(\theta) = \frac{A^T(A\theta - y)}{n} \tag{S1.7}$$

Averages, Probability And Independence

A random variable is a numerical outcome before its realization is known. Expectation is its probability-weighted average, not necessarily an attainable outcome. For a finite-valued variable U, let u_j be its possible values, p_j their probabilities, and m their positive count. The probabilities are nonnegative and sum to one. The notation E means expectation and Pr means probability under the declared random experiment.

U, u_j Finite-valued real random variable and its jth possible value.

m, j, p_j Positive number of possible values, value index, and probability of that value.

$\mathbb{E}, \Pr$ Expectation and probability under the stated distribution.

$$\mathbb{E}[U] = \sum_{j=1}^{m} p_j u_j \tag{S1.8}$$

Linearity of expectation does not require independent variables. Independence is stronger: information about one variable does not change the joint probability law of the others in the relevant factorization. In particular, it permits factorization of expectations of products of integrable functions of the separate variables. Independent and identically distributed observations, abbreviated i.i.d., share one distribution and are jointly independent. Disjoint spreadsheet rows alone do not establish this property.

One geometric fact also used in S2 is convexity: a scalar function on an interval is convex when its value anywhere between two arguments is no greater than the height of the straight chord joining their graph points. The exponential function is convex because its second derivative is positive. Two elementary probability bounds then drive the argument. Markov's inequality uses a nonnegative random variable V with finite expectation and a positive threshold

a. On the event that V reaches a, its contribution to the expectation is at least a times that event's probability. The union bound controls the chance that any one of finitely many bad events occurs, without requiring independence between those events.

V, a Nonnegative random variable with finite expectation, and strictly positive scalar threshold.

$$\Pr(V \geq a) \leq \frac{\mathbb{E}[V]}{a} \tag{S1.9}$$

M, j, E_j Positive finite event count, event index, and jth event; the union means at least one event occurs.

$$\Pr\left(\bigcup_{j=1}^{M} E_j\right) \leq \sum_{j=1}^{M} \Pr(E_j) \tag{S1.10}$$

For the union bound, the indicator of at least one event is no greater than the sum of their individual indicators; taking expectations proves the claim. This simple observation is why several candidate models can be compared even though they are scored on the same sample. What requires independence in the later argument is the sample across observations, not the candidate-model errors across models.

References

Stephen Boyd and Lieven Vandenberghe, Convex Optimization (2004), mathematical background and differentiable convex functions Source

Shai Shalev-Shwartz and Shai Ben-David, Understanding Machine Learning: From Theory to Algorithms (2014), Chapters 3-4 Source

Advanced

Finite-Class Generalization With Bounded Loss

A finite collection and a bounded loss make the cost of selecting a model explicit. The result is a sufficient worst-case guarantee under independent sampling, not a recommended universal dataset size.

Set The Experiment Before Stating The Guarantee

Let Z be one input-target observation drawn from a fixed population distribution P. Let D be an ordered sample of n i.i.d. observations Z_i, with n positive. Let H be a nonempty finite collection of M measurable prediction functions h, fixed independently of D. For an observation z, let $\ell(h, z)$ be the declared loss, taking values between zero and one. Zero-one classification error is one example. The natural logarithm is written log and its inverse is exp. No candidate is assumed to predict perfectly.

P, Z, z	Fixed population distribution, random observation, and a possible observation.
$\mathcal{H}, M, h$	Nonempty fixed finite predictor collection, its positive size, and a member of the collection.
$\ell(h, z), R(h)$	Measurable loss in [0,1] and its population mean for predictor h.

$$R(h) = \mathbb{E}_{Z \sim P}[\ell(h, Z)] \tag{S2.1}$$

n, i, Z_i, D	Positive sample count, index from one through n, indexed i.i.d. observation from P, and their ordered sample.
$\widehat{R}_D(h)$	Empirical mean loss of h on D.

$$\widehat{R}_D(h) = \frac{1}{n}\sum_{i=1}^{n} \ell(h, Z_i) \tag{S2.2}$$

The same theorem also applies conditionally to M fitted predictors produced using a separate training dataset, provided the evaluation observations are independent of all information used to construct that collection. In that application n counts evaluation observations. It does not permit training M models on D and then counting only those M fitted outputs as a fixed collection. A finite hyperparameter list is not, by itself, a finite list of predictors independent of the sample being scored.

Lemma: An Exponential Bound For One Bounded Variable

Let U be any real random variable between zero and one, let mu be its expectation, and let t be any real scalar. The endpoints mu equal to zero or one force U to be constant with probability one, so consider mu strictly between them. Convexity of the exponential puts its value at U below the chord joining its values at zero and one. After taking expectations and centering, define the scalar function psi as follows.

U, μ Real random variable in [0,1] and its expectation.

t, e^t Arbitrary real scalar and its natural exponential.

$$\mathbb{E}[e^{t(U-\mu)}] \le e^{-t\mu}(1 - \mu + \mu e^t) \tag{S2.3}$$

$\psi, \log$ Scalar log bound as a function of t, and natural logarithm.

$$\psi(t) = \log(1 - \mu + \mu e^t) - t\mu \tag{S2.4}$$

Define q(t) as the exponential-weighted fraction below. Direct differentiation gives psi at zero and its first derivative at zero both equal to zero. Its second derivative is at most one quarter, since a number between zero and one multiplied by its complement cannot exceed one quarter.

$q(t)$ Auxiliary scalar in (0,1); not a class probability predicted by a model.

$$q(t) = \frac{\mu e^t}{1 - \mu + \mu e^t} \tag{S2.5}$$

ψ'' Second ordinary derivative of psi with respect to t.

$$\psi''(t) = q(t)(1 - q(t)) \le \frac{1}{4} \tag{S2.6}$$

Taylor's formula with integral remainder, after changing to an integration variable s between zero and one, works for positive and negative t. The vanished constant and linear terms leave the following bound. It also holds at the previously separated deterministic endpoints.

s Scalar integration variable from zero to one.

$$\psi(t) = t^2 \int_0^1 (1-s)\psi''(st)\, ds \le \frac{t^2}{8} \tag{S2.7}$$

$$\mathbb{E}[e^{t(U-\mu)}] \le e^{t^2/8} \tag{S2.8}$$

From One Predictor To Every Predictor

Fix h before drawing D. Set U_i to its loss on observation i and S_n to the sum of the centered losses. Independence across observations lets their exponential expectations multiply. Let ϵ be a strictly positive deviation size; use a positive t for Markov's inequality.

U_i, S_n Loss $\ell(h, Z_i)$ for the fixed predictor and the sum of its centered losses.

ϵ Strictly positive desired deviation in normalized loss units.

$$S_n = \sum_{i=1}^{n} (U_i - \mathbb{E}[U_i]) \tag{S2.9}$$

$$\mathbb{E}[e^{tS_n}] = \prod_{i=1}^{n} \mathbb{E}[e^{t(U_i - \mathbb{E}[U_i])}] \le e^{nt^2/8} \tag{S2.10}$$

$$\Pr(S_n \ge n\epsilon) \le \exp(-tn\epsilon + nt^2/8) \tag{S2.11}$$

The exponent is minimized at t equal to four times epsilon, giving a one-sided bound exp of minus twice n times epsilon squared. Apply the same argument to one minus each loss for the lower tail and add the two probabilities. Finally, take the union over the M fixed predictors. Their errors need not be independent of each other.

$$\Pr(|\widehat{R}_D(h) - R(h)| \ge \epsilon) \le 2e^{-2n\epsilon^2} \tag{S2.12}$$

$$\Pr\left(\max_{h \in \mathcal{H}} |\widehat{R}_D(h) - R(h)| \ge \epsilon\right) \le 2Me^{-2n\epsilon^2} \tag{S2.13}$$

Theorem: Uniform Accuracy And Selection Error

Let delta be a chosen failure probability strictly between zero and one. Define the deviation allowance ϵ_n below. With probability at least one minus delta over the sample D, every predictor in H has empirical and population risks differing by at most this allowance. The probability statement concerns repeated samples, not the probability that one already fixed numerical risk is random.

δ, ϵ_n Chosen failure probability in (0,1) and simultaneous deviation allowance for the stated sample size.

$$\epsilon_n = \sqrt{\frac{\log(2M/\delta)}{2n}} \tag{S2.14}$$

The uniform event follows by substituting ϵ_n into the preceding tail bound. Because it holds for every member at once, it also holds for a member selected using D. Let $\widehat{h}$ be an approximate empirical risk minimizer: its empirical risk exceeds the smallest empirical risk in H by at most a declared nonnegative tolerance tau. Let h_* minimize population risk within H; such a member exists because H is nonempty and finite. Ties may be broken by a fixed ordering.

$\widehat{h}, \tau$ Selected member of H and nonnegative empirical optimization tolerance.

h_* A member of H attaining its minimum population risk; not necessarily a globally best measurable predictor.

$$\widehat{R}_D(\widehat{h}) \le \min_{h \in \mathcal{H}} \widehat{R}_D(h) + \tau \tag{S2.15}$$

On the uniform event, move from the selected model's population risk to its empirical risk, use approximate minimization, then move from the best-in-class model's empirical risk back to population risk. There are two estimation allowances, not one. This proves the excess-risk statement, including the cost of incomplete empirical optimization.

$$R(\widehat{h}) \le \widehat{R}_D(\widehat{h}) + \epsilon_n \tag{S2.16}$$

$$\widehat{R}_D(\widehat{h}) \le \widehat{R}_D(h_*) + \tau \tag{S2.17}$$

$$\widehat{R}_D(h_*) \le R(h_*) + \epsilon_n \tag{S2.18}$$

$$R(\widehat{h}) - R(h_*) \leq 2\epsilon_n + \tau \tag{S2.19}$$

This is an agnostic guarantee: the best available predictor may have nonzero risk. It controls selection relative to H, not the approximation error caused by choosing an unsuitable H. For exact empirical minimization, tau is zero. If gamma is a positive desired upper bound on excess risk, a sufficient integer sample count is the following. The ceiling is the smallest integer not below its argument.

$\gamma, \lceil \cdot \rceil$ Positive target excess-risk allowance for exact empirical minimization, and ceiling operation.

$$n \geq \left\lceil \frac{2\log(2M/\delta)}{\gamma^2} \right\rceil \tag{S2.20}$$

Example: Distinguish A Deviation Budget From A Selection Budget

Suppose one hundred classification predictors have been fixed before an independent evaluation sample is drawn. Use zero-one loss, a failure probability of 0.05, and exact selection of the smallest empirical loss. To guarantee excess population error at most 0.10 above the best of these one hundred predictors, the sufficient count is 1,659. This uses a simultaneous deviation allowance of 0.05: two such allowances become the 0.10 selection budget.

$$\left\lceil \frac{2\log(4000)}{0.10^2} \right\rceil = 1659 \tag{S2.21}$$

With 2,000 independent evaluation observations instead, the deviation allowance is approximately 0.046 and the excess-risk allowance is approximately 0.091. These are percentage-point differences in error probability, not relative percentage improvements. The calculation is a worst-case theoretical illustration, not a measured dataset result, a claim that 1,658 observations fail, or a guarantee of useful absolute error. If the best candidate is poor, selecting almost as well remains poor.

What The Theorem Does Not Cover

An infinite model family needs additional complexity control; replacing its size by the number of models one happened to try is invalid. Adaptive candidate generation using the same evaluation data also breaks the fixed-family argument unless the entire eligible family was fixed in advance and counted. Independent training followed by one finite comparison is covered conditionally, but that comparison sample is then development evidence, not an untouched final test. Refitting a selected pipeline changes the fitted predictor, so the earlier conditional guarantee does not automatically transfer to the refit.

Ordinary squared error and log loss are not bounded by one in general. A known deterministic loss range can be rescaled, with the bound then translated back into the original loss units. Choosing a clipping level after viewing evaluation outcomes is not justified by this proof; fixing a cap in advance changes the risk being estimated and may hide costly tail errors. Related entities, time dependence and population change also require other assumptions. Increasing n does not repair a mismatch between the evaluation distribution and the intended prediction population.

References

Wassily Hoeffding, Probability Inequalities for Sums of Bounded Random Variables, Journal of the American Statistical Association 58(301), 13-30 (1963), original paper Source

Shai Shalev-Shwartz and Shai Ben-David, Understanding Machine Learning: From Theory to Algorithms (2014), Chapter 4: Learning via Uniform Convergence Source

Advanced

Local Coverage In A Uniform Hypercube

Supporting proof for Section 1.4.

Notation For This Result

d, j	Positive input dimension and coordinate index ranging from one through that dimension.
r, q	Subcube side length and captured probability mass, each strictly between zero and one.
A, N_A	Fixed subcube neighborhood and number of sampled inputs lying in it.
n, i, X_i	Sample count, observation index from one through that count, and the indexed random input.

$\mathbf{1}\{\cdot\}$	Indicator equal to one when the enclosed event occurs and zero otherwise.
$\Pr$	Probability over repeated independent input samples under the declared uniform sampling distribution.
δ	Allowed probability of observing no input inside the fixed neighborhood, strictly between zero and one.
$\lceil\cdot\rceil, \log$	Ceiling: the smallest integer greater than or equal to its argument. The logarithm is natural.

Assumptions. The input has independent uniform coordinates on the unit interval. A fixed axis-aligned subcube lies entirely inside the unit hypercube. Training inputs are independent draws from this distribution. The neighborhood is fixed before observing those draws.

A neighborhood with fixed side length captures an exponentially decreasing population fraction as dimension increases. The exact probability of collecting at least one observation in it determines a corresponding sample requirement; this is a geometric statement under the specified sampling model, not a universal impossibility theorem.

□ Proof *Advanced*

A single coordinate lies in an interval of the specified width with probability equal to that width. Independence multiplies these probabilities across coordinates. Equivalently, the probability is the ratio of the subcube volume to the unit-cube volume.

$$q = \prod_{j=1}^{d} r = r^{d} \tag{1.T9}$$

$$r = q^{1/d} \tag{1.T10}$$

For each sampled input, define an indicator recording whether it falls inside the fixed subcube. Their sum counts available local observations. Independent sampling makes this count binomial, including its zero-count probability.

$$N_A = \sum_{i=1}^{n} \mathbf{1}\{X_i \in A\} \tag{1.T11}$$

$$\mathbb{E}[N_A] = nq \tag{1.T12}$$

$$\Pr(N_A = 0) = (1 - q)^n \tag{1.T13}$$

Demanding a probability of at least one local observation of at least one minus a chosen failure level gives the following exact integer requirement. Dividing the logarithms reverses the inequality because the denominator is negative.

$$n \geq \left\lceil \frac{\log \delta}{\log(1 - r^d)} \right\rceil \tag{1.T14}$$

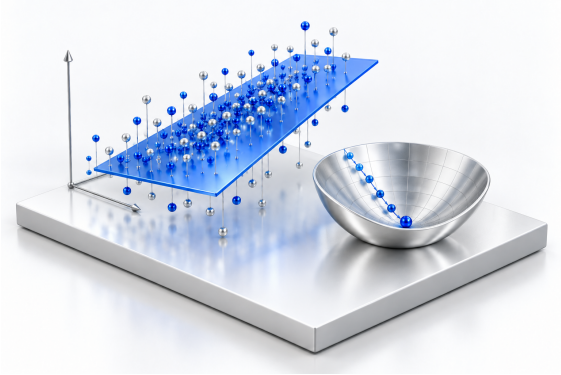
For small captured mass, the negative logarithm of one minus that mass is approximately the mass itself. The required count therefore grows approximately as the reciprocal of the captured mass. Merely obtaining one point is much weaker than having enough points for accurate local regression; this calculation should not be advertised as a sufficient prediction guarantee. □

At fixed neighborhood width the coverage burden grows exponentially with dimension, while at fixed captured mass the neighborhood becomes less local. Structured inputs, sparse relevant variables, or a learned low-dimensional representation can violate the uniform-cube assumptions and change this conclusion. Adding features is not automatically harmful; the question is whether they supply signal commensurate with their effect on geometry.

Return to the chapter and its references

CHAPTER 02

Regression, Optimization, And Regularization



Reading path

Start with Essential. The other levels offer more depth.

Essential
2.1 2.2 2.3 2.4 2.5

Recommended
Proofs 2.1 2.2 2.5

Advanced
Proofs 2.3 2.4

Worked example Lab task

A regression error can come from the data representation, the numerical optimizer, or the model itself. Separate those explanations before adding complexity.

What You Will Be Able To Do

- Build a regression pipeline whose preprocessing is fitted on training observations only.
- Derive the least-squares objective from a stated Gaussian observation model.
- Implement and check a gradient-descent update, including its connection to forward Euler.
- Compare Ridge and Lasso with explicit objective conventions and interpret regression metrics.

Before you begin: Chapter 1: risk and data roles; Matrix multiplication and transpose; Derivatives and elementary logarithms.

Follow fitted preprocessing and predictions first; revisit likelihood and optimization proofs after reproducing a fit.

Algorithm scope: Complete teaching implementations: full-batch gradient descent and cyclic Lasso with KKT checks. Ridge/least squares delegates QR or SVD factorization; the intercept and penalty conventions remain explicit.

Shared chapter notation

d	Positive represented-feature count, excluding the added intercept column.
n, d	Positive training-observation count and represented-feature count, respectively.
$X,\ y,\ n$	Training design (n by d), target column (n entries), and training-observation count.
$y_i,\ \hat{y}_i$	Observed scalar target and fixed-model prediction in the same target units; i indexes evaluation observations.
$\|\cdot\|_2$	Euclidean vector norm: square root of the sum of squared entries. Unsubscripted algorithm norms have the same meaning.
$\bar{x}, \bar{y}$	Training feature-mean column (d entries) and scalar target mean; the pseudocode aliases $\bar{x}$ and $\bar{y}$ render as bars.

Chapter notation

n	Positive observation count: training count in objectives, evaluation count in reported metrics.

Regression makes the learning process visible: represent imperfect observations, specify a predictor, derive and minimize its objective, then evaluate the result. Neural-network training follows the same sequence with a more complicated function and optimization landscape. A transparent model therefore explains what an otherwise opaque fitting call must accomplish.

Linear prediction is a foundation, not an assumption that every useful relationship is linear. Feature transformations can make a model linear in its parameters but nonlinear in the original inputs. Preprocessing changes numerical geometry and penalty interpretation; regularization changes the objective. Keep these choices explicit and let measured comparisons, not a presumed winner, guide selection.

- *Essential*

2.1 Preprocessing is part of the fitted model

First decide what each unusual table entry means. An unknown measurement is not necessarily zero; a rare observation is not necessarily invalid. Missing values, incompatible scales, categories and mistakes need different treatments. Deleting incomplete rows can change the represented population.

Training-median imputation is a useful baseline, not a universal remedy: investigate the missingness mechanism, because missingness itself may carry information.

Fit each column's mean and standard deviation on training data, then reuse them unchanged. Standardization can improve iterative conditioning and make coefficient penalties comparable across units, but correlated columns remain correlated; equal marginal scales do not imply a spherical objective. Define a safe treatment for constant columns. For categories, use a representation such as one-hot encoding and declare how unseen values are handled. Arbitrary integer category codes can introduce an unintended ordering.

All learned preprocessing belongs inside the training procedure. Imputation values, scaling statistics, category vocabularies and clipping thresholds must not be estimated using validation or test observations. A pipeline combines transformations and the estimator so the same fitted operations are reused at prediction time. It also makes cross-validation safer because each training fold can learn its own preprocessing. Clipping unusual observations should be justified by measurement knowledge or a validated robustness strategy, not applied automatically to hide inconvenient errors.

Golub and Van Loan [2.2] explain how scaling and conditioning affect numerical linear algebra. Their analysis clarifies why rescaling inputs can change optimization behavior even when a model class can represent the same predictions.

Notation for equation (2.1)

$i,\ j$	Observation index 1 through n and feature index 1 through d in the current partition.
x_{ij}	Unstandardized feature j of observation i, after fitted missing-value treatment.
z_{ij}	Feature value relative to its training mean and standard deviation.
μ_j^{train}	Feature-j mean from the current training partition only.
s_j^{train}	Strictly positive feature-j standard deviation from that training partition only.

$$z_{ij} = \frac{x_{ij} - \mu_j^{\text{train}}}{s_j^{\text{train}}} \tag{2.1}$$

The location and scale are estimated from the current training partition only. The same fitted values transform later observations. Standardization changes marginal scales but does not generally decorrelate columns.

Declare the variance convention used by the transformer. Constant columns require a separate rule rather than division by zero.

◇ Proposition 2.1: Standardization changes scales, not correlations

Notation for this result

X, n, d	Centered training matrix (n by d), observation count, and feature count.
S	Symmetric empirical feature covariance (d by d), divisor n; also the coefficient Hessian.
D_s, s_j, j	Diagonal scale matrix (d by d), positive column-j standard deviation, and feature index 1 through d.
S_{jj}	Feature-j empirical variance under the stated divisor.
diag	Forms a diagonal matrix from its listed scalar entries.
Z, H_Z	Standardized training design (n by d) and symmetric coefficient-coordinate Hessian (d by d).
k	Second feature index 1 through d; equal indices give diagonal entries.
$S_{jk}, (H_Z)_{jk}$	Row-j, column-k entries of covariance and standardized Hessian.
ρ	Two-feature empirical correlation, strictly between minus one and one.
$\lambda_\pm$	Eigenvalues of the displayed two-feature Hessian.

Assumptions. The training feature matrix is column-centered. Each feature has strictly positive empirical variance under the divisor equal to the sample count. The following Hessians refer to unregularized half-mean-squared error in the feature coefficients; the intercept has been removed by centering.

After column standardization, each coordinate has unit marginal curvature: the Hessian diagonal is one. The off-diagonal correlations still couple coordinates. Correlated columns can therefore create a shallow direction even though every individual scale looks correct. Only uncorrelated standardized columns give a spherical quadratic objective; scaling alone does not remove this coupling.

□ Proof *Recommended*

$$S = \frac{X^T X}{n} \tag{2.T1}$$

$$D_s = \mathrm{diag}(s_1, \dots, s_d) \tag{2.T2}$$

$$s_j = \sqrt{S_{jj}} \tag{2.T3}$$

Dividing each column by its scale is right multiplication by the inverse diagonal scale matrix. Applying the chain rule, or directly differentiating the transformed squared loss, gives the new Hessian.

$$Z = XD_s^{-1} \tag{2.T4}$$

$$H_Z = \frac{Z^T Z}{n} = D_s^{-1} S D_s^{-1} \tag{2.T5}$$

$$(H_Z)_{jk} = \frac{S_{jk}}{s_j s_k} \tag{2.T6}$$

Unit diagonal entries say that each feature has unit variance; they say nothing about redundancy between features. Two almost perfectly correlated standardized columns still give very unequal eigenvalues: changing their coefficients in opposite directions barely changes predictions. Standardization therefore need not remove the flat direction. Whitening is a different fitted transformation that can make an invertible covariance identity, but small or zero eigenvalues require explicit treatment.

$$H_Z = \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix} \tag{2.T7}$$

$$\lambda_\pm = 1 \pm \rho \tag{2.T8}$$

□

Scaling can improve optimization and make coefficient penalties meaningful across units, but it is not a convergence guarantee. Fit scales within the training boundary, remove or separately treat constant columns, and inspect correlation as well as marginal range.

▶ Example: The validation mean is not a preprocessing parameter

For training values 10, 12 and missing, average the two observed values to obtain imputation value 11. The completed training column is 10, 12, 11; its mean is 11 and its population-standard-deviation convention gives approximately 0.816. Transform a validation value of 30 using these stored quantities; do not refit them. Pooling that validation value with the two observed training values would instead give an imputation mean of approximately 17.333. That change demonstrates leakage without using a validation target. The notebook checks the fitted imputer against training-only statistics.

Interpretation and limits

Standardizing every column once before splitting does not become valid because scaling ignores labels. The transformation still learns from observations reserved for evaluation.

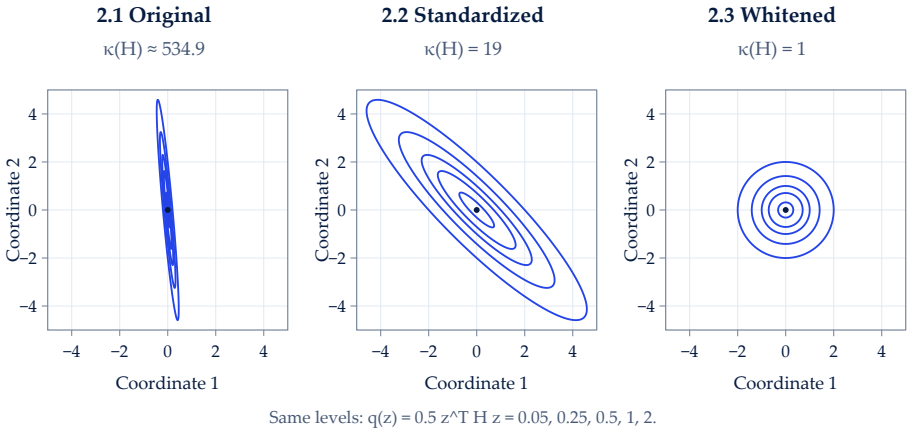


Figure 2.1. Level sets of a coefficient-space quadratic with Hessian diagonal entries 100 and 1 and off-diagonal entry 9. Both scale imbalance and correlation contribute to the elongated contours. The condition number is the ratio of the two positive Hessian eigenvalues.

Figure 2.2. Standardizing the two feature scales makes the Hessian diagonal equal to one but leaves off-diagonal correlation 0.9. The resulting condition number is 19, not one. Unit marginal variance does not remove dependence between the coordinates.

Figure 2.3. An identity Hessian produces circular level sets and condition number one. It is the whitened reference for this example, not the automatic result of feature-wise standardization. Coordinates belong to this transformed parameterization.

Essential

2.2 Least squares and its Gaussian likelihood interpretation

Notation for this section

QR	Orthogonal-triangular factorization for least squares without a normal-equation inverse.
SVD	Singular value decomposition: orthogonal directions and nonnegative singular values; useful for rank-deficient least squares.

Linear regression combines represented features using one coefficient per feature, then adds an intercept. The coefficient-input inner product supplies the feature-dependent part; the intercept supplies an independent offset. Here, linear means linear in the fitted coefficients. Squared inputs and interactions can therefore give a nonlinear relationship in the original variables while preserving a linear model in its parameters.

For unregularized ordinary least squares (OLS), scaling the whole objective by a positive constant preserves minimizers but rescales gradients. Scaling only the loss relative to a fixed penalty generally changes the solution. Here use half the mean squared error (MSE) consistently in gradients and penalties. Solve by stable factorization, not explicit normal-equation matrix inversion.

The objective also follows from a conditional observation model: targets equal their linear predictions plus independent Gaussian errors with common variance. Taking the negative log of the likelihood produces a constant plus a positive multiple of the residual sum of squares. Thus maximizing that likelihood over the coefficients yields least squares. Gaussian noise is an assumption to assess, not a consequence of every measurement process, and least-squares prediction can be useful even when that likelihood interpretation is imperfect.

Bishop [2.1] connects linear regression to Gaussian likelihood, and Golub and Van Loan [2.2] develop stable least-squares algorithms. The normal equations identify an optimum algebraically; QR or singular-value methods can compute it without explicitly inverting a poorly conditioned matrix. Golub and Reinsch's singular-value decomposition algorithm provides a foundational route to least-squares solutions without explicitly inverting the normal-equation matrix [2.8]. Inspecting small singular values also exposes weakly determined coefficient directions; numerical rank tolerance remains an implementation choice, not evidence of a unique scientific interpretation.

Notation for equations (2.2), (2.3)

$n,\ i$	Training sample count and observation index from one through it.
$x_i,\ y_i$	Observation i's input column (d entries) and scalar target.
$w,\ b$	Fitted slope column (d entries, target-per-feature units) and scalar intercept (target units).
$f_{w,b}(x_i)$	Linear prediction: feature-coefficient inner product plus intercept.
$J(w, b)$	Half the mean squared training residual.
$x_i^T w$	Scalar inner product; T transposes the input column to a row.

$$f_{w,b}(x_i) = x_i^T w + b \tag{2.2}$$

$$J(w, b) = \frac{1}{2n} \sum_{i=1}^{n} (y_i - x_i^T w - b)^2 \tag{2.3}$$

w has d coefficients and b is an intercept. J is half the mean squared error, a declared optimization convention. Reported MSE does not include the factor one half.

The factor one half simplifies derivatives and is not part of a reported mean squared error.

Notation for equation (2.4)

$X,\ y$	Training design (n by d) and real target vector (n entries).
$x_i,\ y_i,\ i$	Observation input column, target, and index from one through n.
n	Count of conditionally independent targets in the likelihood.
$w,\ b$	Coefficient vector (d entries) and scalar intercept determining each modeled target mean.
σ^2	Common strictly positive variance of independent conditional Gaussian errors.
$p(y \mid X, w, b, \sigma^2)$	Joint conditional density of the target vector under Gaussian regression.
$\log,\ \pi$	Natural logarithm and circle constant from Gaussian density normalization.

$$\begin{aligned} -\log p(y \mid X, w, b, \sigma^2) = \frac{n}{2} \log(2\pi\sigma^2) \\ + \frac{1}{2\sigma^2} \sum_{i=1}^{n} (y_i - x_i^T w - b)^2 \end{aligned} \tag{2.4}$$

This likelihood assumes conditionally independent Gaussian errors with common positive variance. For fixed variance, maximizing likelihood over w and b is equivalent to minimizing the residual sum of squares.

Targets are modeled as independent Gaussian observations conditional on the design matrix and parameters, with common variance.

◇ Derivation 2.2: From independent Gaussian errors to least squares

Notation for this result

i, n	Observation index from one through the number of training observations.
$\widetilde{X}, \widetilde{x}_i$	Augmented design with n rows and d plus one columns, and the column vector whose transpose forms row i.

θ, y_i	Augmented coefficient vector with d plus one entries and observed scalar target for observation i.
ε_i, σ^2	Independent Gaussian target error and its common strictly positive conditional variance.
$\mathcal{N}$	Normal distribution specified by its mean and variance.
y, p	Target vector with n entries and its conditional joint density under the stated observation model.
$\exp, \pi$	Natural exponential function and the circle constant in the Gaussian density normalization.
ℓ	Natural logarithm of the conditional likelihood, viewed as a function of the model parameters.
$\hat{\theta}, \operatorname{argmin}$	A minimizing fitted coefficient vector and the operation returning the set of minimizing coefficient vectors.

Assumptions. Conditional on the design matrix, target errors are independent Gaussian variables with mean zero and common positive variance. The mean model is linear in augmented coefficients, including an intercept column. For the initial optimization argument, the variance is held fixed.

Maximizing the conditional Gaussian likelihood over the coefficients is equivalent to minimizing the residual sum of squares. This is an equivalence of optimization objectives under a specified probability model, not a proof that actual observation noise is Gaussian.

☐ Derivation *Recommended*

$$y_i = \widetilde{x}_i^T \theta + \varepsilon_i \tag{2.T9}$$

$$\varepsilon_i \mid \widetilde{X} \sim \mathcal{N}(0, \sigma^2) \tag{2.T10}$$

First evaluate each Gaussian density at its observed target. Conditional independence permits multiplying those densities into a likelihood for the whole target vector. The observed targets now stay fixed while the coefficients vary. This is not the probability of one exact real-valued target, which is zero under a continuous distribution; likelihood compares how strongly different parameter values support the observed values through their densities.

$$p(y \mid \widetilde{X}, \theta, \sigma^2) = \prod_{i=1}^{n} \frac{\exp[-(y_i - \widetilde{x}_i^T \theta)^2 / (2\sigma^2)]}{\sqrt{2\pi\sigma^2}} \tag{2.T11}$$

$$-\ell(\theta, \sigma^2) = \frac{n}{2}\log(2\pi\sigma^2) + \frac{\|y - \widetilde{X}\theta\|_2^2}{2\sigma^2} \tag{2.T12}$$

At fixed positive variance, the first term is constant in the coefficients and the residual multiplier is positive. Removing either does not change which coefficient vectors minimize the expression. Multiplying by a positive constant also gives the half-mean-squared objective used throughout this chapter.

$$\hat{\theta} \in \operatorname*{argmin}_{\theta} \|y - \widetilde{X}\theta\|_2^2 = \operatorname*{argmin}_{\theta} \frac{\|y - \widetilde{X}\theta\|_2^2}{2n} \tag{2.T13}$$

Estimating variance introduces a second optimization. If the least-squares residual sum is positive, differentiating in variance gives that sum divided by the sample count. This is the likelihood estimate, not the degrees-of-freedom-corrected unbiased estimate. If the fit has zero residual sum, shrinking variance concentrates the density ever more tightly at the observations. The likelihood then grows without bound, so no finite joint maximum exists with positive variance.

□

Ordinary least squares remains a well-defined optimization problem without a Gaussian assumption. Gaussianity supplies this particular likelihood interpretation. Unequal variances or correlated errors instead lead to differently weighted quadratic objectives when their covariance is known.

◇ Proposition 2.3: Normal equations and orthogonal residuals

Notation for this result

J, θ	Half-mean-squared residual objective and real augmented coefficient vector with d plus one entries.
$\widetilde{X}, y, n$	Augmented n-by-(d-plus-one) design, n-entry target vector, and positive sample count.
$\nabla J, H$	Coefficient gradient and symmetric Hessian matrix of the quadratic objective.
$\hat{\theta}, r$	Any normal-equation solution and its n-entry target-minus-prediction residual vector.
u	Arbitrary augmented coefficient perturbation with d plus one entries.

Assumptions. The augmented design and targets are fixed real arrays. The sample count is positive. Full column rank is required only for uniqueness of coefficients and the displayed inverse formula, not for the normal equations or uniqueness of fitted values.

Each design column describes a direction in which changing a coefficient can move the fitted vector. At a least-squares minimum, the residual is orthogonal to every such direction; otherwise a small coefficient change could reduce squared error. The fitted vector is thus the orthogonal projection onto the design's column space, even when different coefficients represent that same vector.

Proof *Advanced*

$$J(\theta) = \frac{y^T y - 2\theta^T \widetilde{X}^T y + \theta^T \widetilde{X}^T \widetilde{X}\theta}{2n} \tag{2.T14}$$

$$\nabla J = \frac{\widetilde{X}^T(\widetilde{X}\theta - y)}{n} \tag{2.T15}$$

$$H = \frac{\widetilde{X}^T \widetilde{X}}{n} \tag{2.T16}$$

Every Hessian quadratic form is a nonnegative squared norm, so the objective is convex. At a stationary vector the following normal equations hold. Necessity follows from differentiability; sufficiency also follows directly by expanding around that vector.

$$\widetilde{X}^T \widetilde{X}\hat{\theta} = \widetilde{X}^T y \tag{2.T17}$$

$$r = y - \widetilde{X}\hat{\theta} \tag{2.T18}$$

$$J(\hat{\theta} + u) = J(\hat{\theta}) + \frac{\|\widetilde{X}u\|_2^2}{2n} \tag{2.T19}$$

Expand the loss after adding any coefficient perturbation. The term linear in the perturbation is zero because the residual is orthogonal to every design column; the remaining change is a squared norm and cannot be negative. This proves optimality. Uniqueness asks a different question: can a nonzero perturbation leave every fitted value unchanged? Only independent columns rule that out,

making the Gram matrix invertible and every such loss increase strictly positive.

$$\hat{\theta} = (\widetilde{X}^T \widetilde{X})^{-1} \widetilde{X}^T y \tag{2.T20}$$

Rank deficiency separates coefficient identification from fitted-value identification. Adding a null-space vector changes coefficients but leaves every prediction on the training design unchanged. The fitted vector is nevertheless unique: it is the unique projection onto the design's column space. Compute it with QR or singular-value methods. The inverse formula describes the full-rank mathematical solution; it is not an instruction to invert a matrix numerically.

□

Orthogonal residuals describe a best fit within the chosen column space, not absence of every systematic error. They do not imply independence of residuals, causal coefficients, or correct predictions outside the represented sample population.

▶ Example: Solve an intercept and slope together

For inputs zero, one and two and targets one, three and five, append an intercept column of ones. Substituting intercept one and slope two gives predictions one, three and five, so every residual is zero. Duplicate the input column: coefficients two and zero or one and one now produce the same fitted values because their slopes sum to two. Thus these data identify the fitted vector, not a unique coefficient pair. A least-squares factorization handles this rank deficiency without inverting a singular matrix. With noisy targets it balances squared residuals; this constructed exact fit says nothing about future error.

Interpretation and limits

The formula using an inverse of X transpose X requires appropriate rank conditions and is not a recommended numerical implementation. Use a least-squares solver and inspect the design.

Essential

2.3 Gradient descent as a discretized gradient flow

Notation for this section

θ	Real augmented parameter column: intercept, then d coefficients; one-dimensional in the scalar example.
η	Positive learning rate / forward-Euler step in artificial optimization time, not physical elapsed time.

L	Largest positive eigenvalue of the quadratic Hessian; its reciprocal sets the step scale. Not a loss function here.

Build and check the gradient before choosing an optimizer setting. Append a column of ones to the design and stack the intercept with the coefficients. For half mean squared error, multiply the design transpose by predicted minus observed targets, then divide by n. Check each coordinate with equal forward and backward parameter perturbations on fixed data. Try several sizes: large perturbations cause truncation error; extremely small ones can cause floating-point cancellation.

Gradient flow is the continuous-time differential equation that moves parameters opposite the objective gradient. Applying forward Euler with time step eta yields the usual gradient-descent update. This connection explains why the learning rate controls numerical behavior: a step that is too large can overshoot and increase the objective. For a convex quadratic with largest Hessian eigenvalue L greater than zero, a constant learning rate strictly between zero and two divided by L gives stable progress in directions with positive curvature.

Scaling, correlations and rank affect the objective geometry. Standardization can reduce disparities between coordinate scales but does not eliminate poorly conditioned directions caused by correlation. A small learning rate may converge slowly; a large one can diverge. Monitor the objective and compare against a trusted solution rather than assuming a fixed iteration count is sufficient. The analogy to numerical integration does not make an arbitrary optimizer or neural-network landscape equivalent to solving this convex problem.

Boyd and Vandenberghe [2.3] supply the convex-optimization perspective on descent. Hairer, Nørsett, and Wanner [2.6] provide the numerical-analysis framework used here to distinguish a decreasing continuous-time gradient flow from an explicit Euler iteration whose step size can be unstable.

What extends beyond a quadratic

The quadratic calculation exposes the mechanism: a step must be small relative to curvature. The proof gives the corresponding descent lemma for any objective with a globally Lipschitz continuous gradient. Exact full-gradient descent with a positive constant step no larger than the reciprocal of that positive Lipschitz bound cannot increase the objective; it strictly decreases it when the gradient is nonzero. Adding convexity and an attained minimum gives an explicit inverse-iteration bound on the final objective gap, without requiring unique coefficients.

Without convexity, the proved conclusion is weaker: under a finite lower bound, at least one evaluated gradient becomes small at a quantified rate. A small gradient need not mark a minimum. The smooth-descent proof separates these conclusions.
Neither result turns low training loss into evidence of low population risk.

Read the smooth-descent proof

Notation for equations (2.5), (2.6)

$\widetilde{X}$	Augmented design (n by d+1), with a leading ones column.
θ	Parameter column (d+1 entries): intercept, then feature coefficients.
$y,\ n$	Target column (n entries) and training count.
J	Unregularized half-mean-squared training residual.
$\nabla_\theta J$	Gradient column of first partial derivatives for all augmented parameters.
$t,\ \theta_t$	Nonnegative integer iteration and current parameter vector.
η	Positive step multiplying the negative gradient in the update.

$$\nabla_\theta J = \frac{1}{n}\widetilde{X}^T(\widetilde{X}\theta - y) \tag{2.5}$$

$$\theta_{t+1} = \theta_t - \eta\nabla_\theta J(\theta_t) \tag{2.6}$$

The augmented design X tilde includes an intercept column and theta contains its coefficient. The gradient matches J equal to half the mean squared error.

The intercept ordering must match the augmented matrix. This gradient includes no coefficient penalty.

Notation for equations (2.7), (2.8)

θ	Augmented coefficients: continuous trajectory on the left, discrete iterates on the right.
$J,\ \nabla_\theta J$	Differentiable training objective and augmented-parameter gradient.
$\tau,\ t$	Continuous flow time and nonnegative integer iteration index; distinct quantities.
$\theta_t,\ \theta_{t+1}$	Successive parameter vectors, one flow-time step eta apart.
η	Positive forward-Euler step, interpreted as the gradient-descent rate.
$\frac{d\theta}{d\tau}$	Continuous parameter derivative; each component follows its negative partial derivative.
forward Euler	First-order discretization using the derivative at the current iterate.

$$\frac{d\theta}{d\tau} = -\nabla_\theta J(\theta) \tag{2.7}$$

$$\frac{\theta_{t+1} - \theta_t}{\eta} = -\nabla_\theta J(\theta_t) \tag{2.8}$$

Forward Euler discretizes gradient flow at flow times separated by eta. The continuous variable tau is distinct from the integer iteration index t. Stability depends on curvature and step size.

A positive learning rate alone does not guarantee numerical stability; the objective's curvature constrains the step.

◇ Proposition 2.4: Why gradient flow descends but Euler can diverge

Notation for this result

θ, τ, J	Real parameter-column trajectory of fixed finite dimension, continuous optimization flow time, and differentiable scalar objective.
∇J	Gradient column matching the parameter dimension; each entry differentiates the objective with respect to one parameter.
θ_*, H	A minimizing parameter column and its constant symmetric positive-semidefinite square Hessian, with both dimensions matching the parameter count.
t, θ_t, e_t	Nonnegative integer iteration index, Euler iterate, and its error from the chosen quadratic minimizer.
η, I	Positive constant Euler step and the identity matrix matching the parameter dimension.
q_j, λ_j, j	Unit Hessian eigenvector, its nonnegative eigenvalue, and eigen-direction index over the parameter dimension.
$a_{j,t}$	Coefficient of the parameter error in eigen-direction j at iteration t.
L	Largest positive Hessian eigenvalue; this curvature depends on the objective's normalization.

Assumptions. For continuous flow, the objective is continuously differentiable and the trajectory solves its gradient-flow equation on the interval considered. For the discrete stability result, the objective is a convex quadratic with symmetric positive-semidefinite Hessian, at least one positive eigenvalue, and an attained minimum.

Continuous gradient flow cannot increase the objective. Forward Euler contracts every positive-curvature error mode precisely when the constant step

is strictly between zero and twice the reciprocal of the largest Hessian eigenvalue. Flat modes are unchanged.

□ **Proof** *Advanced*

$$\frac{d\theta}{d\tau} = -\nabla J(\theta) \tag{2.T21}$$

$$\frac{dJ(\theta(\tau))}{d\tau} = -\|\nabla J(\theta)\|_2^2 \leq 0 \tag{2.T22}$$

The second equality is the multivariable chain rule followed by substitution of the flow equation. Equality means the gradient is zero at that instant. It does not by itself establish convergence of a general nonlinear trajectory to a global minimum.

$$J(\theta) = J(\theta_*) + \tfrac{1}{2}(\theta - \theta_*)^T H(\theta - \theta_*) \tag{2.T23}$$

$$e_{t+1} = (I - \eta H)\, e_t \tag{2.T24}$$

$$e_t = \theta_t - \theta_* \tag{2.T25}$$

Rotate the parameter error into the symmetric Hessian's orthonormal eigenbasis. In these coordinates, one update cannot mix different curvature directions: each error coordinate obeys its own scalar recurrence. Stability is therefore checked by the multiplier in every direction, rather than by watching one coefficient or only the first loss decrease.

$$H q_j = \lambda_j q_j \tag{2.T26}$$

$$a_{j,t} = q_j^T e_t \tag{2.T27}$$

$$a_{j,t+1} = (1 - \eta\lambda_j)\, a_{j,t} \tag{2.T28}$$

$$|1 - \eta\lambda_j| < 1 \iff 0 < \eta\lambda_j < 2 \tag{2.T29}$$

$$0 < \eta < \frac{2}{L} \tag{2.T30}$$

$$L = \max_j \lambda_j \tag{2.T31}$$

Read the update one eigen-direction at a time. Inside the stated interval, every positive-curvature error coordinate shrinks geometrically. At the upper endpoint, the largest-curvature coordinate flips sign without shrinking; beyond it, that coordinate grows unless it started at zero. Zero-curvature coordinates remain unchanged. Because scaling the objective scales its curvature, residual sum of squares and half-mean-squared error need correspondingly rescaled stability constants.

□

The continuous descent identity does not protect a discretized optimizer from excessive steps. The contraction factor closest to one controls slow convergence; highly unequal positive eigenvalues can make a stable method painfully slow. This explains both the value and the limitations of preprocessing.

▶ Example: One quadratic can converge or diverge

For one half of theta squared, an update multiplies theta by one minus eta. From one, eta 0.5 produces one, 0.5, 0.25, 0.125: the error shrinks. Eta 2.5 produces one, negative 1.5, 2.25, negative 3.375: its sign alternates and magnitude grows. At eta two, it alternates between one and negative one without shrinking. The objective and derivative are correct in all cases; the discretization is unstable in the latter cases. The notebook checks the divergent run and a multivariable gradient before comparing with least squares.

Interpretation and limits

A falling training loss demonstrates progress on the chosen objective, not generalization. A correct derivative also does not guarantee a stable update for every learning rate.

Algorithm 2.1. Full-batch gradient descent

Source and adaptation: Boyd and Vandenberghe [2.3], Chapter 9, for full-gradient descent. This quadratic specialization includes the unpenalized intercept, a gradient-based stopping rule and explicit finite-value and update-budget guards.

Notation and assumptions

$X \in \mathbb{R}^{n \times d},\ y \in \mathbb{R}^n$	Finite training design and target column; $n > 0$ and d is the feature count.

$A = [\mathbf{1}_n, X],\ \theta = (b, w^\top)^\top$ Augmented design and parameter column; b is the first, unpenalized intercept coordinate, w the d slopes; $\mathbf{1}_n$ is the ones column.

$r = A\theta - y, \quad J = \dfrac{\|r\|_2^2}{2n}$ Prediction-minus-target residual and half-mean-squared objective; $\|\cdot\|_2$ is the Euclidean norm.

$g = A^\top r/n, \quad L = \lambda_{\max}(A^\top A/n)$

Joint gradient and positive largest Gram eigenvalue, including the intercept.

η, ϵ, T, t Step with $0 < \eta < 2/L$, positive gradient tolerance, nonnegative integer update budget, and iteration index.

$\theta', r', J',\ \mathcal{T}$ Trial parameter/residual/objective and ordered objective trace; [ ] is empty, and Append adds one last entry.

$\mathrm{finite}, \mathrm{Finish}$ finite checks every entry for a finite number. Finish(status) returns current parameters, trace, current gradient norm when available, and status.

Finish reports converged, budget exhausted or numerical failure. Return-if rows are single-line conditional exits. Before an accepted initial objective the trace is empty. A nonfinite gradient is reported unavailable, never as convergence. Trial values are not committed until their finite-value guard passes.

Algorithm 2.1

Input: Training arrays and finite initial parameters. X, y, θ_0

Validated step, tolerance and update budget.

$0 < \eta < 2/L, \quad \epsilon > 0, \quad T \geq 0$

Output: Last finite intercept and slopes. b, w

Objective trace, available gradient norm and termination status.

$\theta \leftarrow \theta_0$

$\mathcal{T} \leftarrow [\,]$

$r \leftarrow A\theta - y$

$J \leftarrow \|r\|_2^2/(2n)$

Return Finish(numerical failure) if $\neg\,\mathrm{finite}(r, J)$

$\mathcal{T} \leftarrow \mathrm{Append}(\mathcal{T}, J)$

For $t = 0, \ldots, T$

  $g \leftarrow A^\top r/n$

  Return Finish(numerical failure) if $\neg\,\mathrm{finite}(g)$

Algorithm 2.1 (continued)

Return Finish(converged) if $\|g\|_2 \leq \epsilon$

Return Finish(budget exhausted) if $t = T$

$\theta' \leftarrow \theta - \eta g$

$r' \leftarrow A\theta' - y$

$J' \leftarrow \|r'\|_2^2/(2n)$

Return Finish(numerical failure) if $\neg\,\mathrm{finite}(\theta', r', J')$

$(\theta, r, J) \leftarrow (\theta', r', J')$

$\mathcal{T} \leftarrow \mathrm{Append}(\mathcal{T}, J)$

End for

Why this works. Accept is one state transaction using the same old gradient for every parameter. The bound includes the intercept column. Reported MSE is twice J. A finite budget guarantees termination, not convergence. Compare predictions and gradient norm with an orthogonal-triangular (QR) or singular-value (SVD) least-squares solution.

From a full gradient to a mini-batch update

The empirical objective averages one loss per training example. A full-gradient step differentiates every example at the current parameter vector. A mini-batch step instead averages the gradients of a declared nonempty subset. Fresh uniform draws with replacement, independent of the previous history conditional on the current parameters, give an unbiased estimate of the full finite-data gradient. A partly consumed shuffled epoch has a different conditional sampling distribution; do not use the fresh-draw statement without its assumptions. Mini-batch SGD is a general optimization method, not a neural-network-specific operation.

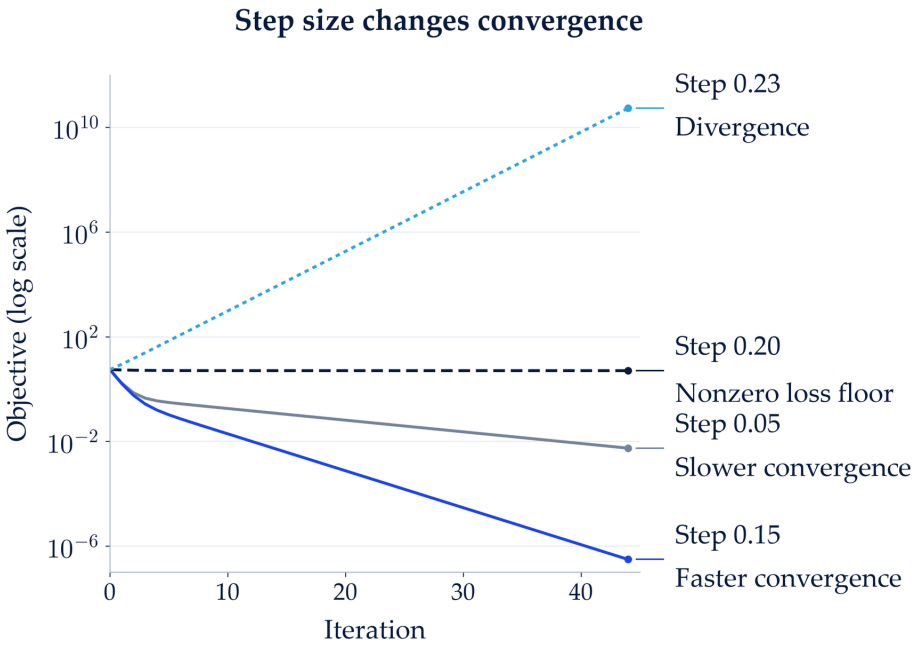


Figure 2.4. Exact gradient-descent iterates on a quadratic with Hessian eigenvalues 1 and 10, from error vector (1, 1). Constant steps strictly between zero and 0.2 converge. At 0.2 the high-curvature component oscillates without decaying; at 0.23 it diverges despite the continuous gradient flow being descending.

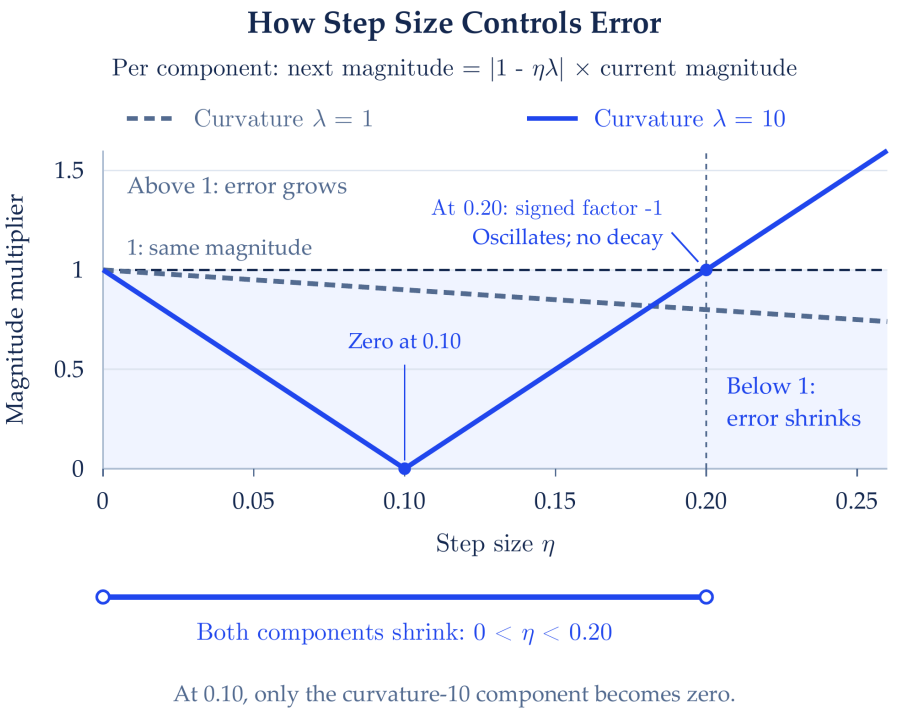


Figure 2.5. Read the vertical error multiplier at a chosen horizontal step size. Curvatures 1 and 10 both shrink only strictly between zero and 0.2 (open-ended bar). At 0.2 the curvature-10 component oscillates without decay; at 0.1 only that component vanishes. One curve below one is insufficient.

▶ Example: Two examples give one update

Take the scalar model prediction w and per-example loss $(w-y)^2/2$, with three targets 0, 3 and 1. At w = 2, the three derivatives are 2, -1 and 1; the full mean is 2/3. A sampled batch containing the first two examples has mean derivative (2-1)/2 = 0.5. With learning rate 0.1, its update gives w = 2-0.1 times 0.5 = 1.95. All batch derivatives use the same old w. A different sample can give a different update, which explains why individual stochastic steps need not decrease the full objective.

Source: Bottou, Curtis and Nocedal (2018), Optimization Methods for Large-Scale Machine Learning, for stochastic-gradient sampling assumptions

and update analysis.

Essential

2.4 Regularization controls the fitted solution

Notation for this section

λ	Nonnegative penalty strength for the declared mean loss; its scale depends on feature units and penalty normalization.
α	Library penalty parameter: training count times the chapter strength for Ridge; equal to that strength for the stated Lasso convention.

Regularization states which solutions we prefer when fitting the data. Ridge discourages large coefficients through their squared norm and often stabilizes correlated-feature estimates. Lasso penalizes the absolute coefficient sum and can make coefficients exactly zero. Neither promises better evaluation performance. Select strength on validation, not the final test set; usefulness depends on signal, sample size and representation.

Hoerl and Kennard [2.4] study the bias-variance trade-off in coefficient estimation; this is distinct from numerical instability. Golub and Reinsch [2.8], Sections 1.1 and 2.3, compute least-squares solutions through singular values without explicitly forming the normal-equation matrix. Discarding tiny singular values changes the effective problem. In the regression experiment, distinguish a stable solver from a selected penalty: numerical stability alone does not establish predictive accuracy, and coefficient shrinkage need not improve every prediction.

We define Ridge using half the mean squared error plus lambda over two times the squared coefficient norm, excluding the intercept. We define Lasso using half the mean squared error plus lambda times the absolute coefficient sum. These conventions make lambda comparable within each stated objective, but library parameters are not universally identical. Scikit-learn Ridge uses a residual sum of squares plus alpha times the squared norm, so its alpha equals n times our Ridge lambda.

Coordinate descent [2.7] uses soft thresholding to obtain zero coefficients; zero is not proof of irrelevance. Zou and Hastie [2.9], Section 2.3, analyze how a positive squared-coefficient penalty encourages correlated predictors to receive similar coefficients. Their grouping bound assumes standardized inputs and same-sign coefficients. Their original rescaled estimator differs from the unrescaled penalized objective. Record scaling and normalization before comparing implementations; grouping does not establish a causal feature group.

Coefficient magnitude depends on feature units, so scale-sensitive penalties require deliberate preprocessing. With strongly correlated variables, Lasso may select one of several near-substitutes; this does not prove that the selected feature is uniquely important or causal. Ridge usually retains all coefficients and shares influence across correlated directions. Regularization choices themselves belong in model selection, and their effects should be assessed with prediction metrics, coefficient behavior and sensitivity to the training sample.

Hoerl and Kennard [2.4] introduced ridge regression to trade bias for stability, and Tibshirani [2.5] introduced the lasso's coefficient-shrinking, sparsity-promoting penalty. These penalties express different preferences; neither guarantees improved performance on every data-generating process. Friedman, Hastie, and Tibshirani [2.7] develop cyclic coordinate-descent methods; the Lasso algorithm here uses its unstandardized-column soft-threshold update with the chapter's mean-loss normalization. The elastic net combines absolute and squared coefficient penalties and can encourage strongly correlated predictors to enter together, extending the comparison between Lasso's sparsity and Ridge's shrinkage [2.9]. This is a useful alternative to test within validation, not a guarantee that grouped selection or predictive improvement occurs for every dataset.

Notation for equation (2.9)

J_{Ridge}	Half-mean-squared training error plus squared-coefficient penalty.
$w,\ b$	Coefficient vector (d entries) and scalar intercept; penalize only coefficients.
$\mathbf{1}_n$	Ones column (n entries); multiplying by b broadcasts the intercept.
λ	Nonnegative mean-loss penalty strength selected using development data.
$\|\cdot\|_2^2$	Sum of squared residual or coefficient entries; dot denotes the argument.

$$J_{\text{Ridge}}(w, b) = \frac{1}{2n}\|y - Xw - b\,\mathbf{1}_n\|_2^2 + \frac{\lambda}{2}\|w\|_2^2 \tag{2.9}$$

The intercept is not penalized. Under this convention, scikit-learn Ridge requires alpha equal to n times lambda to represent the same objective minimizer.

Keeping the library's alpha fixed while changing the training count changes the corresponding mean-loss penalty. State which normalization the experiment holds fixed; the conversion above prevents an apparent disagreement between equivalent objectives.

Notation for equations (2.10), (2.11)

J_{Lasso}	Half-mean-squared training error plus absolute-coefficient penalty.
$w,\ b,\ \mathbf{1}_n$	Coefficient vector (d entries), scalar intercept, and ones column (n entries); the product broadcasts the intercept.
$x_i,\ y_i,\ i$	Input column, target, and index 1 through n; row i of X is the transposed input.
$w_j,\ j,\ d$	Coefficient component, feature index 1 through d, and feature count.
λ	Nonnegative mean-loss penalty; the intercept is unpenalized.
$\|w\|_1$	Sum of absolute coefficients; a nonsmooth penalty allowing exact zeros.
$\|\cdot\|_2^2$	Sum of squared residual entries; dot denotes the supplied residual vector.

$$J_{\text{Lasso}}(w, b) = \frac{1}{2n}\|y - Xw - b\,\mathbf{1}_n\|_2^2 + \lambda\|w\|_1 \tag{2.10}$$

$$J_{\text{Lasso}}(w, b) = \frac{1}{2n}\sum_{i=1}^{n}(y_i - x_i^T w - b)^2 + \lambda\sum_{j=1}^{d}|w_j| \tag{2.11}$$

The first line is vector notation and the second is the same objective expanded observation by observation. The scalar intercept b multiplies the n-dimensional ones vector, broadcasting the same intercept across observations. Only the slopes are penalized.

This convention matches scikit-learn Lasso's alpha when representation and intercept handling also match.

◇ Derivation 2.5: Ridge with an unpenalized intercept

Notation for this result

X, y	Real n-by-d training design and corresponding n-entry target column.
$w, b, \mathbf{1}_n$	d-entry coefficient vector, scalar intercept, and n-entry vector of ones.
λ, J_λ	Strictly positive Ridge penalty strength and the objective under the displayed mean-loss convention.
X_c, y_c	Column-centered training design and centered target vector.
$\widehat{w}, \hat{b}$	Fitted Ridge coefficient vector and its associated unpenalized intercept.
I_d	d-by-d identity matrix acting on the feature-coefficient column.
u	Any nonzero coefficient perturbation vector with d entries.

Assumptions. Use half-mean-squared loss plus one half of a positive penalty strength times the squared coefficient norm. The intercept is unpenalized. The training sample is nonempty. Centering means are learned only from this training sample.

First optimize the unpenalized intercept; this centers the regression problem. Ridge then adds a strictly positive amount to the curvature of every coefficient direction, including directions that dependent columns leave flat. The resulting centered matrix is positive definite, so the coefficient solution is unique for every strictly positive penalty strength.

□ Derivation *Recommended*

$$J_\lambda(w, b) = \frac{\|y - Xw - b\mathbf{1}_n\|_2^2}{2n} + \frac{\lambda}{2}\|w\|_2^2 \tag{2.T32}$$

Differentiate with respect to the intercept. Because it has no penalty, the fitted residuals must sum to zero. This determines the intercept once the coefficients are specified.

$$\hat{b} = \bar{y} - \bar{x}^T\widehat{w} \tag{2.T33}$$

$$X_c = X - \mathbf{1}_n\bar{x}^T \tag{2.T34}$$

$$y_c = y - \bar{y}\mathbf{1}_n \tag{2.T35}$$

Substitute the optimal intercept into the objective and differentiate with respect to the coefficients. Multiplying the resulting equation by the sample count yields the centered Ridge normal equation.

$$(X_c^T X_c + n\lambda I_d)\widehat{w} = X_c^T y_c \tag{2.T36}$$

$$u^T(X_c^T X_c + n\lambda I_d)u = \|X_c u\|_2^2 + n\lambda\|u\|_2^2 > 0 \quad (u \neq 0) \tag{2.T37}$$

Why does Ridge remain unique when the centered design cannot distinguish some coefficient directions? The penalty supplies positive curvature even in those invisible directions. The resulting system is strictly positive definite and hence invertible. To reproduce this optimum with a library using a residual sum rather than a mean, multiply our penalty strength by the sample count; copying the same numerical setting would change the objective.

□

Ridge stabilizes coefficients by making the linear system nonsingular, but does not promise a lower population prediction risk. Feature scaling affects which coefficients are penalized strongly in physical units; select preprocessing and penalty strength together inside validation.

▶ Example: Translate a penalty instead of copying its number

With 200 training observations and lambda equal to 0.1, multiply the chapter's entire Ridge objective by twice n. The half-mean residual term becomes a residual sum of squares, and the squared-coefficient penalty becomes 200 times 0.1, or 20. Thus scikit-learn Ridge needs alpha equal to 20, not 0.1, with the same intercept convention. Lasso's declared mean-loss convention instead gives alpha equal to lambda. Compare the matrix-based and library predictions, as the notebook does; matching parameter names is insufficient. The calculation checks objective equivalence, not that this penalty is the best development choice.

Interpretation and limits

A zero Lasso coefficient is not a causal finding, and the same numeric penalty across libraries need not represent the same optimization problem.

Algorithm 2.2. Least squares or Ridge with an unpenalized intercept

Source and adaptation: Golub and Van Loan [2.2] for QR/SVD least-squares computation, and Hoerl and Kennard [2.4] for Ridge regression. The centered augmented system is derived here using the declared half-mean-squared objective and an unpenalized intercept.

Notation and assumptions

$X \in \mathbb{R}^{n \times d},\ y \in \mathbb{R}^n$	Finite training design and target column; $n > 0$.
$\lambda \geq 0,\quad \bar{x} \in \mathbb{R}^d,\quad \bar{y} \in \mathbb{R}$	Half-MSE slope-penalty strength and training feature/target means.
$X_c \in \mathbb{R}^{n \times d},\ y_c \in \mathbb{R}^n$	Centered training arrays.
$\mathbf{1}_n,\ I_d,\ 0_d$	Ones column, identity matrix, and zero column of the indicated sizes.
$A \in \mathbb{R}^{(n+d) \times d},\ z \in \mathbb{R}^{n+d}$	Vertically augmented least-squares design and target; [U;V] stacks V below U.
$w \in \mathbb{R}^d,\ b \in \mathbb{R},\ \mathcal{D}$	Slope column, unpenalized intercept and numerical rank/residual diagnostics.

LS uses an orthogonal-triangular (QR) or singular-value (SVD) factorization and returns rank/residual diagnostics. At zero penalty use rank-revealing or

minimum-norm least squares. X uses the supplied training-fitted map (identity if unscaled).

Algorithm 2.2

Input: Transformed arrays and fitted scaling. X, y

Penalty and numerical rank tolerance. $\lambda \geq 0$

Output: Coefficients and intercept. w, b

Saved means, fitted scaling and solver diagnostics. $\bar{x}, \bar{y}, \mathcal{D}$

01 $\bar{x} \leftarrow X^\top \mathbf{1}_n / n$

02 $\bar{y} \leftarrow \mathbf{1}_n^\top y / n$

03 $X_c \leftarrow X - \mathbf{1}_n \bar{x}^\top$

04 $y_c \leftarrow y - \bar{y} \mathbf{1}_n$

05 **Augment** $A \leftarrow \begin{bmatrix} X_c \\ \sqrt{n\lambda}\, I_d \end{bmatrix}$

06 **Augment** $z \leftarrow \begin{bmatrix} y_c \\ 0_d \end{bmatrix}$

07 $(w, \mathcal{D}) \leftarrow \mathrm{LS}(A, z)$

08 **Recover** $b \leftarrow \bar{y} - \bar{x}^\top w$

09 **Return** With the supplied scaling. $w, b, \bar{x}, \bar{y}, \mathcal{D}$

Why this works. Do not invert a Gram matrix explicitly. The augmentation minimizes half-MSE plus lambda/2 times the squared slope norm; the intercept is not penalized. Positive lambda gives a unique slope vector. At zero lambda, rank-deficient slopes may be nonunique although training fitted values are unique.

Algorithm 2.3. Cyclic coordinate descent for the Lasso

Source and adaptation: Friedman, Hastie and Tibshirani [2.7], Section 2.1, for coordinate minimization by soft-thresholding; Tibshirani [2.5] for the Lasso objective. This version states the sample-mean convention, partial residual update, KKT test and constant-column policy.

Notation and assumptions

$X_c = [x_1, \dots, x_d], \ y_c \in \mathbb{R}^n$	Centered training design and target; $n > 0$, x_j is a length-n column, d counts retained nonconstant columns.
$w, \bar{x} \in \mathbb{R}^d, \quad b = \bar{y} - \bar{x}^\top w$	Slope and saved training-mean columns; scalar target mean y-bar gives the unpenalized intercept.

$\lambda > 0,\ \epsilon > 0,\ T,\ t,\ j$ Positive penalty and KKT tolerance; T is the positive integer sweep budget, t its index and j the coordinate index.

$r, r_j,\quad a_j = x_j^\top x_j/n,\quad \rho_j$ Current and partial target-minus-prediction residuals; positive column curvature and scalar partial correlation.

$J(w, r) = \dfrac{\|r\|_2^2}{2n} + \lambda \sum_{j=1}^{d} |w_j|$ Penalized half-MSE objective; the norm is Euclidean.

$g(r) = -X_c^\top r/n$ Length-d gradient of the smooth half-MSE term.

$v_j = \begin{cases} |g_j + \lambda \operatorname{sign}(w_j)| & w_j \neq 0 \\ \max(|g_j| - \lambda, 0) & w_j = 0 \end{cases}$ Coordinate Karush-Kuhn-Tucker (KKT) violation. V is $\max_j\ v_j$; an empty maximum is zero; sign(0)=0.

Mathematical core. Remove constant columns and retain their zero-coefficient slots. Center using training means; each retained curvature a_j must be positive. The objective and KKT violation are defined above.

Algorithm 2.3

Input: Centered training arrays and saved training means.

$X_c, y_c, \bar{x}, \bar{y}$

Penalty, tolerance and positive sweep budget. λ, ϵ, T

Output: Slopes and recovered intercept, with convergence or budget status.

$w \leftarrow 0_d$

$r \leftarrow y_c$

Return All columns are constant: zero slopes and the target mean if $d = 0$

For $t = 1, \ldots, T$

  For $j = 1, \ldots, d$

    $r_j \leftarrow r + x_j w_j$

    $\rho_j \leftarrow x_j^\top r_j/n$

    $w_j \leftarrow \dfrac{\operatorname{sign}(\rho_j) \max(|\rho_j| - \lambda, 0)}{a_j}$

    $r \leftarrow r_j - x_j w_j$

  End for

  $V \leftarrow \max_j v_j$

Algorithm 2.3 (continued)

Break if $V \leq \epsilon$

End for

Recover $b \leftarrow \bar{y} - \bar{x}^\top w$

Return Restore removed zero slots; report whether the KKT tolerance was met.

Why this works. The central operation is soft-thresholding the partial correlation, then dividing by positive column curvature. It exactly minimizes one coordinate while the others are fixed; a finite budget does not guarantee the requested KKT tolerance.

Implementation contract. The executable companion validates finite inputs and intermediates, saves the full state before each sweep and rolls back a failed sweep. It records objective values for accepted states and distinguishes convergence, budget exhaustion and numerical failure. A failure before a valid state returns zero slopes, the training target mean and an unavailable residual. These checks do not change the displayed coordinate minimizer.

Example: the centered column (-1, 1), target (-2, 2) and penalty 0.5 give curvature 1 and partial correlation 2. The correlation exceeds the penalty, so soft-thresholding removes 0.5 before dividing by curvature: slope 1.5. The objective drops from 2 to 0.875 and the KKT violation is zero. This solves this coordinate, not every correlated problem uniquely; correlated predictors can have nonunique coefficient solutions.

Interactive companion | Capacity, fit, and generalization

Compare training and validation behaviour as capacity and regularization change. Hold the data conditions fixed for the first comparison, then examine sensitivity to a changed condition. Lower training error alone does not establish better generalization. Use this companion to visualize the trade-off, not to assume that its penalty scale matches the Ridge and Lasso conventions derived here.

Open Generalization and Regularization

Sign-in may be required. Suggested comparisons are in the separate teaching companion.

Essential

2.5 Report errors in a form the application can use

Notation for this section

R^2	Dimensionless coefficient of determination relative to the evaluation-target mean; the superscript belongs to the metric's conventional name, not a generic risk squared.

Interpret an error metric through its units and the mistakes it emphasizes. Mean squared error averages squared residuals, emphasizing large errors. Root mean squared error restores target units. Mean absolute error averages magnitudes, responding linearly to large errors. None describes the entire distribution: inspect residuals for subgroup failures and unacceptable tails even when the average looks adequate.

R squared compares two residual sums on the evaluation sample: the model's squared errors and squared deviations from that sample's target mean. For nonconstant targets it is one minus their ratio, so a model worse than this reference can score below zero. The evaluation mean defines the metric, not a permitted deployed predictor. A deployable constant baseline must instead learn its mean from training observations.

Metrics answer different questions and should be chosen before comparing alternatives. Squared error has a convenient connection to Gaussian likelihood and gradient optimization, but that does not make it the appropriate operational cost in every application. Report the target units, sample count, split and principal metric, and avoid presenting a single score as a universal certificate. In the closing experiment we evaluate the validation-selected procedure once on the held-out test partition and do not use that score to revise the choice.

Bishop’s decision-theoretic treatment [2.1] connects an error criterion to the quantity being estimated. This is why reporting only an optimization loss can be insufficient: the evaluation should also preserve the target’s units and expose consequential residual patterns. Huber's study of location estimation under contaminated normal distributions motivates losses that are quadratic near zero residual and linear in their tails [2.10]. Such a robustness choice changes the fitted objective; it does not justify deleting difficult evaluation cases or claiming that every form of contamination has been handled.

Notation for equations (2.12), (2.13)

$n,\ i$	Number of evaluated observations and their index; each sum runs over all n observations.
MAE	Mean absolute error: average magnitude of the prediction residuals, reported in target units.

RMSE — Root mean squared error: square root of average squared residuals, also reported in target units.

$|\cdot|$ — Absolute value of a scalar residual; signs are removed before its contribution to MAE is averaged.

$$\mathrm{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{2.12}$$

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{2.13}$$

Both metrics have the target's units. RMSE penalizes large residuals more strongly because it squares them before averaging.

Notation for equation (2.14)

R^2 — Coefficient of determination relative to the evaluation-target mean; the superscript is part of the metric's conventional name.

$\bar{y}$ — Arithmetic mean of the targets in this evaluation set, used only as the metric's reference value.

i — Index over all evaluation observations; both sums use the same observations and targets.

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2} \tag{2.14}$$

The formula assumes nonconstant evaluation targets. The mean in the denominator is the evaluation-target mean; a negative result indicates a residual sum of squares larger than this reference.

The denominator must be positive. Constant evaluation targets therefore need an explicit reporting convention. With nonconstant targets, a negative score is valid: the predictions have larger squared error than the evaluation-target mean, not necessarily than a deployable training-mean predictor.

▶ Example: One large residual changes the story

Targets one, two, three and four with predictions one, two, three and ten have only one nonzero residual, of magnitude six. MAE averages magnitudes, giving 1.5; RMSE averages squares before taking a square root, giving three. For R squared, compare residual sum 36 with the targets' squared-deviation sum five around their mean 2.5: the result is negative 6.2. This means the predictions lose to the evaluation-sample mean under squared error. One

failure drives all three summaries, but each describes it differently. R squared has no general zero lower bound.

Interpretation and limits

An R-squared value is neither a percentage accuracy nor proof of causal explanation. A negative held-out value is possible and must not be silently clipped to zero.

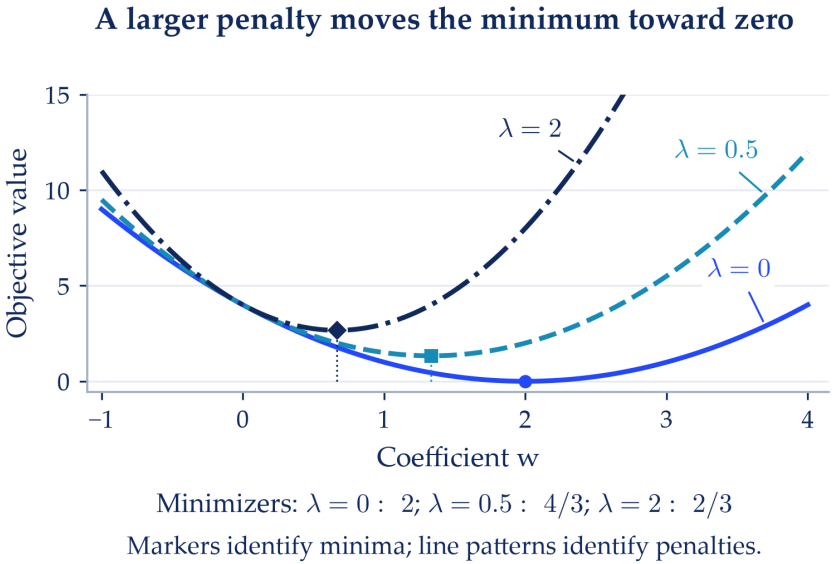


Figure 2.6. Analytical illustration: the data-fit term is $(w-2)^2$ and the penalty is λw^2. The minimizing coefficient is $2/(1+\lambda)$. This scalar example isolates shrinkage; it does not show that every feature or penalty behaves identically.

Table 2.1. Least squares and regularized alternatives

Method	Objective effect	When the distinction matters
Least squares	Minimizes squared residuals without coefficient shrinkage.	Sensitive to collinearity; QR or SVD avoids unstable normal-equation inversion.
Ridge	Adds a squared coefficient penalty.	Stabilizes correlated directions; features must have a deliberate scale.
Lasso	Adds an absolute coefficient penalty.	Can set coefficients to zero; correlated predictors may be selected unstably.
Gradient descent	An optimization procedure, not a different statistical model.	Step size, scaling and stopping determine whether the chosen objective is solved.

Table 2.2. Chapter synthesis

Concept	Main conclusion	Engineering check
Preprocessing	Transformations are fitted components of the model.	Save their training statistics and apply the same transformation at inference.
Least squares	Gaussian constant-variance noise connects squared loss to likelihood.	Inspect residual structure; the noise assumption is not automatic.
Optimization	The gradient defines local descent, not a universal safe step size.	Check loss traces and a numerical derivative on a small problem.
Regularization	Penalty conventions and feature scale affect coefficient meaning.	Record whether losses are sums or means and whether the intercept is penalized.

Dataset: Diabetes progression. See its first-use portrait [2.11].

Case Study: Diabetes: connect the ridge objective to held-out error

Problem and data

The Diabetes benchmark pairs ten baseline measurements from 442 observations with a quantitative progression score recorded one year later. Build a regression experiment that separates numerical correctness from predictive usefulness. A correct optimizer can still yield poor held-out predictions. Load raw features rather than the default globally scaled representation, reserve a final test, and state that this historical exercise neither estimates treatment effects nor validates a clinical forecasting system for a new patient population.

Experimental design

On development observations, standardize predictors and center the target. Solve the ridge normal equations with an unpenalized intercept and compare coefficients and predictions with the library. Explicitly use the summed squared-error objective plus alpha times squared coefficient norm. Then select among four alpha values through three-fold cross-validation of the entire scaling-and-ridge pipeline. Refit the selected pipeline on development data and compare it once with a development-mean reference and ordinary least squares on identical test rows.

Measured Reference Run

Raw predictors were divided into 353 development and 89 final observations with seed 2027. Three-fold cross-validation fitted scaling inside each fold and selected alpha=0.1 from 0.1, 1, 10 and 100. The selected pipeline and the predeclared ordinary least-squares and development-mean references were then evaluated on the same test. MAE and RMSE use progression-score units.

Table 2.3. Measured case-study evidence

Procedure	MAE	RMSE	R-squared
Development mean	62.127	74.016	-0.001
Ordinary least squares	43.002	53.714	0.473
Selected ridge	42.973	53.689	0.473

The selected alpha=0.1 ridge only narrowly improved on least squares; their R-squared values coincide at displayed precision. Separately, alpha=10 reduced the coefficient norm from 77.414 to 43.323 and matched the library solution. The second check verifies shrinkage, not the predictive merit of alpha=10.

Failure Analysis And Limits

For source row 385, the observed score was 219 but selected ridge predicted 139.453. This concrete underestimate remains despite acceptable aggregate error. One residual does not establish a systematic mechanism; neither the small ridge-versus-OLS difference nor coefficient signs justify population superiority or causal interpretation.

Historical saved run; raw-feature loading, fold-local scaling and the summed-loss penalty convention are part of the reproducible procedure. This is not clinical validation.

▶ Example: inspecting the real data

Final-test row 385 has progression score 219 and displayed prediction 139.453. Using rounded operands, target minus prediction and absolute error both equal 79.547; squared error is about 6,327.725. The subtraction compares progression scores. Subtracting BMI from the target would not measure prediction error.

Dataset source and documentation

Companion: Week 02, Real-Data Case Study notebook. Reproduce the declared run, then investigate a changed condition. No fixed benchmark score is promised.

A Controlled Learning Cycle

The multivariate Diabetes case connects preprocessing and penalty normalization to held-out error in target units. Week 2 notebooks verify gradients and regularization conventions; the one-feature laboratory isolates degree and Ridge shrinkage. Record a baseline, one changed setting, the evaluation roles and an explanation of the measured error. Its different basis and split, and the implementation limits below, preclude direct score matching.

Matched practice and simulator protocol · Packaged edition; launch instructions in the guide.

Laboratory 02 | Regression, Optimization, And Regularization

Ridge shrinkage, degree and measured error curves.

Polynomial Ridge regression connects coefficient shrinkage and model flexibility to measured training and validation error.

Open Laboratory 02 | Use the live view for readable controls.

Transfer: Ten-feature Diabetes regression → One-feature Diabetes polynomial regression. No Lasso or gradient-descent implementation; its basis and split differ.

Chapter Capability

Fit a training-only preprocessing pipeline, check the optimizer against a reference solution, and report errors in the target's units.

Week 02: separate practice, solutions and assessment

References

[2.1] Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer. Source / DOI

[2.2] Golub, G. H.; Van Loan, C. F. (2013). Matrix Computations. 4th ed., Johns Hopkins University Press. Source / DOI

[2.3] Boyd, S.; Vandenberghe, L. (2004). Convex Optimization. Cambridge University Press. Source / DOI

[2.4] Hoerl, A. E.; Kennard, R. W. (1970). Ridge Regression: Biased Estimation for Nonorthogonal Problems. Technometrics 12(1), 55-67. Source / DOI

[2.5] Tibshirani, R. (1996). Regression Shrinkage and Selection Via the Lasso. Journal of the Royal Statistical Society: Series B 58(1), 267-288. Source / DOI

[2.6] Hairer, E.; Nørsett, S. P.; Wanner, G. (1993). Solving Ordinary Differential Equations I: Nonstiff Problems. 2nd ed., Springer. Source / DOI

[2.7] Friedman, J.; Hastie, T.; Tibshirani, R. (2010). Regularization Paths for Generalized Linear Models via Coordinate Descent. Journal of Statistical Software 33(1), 1-22. Source / DOI

[2.8] Golub, G. H.; Reinsch, C. (1970). Singular value decomposition and least squares solutions. Numerische Mathematik 14, 403-420. Source / DOI

[2.9] Zou, H.; Hastie, T. (2005). Regularization and Variable Selection Via the Elastic Net. Journal of the Royal Statistical Society Series B: Statistical Methodology 67(2), 301-320. Source / DOI

[2.10] Huber, P. J. (1964). Robust Estimation of a Location Parameter. The Annals of Mathematical Statistics 35(1), 73-101. Source / DOI

[2.11] Efron, B.; Hastie, T.; Johnstone, I.; Tibshirani, R. (2004). Least angle regression. The Annals of Statistics, 32(2), 407-451. Source / DOI

[2.12] Bottou, L.; Curtis, F. E.; Nocedal, J. (2018). Optimization Methods for Large-Scale Machine Learning. SIAM Review 60(2), 223-311. Source / DOI

Advanced

Smooth Descent, Convex Progress And Nonconvex Limits

Chapter 2 analyzes quadratic curvature directly. This section explains what survives for general smooth objectives, what convexity adds, and why a small gradient is not a prediction guarantee.

Smoothness Controls A Local Approximation Error

Let p be a positive parameter count and let F map real p-entry vectors to real objective values. Assume F is continuously differentiable on the whole space. Let L be a strictly positive constant bounding how rapidly its gradient changes in Euclidean norm. The following condition must hold for every pair of parameter vectors u and v, not just at the starting point. Such a function is called L-smooth. It need not be convex, and smoothness of the gradient does not mean that F itself is Lipschitz with the same constant.

$p, F, \nabla F$	Positive parameter dimension, continuously differentiable real objective on the whole p-dimensional real space, and its p-entry gradient.
u, v, L	Arbitrary real p-entry vectors and a strictly positive global gradient-Lipschitz bound.
$\|\cdot\|_2$	Euclidean vector norm.

$$\|\nabla F(v) - \nabla F(u)\|_2 \leq L\|v - u\|_2 \tag{S3.1}$$

A convex region contains the whole straight segment between any two of its points. The assumption may instead be checked on a convex region containing every segment used by the algorithm, but then proving that the iterates stay in that region is an additional obligation. The whole-space statement avoids concealing this obligation. The Hessian is the p-by-p matrix of second partial derivatives; its Euclidean operator norm is the largest factor by which that matrix can stretch a vector's Euclidean length. No Hessian is needed for the proof. For a twice continuously differentiable function, a global upper bound on the Hessian's operator norm is sufficient; a Hessian value at a single point is not.

Lemma: The Quadratic Upper Bound

Under the smoothness assumption, the linear approximation to F at u has the following controlled upper error at v. This is often called the descent lemma, even though no descent direction or convexity has yet been assumed.

$$F(v) \le F(u) + \nabla F(u)^T (v - u) + \frac{L}{2} \|v - u\|_2^2 \tag{S3.2}$$

Proof. Set Delta to v minus u. Along the segment, let r(s) be the change in the gradient from its value at u, where s lies between zero and one. The fundamental theorem of calculus and the chain rule from S1 express the linear-approximation remainder as an integral.

$\Delta, s, r(s)$ Displacement v minus u, segment coordinate in [0,1], and p-entry gradient-change vector.

$$r(s) = \nabla F(u + s\Delta) - \nabla F(u) \tag{S3.3}$$

$$F(v) - F(u) - \nabla F(u)^T \Delta = \int_0^1 r(s)^T \Delta \, ds \tag{S3.4}$$

Cauchy-Schwarz and smoothness bound the absolute integrand by L times s times the squared displacement norm. Integrating s from zero to one contributes the factor one half and proves the lemma. The same argument bounds the absolute remainder, but the upper bound is the part needed for descent.

$$|r(s)^T \Delta| \le L s \|\Delta\|_2^2 \tag{S3.5}$$

One Exact Gradient Step

Let θ_k be the parameter vector at nonnegative integer iteration k. Let g_k denote the exact full gradient there, and let η be a positive constant step size. Substitute the gradient-descent update into the lemma. This gives a decrease whenever η is strictly below two divided by L and the gradient is nonzero. At the upper endpoint the bound only promises nonincrease, not strict progress.

k, θ_k, g_k Nonnegative iteration index, p-entry iterate, and exact p-entry full gradient at that iterate.

η Strictly positive constant gradient-descent step size.

$$g_k = \nabla F(\theta_k) \tag{S3.6}$$

$$\theta_{k+1} = \theta_k - \eta g_k \tag{S3.7}$$

$$F(\theta_{k+1}) \le F(\theta_k) - \eta(1 - L\eta/2) \|g_k\|_2^2 \tag{S3.8}$$

For the conservative choice of η no greater than one divided by L, the coefficient of the squared gradient is at least η divided by two. In the particular choice η equal to one divided by L, this yields the useful progress bound below. Nothing here requires convexity, but exact gradients and the stated smoothness condition are essential.

$$F(\theta_{k+1}) \leq F(\theta_k) - \frac{\|g_k\|_2^2}{2L} \tag{S3.9}$$

Theorem: A Convex Objective-Gap Guarantee

Add two assumptions: F is convex and attains its global minimum at a vector θ_*. Write F_* for this minimum value. For differentiable convex functions, the first-order supporting-plane inequality below holds for every u and v. One can obtain it by applying the defining convex chord inequality to u plus a positive fraction of v minus u, dividing by that fraction and letting it decrease to zero.

θ_*, F_* An attained global minimizing p-entry vector and its objective value; the minimizer need not be unique.

$$F(v) \geq F(u) + \nabla F(u)^T (v - u) \tag{S3.10}$$

Use exact gradient descent with a fixed positive η no greater than one divided by L, starting from any θ_0. After T updates, where T is a positive integer, the final iterate obeys the following bound. It controls objective value, not distance to a particular minimizer, coefficient uniqueness or population prediction risk.

T, θ_0, θ_T Positive number of updates, initial parameter vector, and iterate after those T updates.

$$F(\theta_T) - F_* \leq \frac{\|\theta_0 - \theta_*\|_2^2}{2\eta T} \tag{S3.11}$$

Proof. Define d_k as the Euclidean distance from θ_k to the chosen minimizer. The update's squared-distance expansion is exact. Convexity bounds the current objective gap by the gradient inner product with the displacement from that minimizer. Combine this with the decrease of at least η times the squared gradient divided by two.

d_k Nonnegative scalar distance from the kth iterate to the selected minimizer; not the feature dimension.

$$d_k = \|\theta_k - \theta_*\|_2 \tag{S3.12}$$

$$d_{k+1}^2 = d_k^2 - 2\eta g_k^T(\theta_k - \theta_*) + \eta^2\|g_k\|_2^2 \tag{S3.13}$$

$$F(\theta_{k+1}) - F_* \le g_k^T(\theta_k - \theta_*) - \frac{\eta}{2}\|g_k\|_2^2 \tag{S3.14}$$

$$F(\theta_{k+1}) - F_* \le \frac{d_k^2 - d_{k+1}^2}{2\eta} \tag{S3.15}$$

Sum over k from zero through T minus one. Successive squared distances cancel; the final squared distance is nonnegative and can be dropped. Descent makes the final objective gap no larger than each of the preceding T gaps in this sum. Dividing by T proves the advertised final-iterate result.

$$\sum_{k=0}^{T-1}[F(\theta_{k+1}) - F_*] \le \frac{d_0^2}{2\eta} \tag{S3.16}$$

For η equal to one divided by L, the right side of the final-iterate theorem is L times the squared initial distance divided by twice T. The inverse-T dependence is a sufficient worst-case rate; special problems can converge faster. A known upper bound on initial distance gives a sufficient iteration budget for a chosen objective tolerance. Without that distance bound and an attained minimizer, one should not present this formula as a computable stopping certificate.

Without Convexity: Stationarity Is A Different Conclusion

Now drop convexity and the attained-minimum assumption. Instead suppose F has a finite global lower bound F_{low}. Choose η equal to one divided by L. Summing the one-step descent bound and using this lower bound controls the sum of squared gradient norms, so at least one of the first T evaluated gradients is small.

F_{low} Finite scalar lower bound valid for F at every parameter vector; need not be attained or known to the algorithm.

$$\sum_{k=0}^{T-1}\|g_k\|_2^2 \le 2L[F(\theta_0) - F_{\text{low}}] \tag{S3.17}$$

$$\min_{0 \le k < T} \|g_k\|_2^2 \le \frac{2L[F(\theta_0) - F_{\text{low}}]}{T} \tag{S3.18}$$

The minimum is over evaluated iterates, not necessarily the last iterate. A stationary point has zero gradient; it can be a minimum, a maximum or a saddle. The bound does not establish convergence of the parameter sequence or selection of a global minimum. For example, the scalar smooth objective below has L equal to one and global minimum zero, but starting at θ equal to π gives a zero gradient at its maximum value two. Exact gradient descent remains there.

$\theta, \cos, \pi$ Scalar real parameter in this example, cosine in radians, and the circle constant.

$$F(\theta) = 1 - \cos\theta \tag{S3.19}$$

$F', \sin$ Ordinary scalar derivative and sine in radians.

$$F'(\theta) = \sin\theta \tag{S3.20}$$

Algorithm: Fixed-Step Full-Gradient Descent

For a finite computation, let K be a positive maximum number of updates and let ρ be a nonnegative gradient-norm tolerance. The algorithm returns why it stopped. A stationarity status has only the mathematical meaning justified above; it is not a certificate that a neural model generalizes.

Full-gradient descent with explicit termination status

Source and adaptation: Vandenberghe (2022), *Gradient Method*, and Boyd and Vandenberghe (2004), Chapter 9; see the References at the end of S3. The finite-value, descent-diagnostic and termination policies are specified for this teaching implementation.

Input. A continuously differentiable objective F and exact gradient; initial p-entry vector θ_0; known positive valid smoothness bound L; fixed positive η no greater than $1/L$; positive integer update limit K; nonnegative gradient-norm tolerance ρ.

Output. Last accepted parameter vector, objective/gradient history, update count and status: gradient tolerance, update budget, or numerical failure.

1 Initialize θ to θ_0 and the accepted-update count to zero.

2 Evaluate F and the exact full gradient at the current θ. If either contains a nonfinite value, return numerical failure without accepting another update. Record the values.

3 If the gradient norm is at most ρ, return gradient tolerance. Otherwise, if K updates have already been accepted, return update budget.

4 Form the candidate by subtracting η times the current gradient from θ. Check the candidate first, then its objective. If nonfinite, return numerical failure at the last accepted θ.

5 Check the candidate against the one-step descent inequality, allowing a documented floating-point tolerance in objective units. If a material violation occurs, return numerical failure and investigate derivatives, normalization or the asserted L.

6 Accept the candidate, increase the accepted-update count by one, and return to gradient evaluation.

Verification. At most K updates are accepted. This is unconstrained, deterministic full-gradient descent. Mini-batch gradients, momentum, Adam, nonsmooth penalties and projections require different analyses. A successful observed decrease is a diagnostic, not proof that L is globally valid.

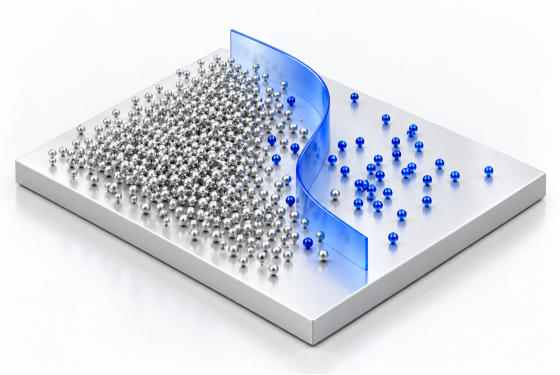
References

Lieven Vandenberghe, Gradient Method, lecture notes (2022), smoothness and gradient-method convergence Source

Stephen Boyd and Lieven Vandenberghe, Convex Optimization (2004), Chapters 3 and 9 Source

CHAPTER 03

Classification And Imbalanced Data



Reading path

Start with Essential. The other levels offer more depth.

Essential
3.1 3.2 3.3 3.4 3.5

Recommended
Proofs 3.4 3.5

Advanced
Proofs 3.2 3.3

Worked example Lab task

A classifier that is usually correct can still miss the events that matter most. The practical task is to distinguish probability quality, ranking quality, and the consequences of a chosen decision threshold.

What You Will Be Able To Do

- Derive binary cross-entropy from a Bernoulli likelihood and implement it stably from logits.

- Compute confusion-matrix metrics and distinguish thresholded decisions from ranking quality.

- Choose a decision threshold using validation data and an explicit cost rule.

- Compare class weighting and training-only resampling without contaminating the evaluation population.

Before you begin: Regression objectives and gradients; Conditional probability and logarithms; Training, validation and test roles.

Compute confusion counts and costs before studying loss curvature or class-weighted probability distortions.

Algorithm scope: Complete teaching implementations: tied-score ROC AUC and validation threshold search. Logistic fitting delegates a numerical optimizer; neither ranking nor empirical cost optimization certifies calibrated probabilities.

Shared chapter notation

d	Positive count of the fixed represented input features, excluding the intercept column.
TP, FP, FN, TN	Counts at one declared threshold on the stated dataset: detected positives, falsely flagged negatives, missed positives, and correctly rejected negatives, respectively. Algorithm 3.2 uses validation counts.
$C_{FP},\ C_{FN}$	Nonnegative costs per false-positive and false-negative decision, respectively, in a common cost unit; their sum is strictly positive.
$+\infty$	Threshold sentinel above every finite score, representing the initial all-negative decision.
$n,\ i$	Training-observation count and index from one through n; each training sum includes all observations.
p_i	Modeled conditional probability of class one for observation i.
$m,\ j$	Number of decreasing-threshold transitions and transition index from one through m.
$\mathbf{1}\{\cdot\}$	Indicator: one when the condition in braces holds and zero otherwise. In the ranking identity the condition specifies a strict win or a tie.

Chapter notation

n	Positive observation count; fitting, validation, or final-evaluation role is specified locally.

A classifier's score can support ranking, probability estimation or a decision under a cost constraint; each use needs a different check. A fraud queue may prioritize precision among its highest-ranked cases, while an intervention system needs a threshold tied to a declared tradeoff. Neither overall accuracy nor a probability-shaped output answers all three questions.

Starting from a conditional Bernoulli model, this chapter separates fitting, scoring and decision-making. Imbalance makes the distinction essential: high accuracy can coexist with neglect of the less frequent class. The notebook's synthetic experiments compute comparisons under a locked test boundary; weighting, resampling and lower thresholds are candidates to evaluate, not guaranteed improvements.

Essential

3.1 Logistic regression models a conditional probability

Logistic regression separates scoring from probability conversion. First encode the binary target as zero or one and identify the positive class. A linear feature combination plus intercept produces a real-valued logit; the sigmoid converts it to a value between zero and one. The model treats that value as the conditional positive-class probability. Whether its probabilities are accurate remains an evaluation question.

The logit is the logarithm of modeled positive-class odds. Holding other represented features fixed, a one-unit increase adds that feature's coefficient to the log-odds; exponentiating the coefficient gives the odds multiplier, not an additive probability change. This representation-dependent association is not automatically causal. Nonlinear feature transformations can yield nonlinear boundaries in the original inputs while the logit remains linear in the represented features.

Estimating a probability and choosing an action are different tasks. Thresholding a score or probability produces a label; some methods supply scores that are not probabilities. A threshold of one half has a decision-theoretic justification only under particular cost and calibration conditions. For multiclass tasks, specify how classes are combined and evaluated before reusing any binary formula.

Cox's analysis of binary sequences [3.1] is a foundational source for logistic regression. The model describes a conditional probability through its log odds; thresholding that probability is a separate decision rather than part of the probability model itself.

Notation for equation (3.1)

$i,\ x_i$	Observation index from one through n and represented input column vector with d entries.
$w,\ b$	Coefficient column vector with d entries and scalar intercept.
z_i	Real-valued logit of observation i; superscript T denotes transpose.

$$z_i = x_i^T w + b \tag{3.1}$$

The represented input determines a real-valued logit.

Notation for equation (3.2)

$z_i,\ \sigma$	Real logit and logistic sigmoid function, respectively.
e	Base of the natural exponential; the minus sign negates the logit.

$$p_i = \sigma(z_i) = \frac{1}{1 + e^{-z_i}} \tag{3.2}$$

The sigmoid maps the logit to a modeled positive-class probability. This construction alone does not establish calibration.

Notation for equation (3.3)

$p_i,\ z_i$	Modeled class-one probability, strictly between zero and one, and its logit.
$\log$	Natural logarithm; the probability ratio is the modeled positive-class odds.

$$\log \frac{p_i}{1 - p_i} = z_i \tag{3.3}$$

The logit is the modeled log-odds of the positive class.

▶ Example: The same probability supports different decisions

Exponentiate logit zero to obtain odds one, then divide one by one plus one: probability 0.5. A logit equal to the natural logarithm of three gives odds three and probability three quarters. With positive decisions at or above the threshold, both pass 0.5 and neither passes 0.8. Changing that threshold changes actions, not the scores or probability estimates. For urgent support requests, a capacity-limited team might instead review the highest-ranked requests. Evaluate the three objects separately: ranking for the score, probability quality for the forecast, and costs or capacity for the action.

Interpretation and limits

An output inside zero and one is not automatically a well-calibrated probability. A sigmoid defines a model form, not a certificate that its numerical confidence matches observed frequencies.

Essential

3.2 Bernoulli likelihood leads to binary cross-entropy

Notation for this section

p	Scalar Bernoulli success probability, between zero and one. The later vector-gradient formula redefines p as a probability column.

Binary cross-entropy comes from asking how probable the observed labels are under the model. A Bernoulli outcome is one with probability p and zero with probability one minus p. Its mass uses the observed target as an exponent. For conditionally independent examples, multiply their masses; a negative logarithm converts the product to a sum. Dividing by n gives mean binary cross-entropy, or mean log loss.

Cross-entropy penalizes confident wrong predictions strongly. Predicting 0.99 when the outcome is zero incurs a much larger loss than predicting 0.6. This is appropriate when the objective is probability quality under log loss, but it should not be confused with a direct implementation of every operational cost. The unregularized logistic gradient has a simple residual form: represented inputs transposed, multiplied by predicted probabilities minus binary targets, and divided by n.

Evaluate log loss from logits using stable softplus minus target times logit, rather than taking a logarithm after a sigmoid has rounded to zero or one. Add the declared regularization term and its gradient separately. Numerical stability does not guarantee a finite optimum: on strictly separable training data, unregularized logistic coefficients can grow while loss approaches an unattained infimum. Check both arithmetic and the optimization problem before interpreting a solver warning.

Gneiting and Raftery [3.2] place logarithmic loss within the theory of proper scoring rules. This perspective explains why a probability forecast should be evaluated for its probabilistic quality, not only for the class label obtained after thresholding. Brier's probability-verification score offers another way to evaluate confidence rather than thresholded labels [3.7]. In the common binary single-event convention, it averages the squared difference between the positive-class probability and the zero-or-one outcome; this differs by a factor of two from summing the squared errors over both binary outcome categories.

Notation for equation (3.4)

$Y,\ y_i$	Binary target random variable and observed zero-or-one target.
$x_i,\ p_i$	Represented input and modeled positive-class probability for observation i.
$P(\cdot \mid x_i)$	Conditional probability mass under the Bernoulli model at the specified input.

$$P(Y = y_i \mid x_i) = p_i^{y_i}(1 - p_i)^{1-y_i} \tag{3.4}$$

This is the probability mass assigned to one binary observation, not the training objective.

Notation for equation (3.5)

J	Mean binary cross-entropy without regularization.
$y_i,\ p_i$	Observed binary target and modeled positive-class probability.
$\log$	Natural logarithm; losses are measured in nats.

$$J = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log p_i + (1 - y_i)\log(1 - p_i)] \tag{3.5}$$

Averaging the negative log-likelihood gives binary cross-entropy. Natural logarithms express the loss in nats.

The likelihood factorization assumes conditionally independent observations under the model.

Notation for equation (3.6)

$J,\ n,\ i$	Mean unregularized log loss, sample count and observation index.
$z_i,\ y_i$	Real logit and binary target of observation i.
$\log,\ e$	Natural logarithm and exponential base; log of one plus exp is softplus.

$$J = \frac{1}{n}\sum_{i=1}^{n}[\log(1 + e^{z_i}) - y_i z_i] \tag{3.6}$$

Evaluate the equivalent logits-based objective using a stable softplus or logaddexp implementation.

Use logaddexp(0, z) or an equivalent stable routine rather than computing exp(z) directly at extreme logits.

Notation for equation (3.7)

$\theta,\ J$	Parameter column with d plus one entries, intercept first, and mean binary cross-entropy, respectively.
$\widetilde{X}$	Augmented n-by-(d+1) design matrix, with a leading column of ones.
$p,\ y$	Length-n columns of modeled probabilities and observed binary targets.
$n,\ \nabla_\theta$	Sample count and gradient with respect to the parameter vector; T denotes transpose.

$$\nabla_\theta J = \frac{1}{n}\widetilde{X}^T(p - y) \tag{3.7}$$

The augmented design matrix includes an intercept. This gradient excludes any regularization term.

◇ Derivation 3.1: Bernoulli likelihood and the probability targeted by log loss

Notation for this result

$\mathcal{L}$	Conditional Bernoulli likelihood of the observed target sequence under the proposed probabilities.
n, i, y_i, p_i	Sample count, observation index from one through n, binary target, and modeled positive-class probability.
$J, \log$	Mean negative log-likelihood and the natural logarithm, expressing losses in nats.
X, x, Y	Random input, its fixed evaluation value, and binary target at that value.
q	Actual conditional positive-class probability at the fixed input.
$p, r(p; q)$	Proposed positive-class probability and its conditional expected log loss.

Assumptions. Targets are zero or one, and are conditionally independent given the modeled inputs and parameters. Predicted probabilities lie strictly between zero and one. For the population minimization argument, hold an input fixed and assume its actual positive-class probability lies strictly between zero and one. Boundary cases follow by limits.

Mean binary cross-entropy is the mean negative Bernoulli log-likelihood. At a fixed input, its expected value is uniquely minimized by the true conditional positive-class probability, not necessarily by the most common class label.

□ Derivation ▪ *Essential*

The observed binary target selects one of the two Bernoulli probabilities through its exponent. Conditional independence converts the joint likelihood into a product; the logarithm converts that product into a sum.

$$\mathcal{L} = \prod_{i=1}^{n} p_i^{y_i} (1 - p_i)^{1 - y_i} \tag{3.T1}$$

$$J = -\frac{1}{n} \log \mathcal{L} = -\frac{1}{n} \sum_{i=1}^{n} \left[y_i \log p_i + (1 - y_i) \log(1 - p_i) \right] \tag{3.T2}$$

For a fixed input, average the two possible losses with their actual conditional probabilities. This distinguishes the actual outcome distribution from the probability being proposed by the model.

$$q = \Pr(Y = 1 \mid X = x) \tag{3.T3}$$

$$r(p; q) = -q \log p - (1 - q) \log(1 - p) \tag{3.T4}$$

$$\frac{\partial r}{\partial p} = \frac{p - q}{p(1 - p)} \tag{3.T5}$$

The derivative's denominator is positive inside the probability interval, so its numerator determines the direction of improvement. Below the true conditional probability the risk decreases as the proposed probability rises; above it, the risk increases. The true probability is therefore the minimum. At q=0 or 1, the infimum is approached as p tends to q. Averaging these pointwise comparisons over inputs gives the population result.

A restricted parametric family may not contain that target. It then finds the best approximation under its population objective, and a finite-data optimizer only estimates that approximation. Neither the sigmoid output range nor use of cross-entropy proves calibration on a new population.

□

A probability forecast carries more information than a hard label. Two forecasts can make the same decisions at one threshold while receiving very different log losses. Fit probability quality and choose operational decisions as separate, explicitly evaluated steps.

◇ Proposition 3.2: Logistic convexity, rank, and the missing finite optimum

Notation for this result

$\widetilde{X}, \widetilde{x}_i$	Fixed augmented design (n by d+1) and the input column whose transpose is row i.
θ, z_i	Augmented coefficients and observation-i scalar logit.
y_i, J	Observed binary target and mean unregularized logistic cross-entropy.
p, y	Modeled positive probabilities and binary-target columns (n entries each).
H, W	Symmetric coefficient Hessian (d+1 by d+1) and diagonal Bernoulli-variance weights (n by n), strictly positive at finite logits.
diag	Diagonal matrix from listed entries; all off-diagonal entries are zero.
u	Arbitrary coefficient direction (d+1 entries).

s_i	Observation-i target in minus-one/plus-one coding.
v	Real direction column (d+1 entries) strictly separating every training example.
a	Positive multiplier of the separating direction.

Assumptions. The input representation is fixed, and each logit is an affine function of the coefficient vector. Training targets are binary. All coefficients are finite when derivatives are evaluated. The statement is about logistic regression in fixed features, not a neural network that learns those features jointly.

Logistic cross-entropy is convex in its coefficients, and full column rank makes it strictly convex at finite coefficients. Strict convexity excludes two distinct finite minimizers if a minimizer exists; it does not prove existence. On separable data, improving the margin can keep decreasing loss while coefficients escape to infinity. The two questions must be checked separately.

□ **Proof** *Advanced*

$$z_i = \widetilde{x}_i^T \theta \tag{3.T6}$$

$$p_i = \frac{1}{1 + e^{-z_i}} \tag{3.T7}$$

$$J(\theta) = \frac{1}{n} \sum_{i=1}^{n} [\log(1 + e^{z_i}) - y_i z_i] \tag{3.T8}$$

Differentiate one operation at a time. Softplus contributes a sigmoid; subtracting the target-weighted logit leaves probability minus target. Differentiating the linear logit brings in the input vector, yielding the gradient. Differentiate again: sigmoid contributes p times one minus p, and the input appears a second time, yielding the Hessian. These two input factors explain its outer-product structure.

$$\frac{dp_i}{dz_i} = p_i(1 - p_i) \tag{3.T9}$$

$$\nabla J = \frac{\widetilde{X}^T (p - y)}{n} \tag{3.T10}$$

$$H = \nabla^2 J = \frac{\widetilde{X}^T W \widetilde{X}}{n} \tag{3.T11}$$

$$W = \operatorname{diag}(p_i(1-p_i))_{i=1}^n \tag{3.T12}$$

$$u^T H u = \frac{1}{n}\sum_{i=1}^n p_i(1-p_i)(\widetilde{x}_i^T u)^2 \geq 0 \tag{3.T13}$$

The nonnegative Hessian quadratic form establishes convexity. With independent design columns, every nonzero coefficient direction changes at least one observation's score, making the form strictly positive; rank deficiency leaves flat directions. This concerns curvature at finite coefficients, not behavior at infinity. Even a full-rank design need not make the loss coercive: coefficient magnitudes can grow without the objective growing.

$$s_i = 2y_i - 1 \tag{3.T14}$$

$$s_i \widetilde{x}_i^T v > 0 \quad \text{for every } i \tag{3.T15}$$

$$J(av) = \frac{1}{n}\sum_{i=1}^n \log(1 + e^{-a s_i \widetilde{x}_i^T v}) \longrightarrow 0 \quad (a \to \infty) \tag{3.T16}$$

Along a strictly separating direction, increasing coefficient magnitude drives every loss term toward zero, but no finite coefficients attain it. A positive squared-norm penalty on every coefficient restores coercivity and strict convexity. Exempting the intercept changes the argument: with only one observed class, that unpenalized intercept can still escape to infinity.

□

Convex does not mean numerically effortless, uniquely parameterized, or guaranteed to have a finite unregularized solution. Large coefficients and an apparently stalled solver may reflect separation rather than a software bug. Evaluate softplus stably and report the regularization convention.

▶ Example: Confidence changes log loss without changing a label

At threshold 0.5, probabilities 0.6 and 0.99 both produce a positive label. For a positive target, take the negative natural logarithm of each probability: losses are approximately 0.511 and 0.010. For a negative target, use one minus each probability instead: approximately 0.916 and 4.605. Confidence helps when

correct and hurts when wrong, although the hard labels agree. The notebook also tests logits of positive and negative one thousand; stable evaluation gives finite losses, near zero for correct confident predictions and approximately one thousand for incorrect ones. Label accuracy discards this distinction.

Interpretation and limits

Do not compute log loss by taking logarithms of numerically rounded sigmoid outputs when logits are available. Use a stable logits-based implementation and check its gradient.

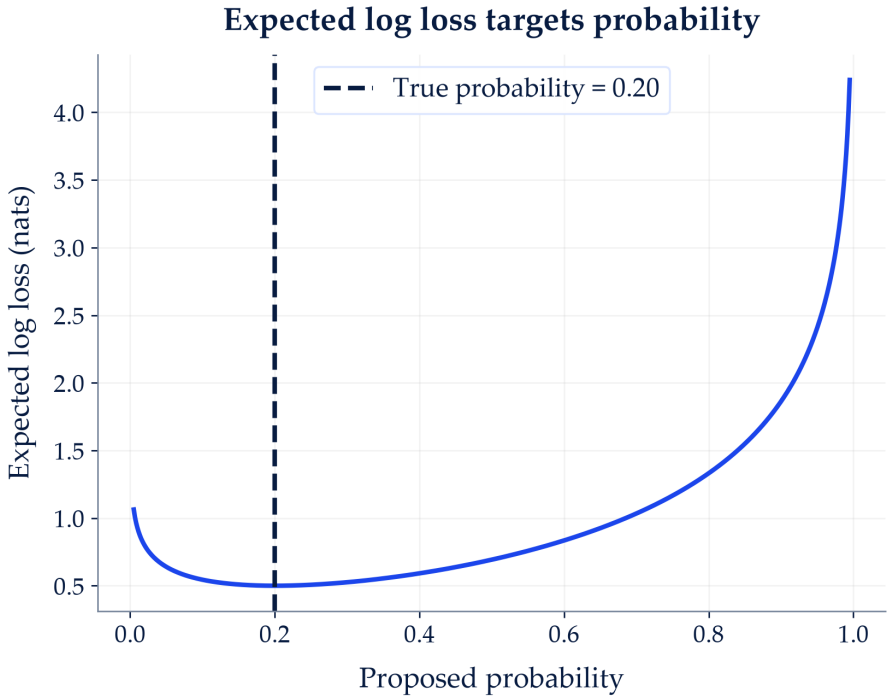


Figure 3.1. Exact conditional expected binary log loss when the positive-class probability is 0.20. Its minimum occurs at that true probability. This population calculation explains probability quality; it is not a measurement of finite-data classifier performance.

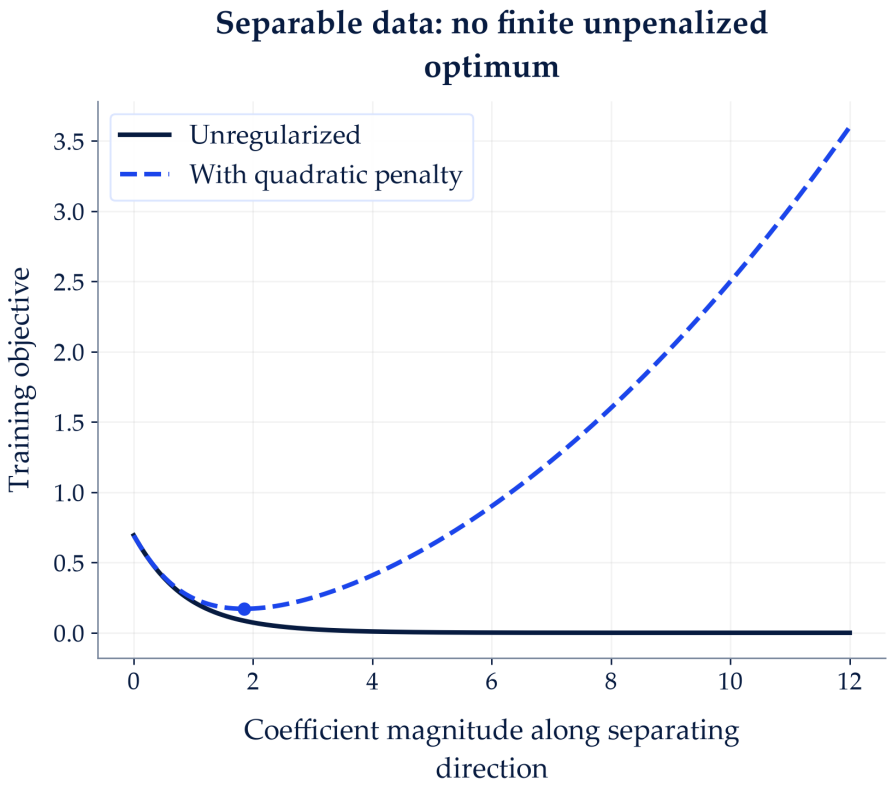


Figure 3.2. Mean logistic loss on separable observations with inputs (-2, -1, 1, 2), targets (0, 0, 1, 1), and one positive slope. Unregularized loss approaches zero only as the slope grows without bound. Adding 0.05 times half the squared slope creates a finite optimum, marked by the grid minimum.

A generative route to a class score

Logistic regression models class probabilities directly from input features. Naive Bayes instead estimates a class prior and a feature distribution within each class, then applies Bayes' rule. Its simplifying assumption is that the measured features are conditionally independent given the class. Multiply the class prior by the feature likelihoods to obtain an unnormalized score; divide by the sum across classes when model probabilities are required. Conditional independence is a modeling assumption to examine, not a property created by the algorithm.

▶ Example: Combine two feature likelihoods

Consider two binary indicators, both observed as one. Let the two class priors be 0.5 each. Their probabilities of observing a one are (0.8, 0.6) in class A and (0.1, 0.2) in class B. Under conditional independence, the scores are 0.5 times 0.8 times 0.6 = 0.24 and 0.5 times 0.1 times 0.2 = 0.01. Normalization gives probability 0.24/(0.24+0.01) = 0.96 for A under this model. Dependence between the indicators can make that probability overconfident, so evaluate calibration as well as class decisions.

Estimate the prior and likelihood parameters only from training data. The text-classification chapter develops count-based Naive Bayes, smoothing and sparse computation, and compares it with logistic regression and a linear SVM.

Source: Manning, Raghavan and Schutze (2008), Introduction to Information Retrieval, Chapter 13, for class priors, conditional-independence classification and text-specific likelihood models.

Essential

3.3 Confusion matrices and ranking metrics answer different questions

Precision asks whether predicted positives are correct; recall asks how many actual positives were found. They use different denominators. At a specified threshold, the confusion matrix supplies their counts: true and false positives, and true and false negatives. State the row-column orientation and an explicit policy for zero denominators. Neither ratio substitutes for the other.

There are three separate evaluation questions: how good are the decisions at a declared threshold, how well do the scores rank positives above negatives, and how accurate are the probability estimates? Confusion-matrix metrics answer the first question; area under the receiver operating characteristic curve (ROC AUC) and average precision (AP) summarize ranking; log loss and calibration checks evaluate probability quality. We develop each distinction explicitly below.

Fawcett [3.3] develops ROC analysis and the ranking interpretation of AUC, while Davis and Goadrich [3.4] explain the relationship between ROC and precision-recall curves. These sources motivate reporting operating-point counts alongside ranking measures, especially when the positive class is uncommon. Saito and Rehmsmeier demonstrate why precision-recall plots can expose a rare-positive classifier's false-alarm burden more directly than a visually favorable ROC curve [3.8]. Their comparison reinforces reporting prevalence and operating-point counts alongside ranking summaries, rather than treating either curve as a universal decision criterion.

From scores to a confusion matrix

Notation for equation (3.8)

$\widehat{y}_i(\tau)$	Predicted zero-or-one label for observation i at threshold tau.
$s_i,\ \tau$	Real-valued score oriented toward the positive class and decision threshold.

$$\widehat{y}_i(\tau) = \mathbf{1}\{s_i \geq \tau\} \tag{3.8}$$

The score is oriented so larger values favor class one; equality is classified positive.

Here s_i is a prediction score, not the signed target used in the preceding convexity proof. Every score is finite.

Table 3.1. Confusion matrix: rows are actual classes; columns are predicted classes

Actual class	Predicted positive (1)	Predicted negative (0)
Positive (1)	TP: correctly detected	FN: missed positive
Negative (0)	FP: false alarm	TN: correctly rejected

All counts below use the same evaluation observations and the same threshold. The total sample count is the sum of the four cells. Scores may be calibrated probabilities, uncalibrated decision-function values, or another ranking score; the confusion counts require only a declared decision rule.

One metric, one denominator

Notation for equation (3.9)

accuracy	Correct decisions divided by all evaluated observations; undefined for an empty set.

$$\text{accuracy} = \frac{TP + TN}{TP + FP + FN + TN} \tag{3.9}$$

Accuracy measures the fraction of all observations assigned the correct label.

Notation for equation (3.10)

precision — Fraction of predicted positives that are correct; undefined if no positives are predicted.

$$\text{precision} = \frac{TP}{TP + FP} \tag{3.10}$$

Precision asks how many predicted positives are actually positive.

Notation for equation (3.11)

TPR — True-positive rate, also called recall or sensitivity; undefined if there are no actual positives.

$$\text{recall} = \text{TPR} = \frac{TP}{TP + FN} \tag{3.11}$$

Recall, sensitivity and true-positive rate name the same quantity: the fraction of actual positives detected.

Notation for equation (3.12)

TNR — True-negative rate, or specificity; undefined if there are no actual negatives.

$$\text{specificity} = \text{TNR} = \frac{TN}{TN + FP} \tag{3.12}$$

Specificity is the fraction of actual negatives correctly rejected.

Notation for equation (3.13)

FPR — False-positive rate among actual negatives; undefined if there are no actual negatives.

$$\text{FPR} = \frac{FP}{FP + TN} \tag{3.13}$$

False-positive rate is the fraction of actual negatives incorrectly flagged; it equals one minus specificity, not one minus precision.

Notation for equation (3.14)

F_1 F1 score; the count formula is undefined when all three displayed counts are zero.

$$F_1 = \frac{2TP}{2TP + FP + FN} \tag{3.14}$$

When precision and recall are positive, F1 is their harmonic mean. The count form also handles some cases where that mean would need a limiting convention.

True negatives do not enter F1. Declare a reporting convention when its denominator is zero.

Notation for equation (3.15)

$\mathrm{TPR},\ \mathrm{TNR}$ Recall for the positive class and recall for the negative class at the same threshold.

$$\text{balanced accuracy} = \frac{\mathrm{TPR} + \mathrm{TNR}}{2} \tag{3.15}$$

Binary balanced accuracy gives the two class-conditional recalls equal weight, rather than weighting by class frequency.

Both classes must be represented to interpret their average as this binary balanced-accuracy measure.

Report undefined ratios explicitly, or state the numerical replacement used by the software. For example, with no predicted positives, precision is undefined rather than an observed zero; a library may return zero under a chosen convention. An all-negative rule on a population with 1% positives has 99% accuracy and zero recall. Balanced accuracy is 0.5 when both classes are represented. This is why a single favorable-looking number is insufficient.

Table 3.2. Choosing a classification measure

Measure	What it answers	Main limitation
Accuracy	How often is the label correct?	Can conceal failure on a rare class.
Precision	How reliable is a positive flag?	Depends on prevalence; misses are not in its denominator.
Recall / TPR	What fraction of positives is found?	Does not measure the false-alarm burden.

Measure	What it answers	Main limitation
Specificity / TNR	What fraction of negatives is rejected?	Does not measure missed positives.
FPR	What fraction of negatives is flagged?	Even a small rate can create many false alarms.
F1	How do precision and recall combine?	Ignores true negatives and explicit error costs.
Balanced accuracy	How good is recall for each class?	Equal class weight is not a deployment cost model.
ROC AUC	How well are positives ranked above negatives?	No calibration claim or chosen operating threshold.
Average precision	How pure is the ranked positive retrieval?	Depends on prevalence and the AP convention.
Log loss	How good are assigned probabilities?	Not a direct operational-cost objective.

ROC: a threshold sweep, not one decision

ROC abbreviates receiver operating characteristic. Its horizontal axis is false-positive rate and its vertical axis is true-positive rate. Begin with a threshold above the largest finite score: no observation is flagged, so the curve starts at (0, 0). Lower the threshold through the distinct score values. Each newly admitted positive increases TPR, and each newly admitted negative increases FPR. Once all observations are flagged, the endpoint is (1, 1). Both classes must be present for these rates to define an ROC curve.

The upper-left region means high detection with few false alarms; the equal-TPR-and-FPR diagonal represents an uninformative ranking in expectation. Compare curves where the application will operate. Crossing curves can give the larger full-area score to a model that is worse under a stringent false-positive constraint. An empirical ROC curve need not be concave; concavity is not required for its area calculation.

Changing only the decision threshold moves the operating point on a fixed curve. It does not change the scores or the full ROC AUC. Changing the model, observations or scores can change the curve. In software, pass the continuous positive-class scores to roc_curve and roc_auc_score, not already-thresholded

zero-or-one predictions; hard labels discard the ranking information needed for a full sweep.

AUC: area and pairwise ranking

Notation for equation (3.16)

$\mathrm{AUC}_{\mathrm{ROC}}$	Area under the receiver operating characteristic curve, between zero and one.
u	False-positive rate serving as the horizontal integration variable.
$\mathrm{TPR}(u)$	True-positive rate on the completed ROC curve at horizontal coordinate u.

$$\mathrm{AUC}_{\mathrm{ROC}} = \int_0^1 \mathrm{TPR}(u)\, du \tag{3.16}$$

ROC AUC integrates detection rate over the false-positive-rate axis of the completed ROC curve.

For tied scores, the completed empirical curve uses straight segments; vertical segments contribute zero area.

Notation for equation (3.17)

$\widehat{\mathrm{AUC}}_{\mathrm{ROC}}$	Empirical area under the piecewise-linear ROC curve.
$F_j,\ T_j$	False-positive and true-positive rates at the j-th point, ordered by decreasing threshold.

$$\widehat{\mathrm{AUC}}_{\mathrm{ROC}} = \sum_{j=1}^{m} (F_j - F_{j-1}) \frac{T_j + T_{j-1}}{2} \tag{3.17}$$

For an empirical ROC curve, join consecutive threshold points linearly and sum trapezoid areas. Group tied scores before forming the points.

Start at F0 = T0 = 0; end at Fm = Tm = 1. All observations with the same score enter together. Both classes must be present.

A threshold cannot separate observations with the same score, so move the whole tied block together. The straight segment between its endpoint counts assigns half credit to tied positive-negative pairs in the area calculation; it does not claim a deterministic score threshold can select part of that block. Vertical segments add no area, while horizontal segments add height times width.

Notation for equation (3.18)

$n_+,\ n_-$	Numbers of actual positive and negative observations, both strictly positive.

$a,\ b$	Positive-observation index from one through n-plus and negative-observation index from one through n-minus, respectively.
$s_a^+,\ s_b^-$	Positive-observation and negative-observation scores; superscripts label classes, not score signs.
$\widehat{\mathrm{AUC}}_{\mathrm{ROC}}$	Empirical mean ranking credit over all positive-negative pairs.

$$\widehat{\mathrm{AUC}}_{\mathrm{ROC}} = \frac{\sum_{a=1}^{n_+}\sum_{b=1}^{n_-}\left[\mathbf{1}\{s_a^+ > s_b^-\} + \frac{1}{2}\mathbf{1}\{s_a^+ = s_b^-\}\right]}{n_+ n_-} \tag{3.18}$$

Equivalently, average positive-negative ranking comparisons, awarding one point for a strict win and half a point for a tie.

These are unweighted binary formulas. Do not impose an arbitrary order inside a tied score block.

AUC = 1 means every observed positive outranks every observed negative. AUC = 0 means the ordering is completely reversed. An AUC of 0.5 is chance-level average pairwise ranking, not proof that a model is literally random or that its ROC curve lies on the diagonal at every threshold. Reversing score orientation replaces AUC by one minus AUC under the same half-tie convention; orientation must be fixed before final evaluation.

A strictly increasing score transformation preserves every pairwise ordering and tie, so it preserves ROC AUC. It need not preserve the numerical probabilities: ranking can stay correct while confidence becomes badly wrong. AUC therefore cannot choose a probability threshold or establish accuracy or calibration. The proposition below derives this ranking interpretation from the trapezoid areas.

A six-observation example, including a tied score

Table 3.3. Fixed scores and labels

Observation	True class	Score
A	1: positive	0.90
B	0: negative	0.80
C	1: positive	0.80
D	0: negative	0.60
E	1: positive	0.40
F	0: negative	0.20

There are three positives and three negatives. Predict positive at score greater than or equal to the threshold. The two observations scored 0.80 enter at the same step. The following table lists every distinct operating point, including the initial endpoint above the maximum score.

Table 3.4. The complete threshold sweep

Threshold	TP	FP	FN	TN	TPR	FPR
Above 0.90	0	0	3	3	0	0
0.90	1	0	2	3	1/3	0
0.80	2	1	1	2	2/3	1/3
0.60	2	2	1	1	2/3	2/3
0.40	3	2	0	1	1	2/3
0.20	3	3	0	0	1	1

The nonzero trapezoid contributions are 1/6, 2/9 and 1/3, giving AUC = 13/18, approximately 0.722. The pairwise calculation agrees: the positive scored 0.90 beats all three negatives; the positive scored 0.80 ties one and beats two; the positive scored 0.40 beats one and loses to two. Six strict wins plus half a tie, divided by nine pairs, again gives 13/18. At threshold 0.80, precision, recall, specificity, accuracy and F1 happen to equal 2/3; this coincidence does not make them interchangeable definitions.

Why ROC AUC is not enough for rare positives

ROC rates condition on the actual class. If the positive and negative score distributions are unchanged and only their mixing proportion changes, the population ROC curve and AUC are unchanged. Precision is different: it conditions on a positive prediction. Bayes' rule gives the following relationship.

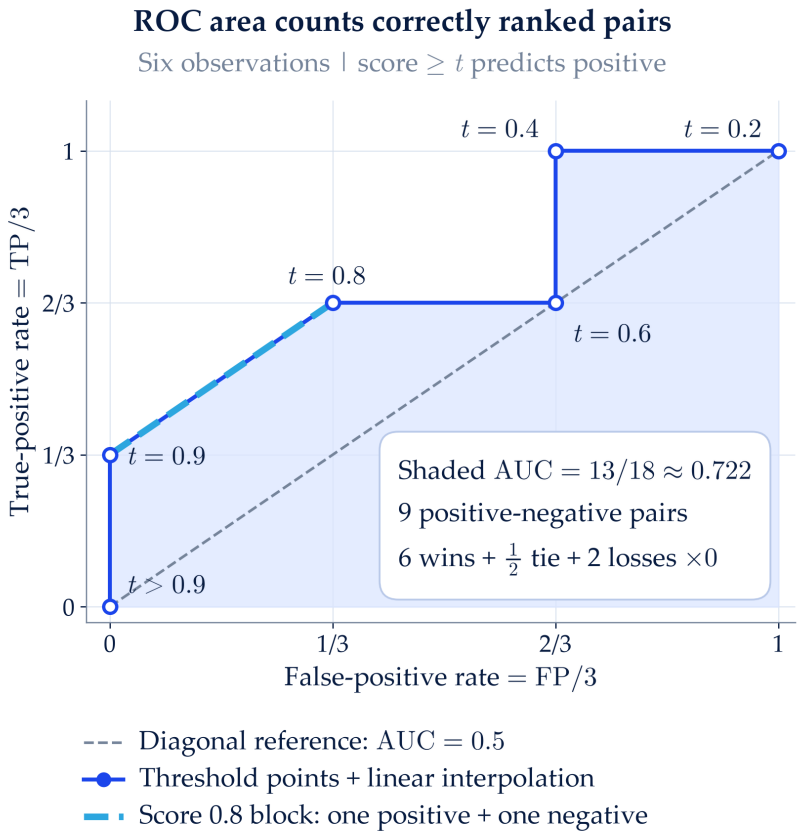


Figure 3.3. Follow threshold markers from low to high false-positive rate; height gives true-positive rate. For scores (0.9, 0.8, 0.8, 0.6, 0.4, 0.2) and labels (1, 0, 1, 0, 1, 0), a score at least t predicts positive. The 0.8 tie enters together; its cyan segment interpolates, not separates, those observations. Shaded trapezoids total 13/18, about 0.722. Equivalently, nine positive-negative pairs give six wins, one half-credited tie and two losses. The diagonal has area 0.5. These exact ranking calculations do not select an operating threshold.

Notation for equation (3.19)

π	Positive-class prevalence in the target population, strictly between zero and one.
$\mathrm{TPR},\ \mathrm{FPR}$	Class-conditional detection and false-alarm rates at the chosen threshold.

precision Positive predictive value, provided the predicted-positive probability is nonzero.

$$\text{precision} = \frac{\pi\,\text{TPR}}{\pi\,\text{TPR} + (1-\pi)\,\text{FPR}} \tag{3.19}$$

Bayes' rule exposes how precision depends on prevalence, even when both class-conditional rates remain fixed.

Keep TPR at 0.90 and FPR at 0.05 while changing prevalence. Precision is 0.947 at 50 percent prevalence but only 0.154 at 1 percent. To see why, use 10,000 cases at the latter prevalence: 90 true positives and 10 false negatives compete with 495 false positives and 9,405 true negatives. The many negative cases create more false than true alerts despite the small FPR. This comparison assumes unchanged class-conditional score behavior; it is not a guarantee when the population changes in other ways.

Precision-recall (PR) curves plot precision against recall across thresholds. They make the positive retrieval trade-off more visible when positives are rare. The no-skill precision reference is the prevalence for an independent random ranking in the population; finite samples fluctuate around it. Average precision summarizes these curves by recall increments, with each tied group treated as one step.

The ROC/PR correspondence in [3.4] fixes the class totals. A change in prevalence can alter precision even at fixed true- and false-positive rates.

Chawla and colleagues [3.6], Section 4, construct SMOTE observations along segments between minority-class neighbors. Interpolation is a modeling assumption, not a new measurement or guaranteed plausible example. Apply it only within training folds. Evaluate on untouched representative observations; oversampling neither repairs invalid neighborhoods nor calibrates probabilities automatically.

Notation for equation (3.20)

AP	Non-interpolated average precision over decreasing distinct score thresholds.
$R_j,\ R_{j-1}$	Recall after and before transition j, with initial recall R0 = 0.
P_j	Precision after transition j, with tied scores included as one block.

$$\mathrm{AP} = \sum_{j=1}^{m} (R_j - R_{j-1}) P_j \tag{3.20}$$

Average precision weights precision by successive recall increments. It is not the trapezoidal area under a precision-recall curve.

P and R denote precision and recall here, not probability and statistical risk. Recall requires actual positives.

For the six-observation example, recall increases at thresholds 0.90, 0.80 and 0.40. The corresponding precisions are 1, 2/3 and 3/5. Their equally sized recall increments give AP = 34/45, approximately 0.756. This differs from ROC AUC because the quantities answer different questions. It also differs from trapezoidal PR area because AP uses precision after each recall increase rather than averaging adjacent endpoint precisions.

Interpretation, uncertainty and a deployable operating point

A final report should include the positive-class definition, prevalence and sample counts, score orientation, ROC AUC, a precision-recall summary, and confusion counts at the selected threshold. ROC AUC is undefined when the evaluation set has only one class. Very small class counts produce coarse curves and unstable estimates. A bootstrap interval must respect the sampling unit and any groups or temporal dependence; treating dependent rows as independent can understate uncertainty. Confidence intervals quantify sampling variability, not arbitrary distribution shift.

Choose models and thresholds on training/validation data, then freeze the full pipeline before reporting a final test. Do not optimize a threshold on the final test and describe its operating-point performance as independent evidence. When only a narrow false-alarm range matters, report sensitivity at the declared FPR constraint or an explicitly defined partial AUC; specify whether any partial-area normalization is used. For multiclass problems, define one-versus-rest or one-versus-one aggregation rather than silently applying a binary formula.

Interactive illustration

The AUC-ROC Evaluation simulator accompanies this topic. Holding its dataset and scores fixed makes the distinction concrete: changing a classification threshold changes confusion counts and the ROC operating point, while the complete ROC curve and its area stay fixed. Access may require sign-in. The link below opens the existing lab; the mathematical example above is fully reproducible without it.

◇ Proposition 3.3: ROC area is pairwise ranking accuracy with half-credit for ties

Notation for this result

K, k, t_k	Distinct-score count, group index 1 through it, and group-k score.
a_k	Actual-positive count in score group k.
n_+	Strictly positive evaluation-positive count.
b_k	Actual-negative count in score group k.
n_-	Strictly positive evaluation-negative count.
A_k	Positive count after admitting k groups; zero before the first.
ℓ	Admitted-group summation index from one through k.
B_k	Negative count after admitting k groups; zero before the first.
u_k	False-positive rate after group k, initially zero.
v_k	True-positive rate after group k, initially zero.
Δ_k	ROC area of the straight segment from the previous point to the point after group k.
$\widehat{\mathrm{AUC}}$	Empirical piecewise-linear ROC area for the evaluation scores.
r, j	Positive/negative observation indices, each ranging over its class count.
s_r^+, s_j^-	Indexed positive/negative scores; superscripts denote classes, not signs.
$\mathbf{1}\{\cdot\}$	Indicator: one for a true enclosed comparison, zero otherwise.
S^+, S^-	Independent scores from positive/negative class-conditional populations under a fixed scoring rule.
$\mathrm{AUC}_{\mathrm{pop}}$	Population ROC area, with the same linear interpolation at score-distribution atoms.
$\Pr$	Probability over the independent class-conditional score pair.

Assumptions. Consider a finite binary evaluation set containing at least one positive and one negative observation. Each observation has one finite real-valued score; larger scores favor the positive class. No probability calibration is assumed. A threshold predicts positive when the score is at least that threshold. Sweep downward through distinct score values, admitting every member of a tied-score group together. Include the no-positive-predictions endpoint, and calculate ROC area by straight-line interpolation between successive operating points.

The empirical area under this ROC curve equals the fraction of positive-negative pairs ranked in the correct order, counting an equal-score pair as one half. This is an exact identity for the observed scores, not a statement that the fitted model has the same performance on new data.

Read the complete supporting proof . *Advanced*

ROC AUC assesses ordering across positive-negative pairs. It does not measure calibration, specify a deployment threshold, or encode an application's error costs. A strictly increasing score transformation preserves every pairwise comparison and therefore preserves AUC. An area of one half describes chance-level pairwise ranking; it does not prove that the model was generated by random guessing.

▶ Example: High accuracy can hide a useless positive-class decision

Among 1,000 hypothetical observations, let 20 be positive. Predicting all negative gives 980 correct decisions, or 98% accuracy, and zero recall. Precision has denominator zero and is undefined unless a reporting convention is declared. A second system finds 15 positives and flags 30 negatives: it misses five positives and correctly rejects 950 negatives. Its recall is 15 divided by 20, or 0.75; precision is 15 divided by 45, or one third; accuracy is 96.5%. Lower accuracy accompanies greater detection. Compare costs and review capacity before choosing; these are illustrative counts, not notebook performance claims.

Interpretation and limits

Read the curve together with a confusion matrix at the intended threshold. AUC alone cannot tell you how many alerts that threshold produces, whether its probabilities are calibrated, or how uncertain a small subgroup estimate is.

Algorithm 3.1. Empirical ROC AUC with tied scores

Source and adaptation: Fawcett [3.3], Sections 5 and 7 and Algorithms 1-2, for the tied-score ROC sweep and trapezoidal area. This normalized all-block formulation admits ties together and checks the final point (1,1). It is an evaluation teaching algorithm, not a model-training or deployment-threshold selection procedure.

Notation and assumptions

$s \in \mathbb{R}^n,\ y \in \{0,1\}^n$ Finite scores and binary labels indexed by i=1,...,n; larger scores favor class one.

$n_+,\ n_-$ Positive and negative observation counts; both must be positive.

$\tau_1 > \cdots > \tau_K, \quad G_k = \{i : s_i = \tau_k\}$

Distinct decreasing scores and their exact-tie blocks; $1 \le K \le n$.

A_k, B_k Cumulative admitted positive and negative counts after block k.

$F_k = B_k/n_-, \quad T_k = A_k/n_+$ False-positive and true-positive rates; T here is not a training budget.

$U,\ Q,\ +\infty$ Running trapezoidal area, ordered threshold/ROC table, and threshold sentinel above all finite scores.

Group sorts decreasing scores and forms exact-equality blocks without consulting labels. Append adds one complete table row. Predictions are positive at scores greater than or equal to the threshold; the initial infinite threshold predicts no positives.

Algorithm 3.1

Input: Finite scores and both binary classes. s, y

Output: Complete threshold/ROC table and area. Q, U

$n_+ \leftarrow \sum_{i=1}^{n} y_i$

$n_- \leftarrow n - n_+$

Require $n_+ > 0, \quad n_- > 0$

Group $(\tau_k, G_k)_{k=1}^{K} \leftarrow \mathrm{Group}(s)$

$(A_0, B_0, F_0, T_0) \leftarrow (0,0,0,0)$

$U \leftarrow 0$

$Q \leftarrow [(+\infty, 0, 0)]$

For $k = 1, \ldots, K$

  $A_k \leftarrow A_{k-1} + \sum_{i \in G_k} y_i$

Algorithm 3.1 (continued)

$B_k \leftarrow B_{k-1} + \sum_{i \in G_k} (1 - y_i)$

Rate $F_k \leftarrow B_k / n_-$

Rate $T_k \leftarrow A_k / n_+$

Accumulate $U \leftarrow U + \frac{1}{2}(F_k - F_{k-1})(T_k + T_{k-1})$

$Q \leftarrow \mathrm{Append}(Q, (\tau_k, F_k, T_k))$

End for

Check $(F_K, T_K) = (1, 1)$

Return Q, U

Why this works. Reject one-class data as undefined for ROC AUC. Vertical segments have zero area. Tied groups move together; straight-line interpolation gives half-credit for positive-negative ties. The finite sweep has at most n blocks after sorting. This area summarizes ranking, not calibration or one operating threshold.

Interactive companion | A threshold is not a ranking metric

Keep the scored dataset fixed and move the decision threshold. Compare the confusion counts, precision, recall, and the operating point on the ROC curve. The selected decisions change, but the area under the full ROC curve should not change unless the scores or labels change. This separates choosing an operating point from evaluating ranking quality.

Open AUC-ROC Evaluation

Sign-in may be required. Suggested comparisons are in the separate teaching companion.

Essential

3.4 Thresholds connect predictions to decisions

Notation for this section

p	Scalar positive-class probability supplied to the rule; exact Bayes optimality requires the decision population's true conditional probability.

A decision rule should reflect the action a system will take. One simple setup assigns cost C_{FP} to a false positive and C_{FN} to a false negative, with zero cost for correct decisions. If p is the actual conditional positive-class probability given the available input in the deployment population and the stated costs fully describe the decision, the expected cost of predicting positive is C_{FP} times one minus p; predicting negative costs C_{FN} times p.

Comparing these expected costs gives a positive decision when p exceeds C_{FP} divided by the sum of the two costs. The familiar one-half threshold follows only when costs are equal under this setup. Capacity limits, different benefits, case-dependent costs and uncertainty about probabilities can require other rules. The formula is an interpretable reference, not permission to assume that a model's probability estimates or an application's costs are known perfectly. Calibration of a score alone supports decisions conditional on that score; it need not recover the optimal decision based on every available input. An estimated probability gives an approximate rule whose realized cost must be evaluated.

In practice, choose or verify a threshold on validation data using a declared criterion. Search only a predeclared range or use a training-based cross-validation procedure; never select it on the final test labels. Report the number of relevant cases and sensitivity to nearby thresholds because a minimum based on very few errors can be unstable. If several thresholds tie, specify a deterministic rule. The selected threshold is part of the complete fitted decision system.

Elkan [3.5] derives the decision-theoretic basis for cost-sensitive classification. The optimal threshold depends on the costs of errors and on probabilities appropriate to the decision population; a default threshold of one half need not minimize the relevant risk.

Notation for equation (3.21)

τ	Threshold mapping a model score or estimated positive-class probability to a label.
$FP(\tau),\ FN(\tau)$	Validation false-positive/false-negative counts after thresholding.
n	Total validation count, including correct and incorrect decisions.
$\mathrm{cost}(\tau)$	Mean validation cost per observation at the candidate threshold.

$$\mathrm{cost}(\tau) = \frac{C_{FP} FP(\tau) + C_{FN} FN(\tau)}{n} \tag{3.21}$$

This empirical mean cost applies the declared penalties to validation decisions at threshold tau. It is used for selection, not presented as an untouched final estimate.

Correct decisions have zero cost in this setup. Selecting a threshold on these data makes this a development estimate.

Notation for equation (3.22)

τ^*	Optimal probability threshold under true conditional probabilities and the stated constant costs. Calibration alone ensures a score-based, not necessarily full-input, optimum.

$$\tau^* = \frac{C_{FP}}{C_{FP} + C_{FN}} \tag{3.22}$$

The threshold follows from comparing expected costs using actual deployment conditional probabilities, constant nonnegative error costs with positive sum, and zero cost for correct decisions. At equality either action has equal cost. Other decision constraints can change the rule.

Correct decisions have zero cost. At equality, either decision has equal expected cost; apply a declared tie rule.

◇ Proposition 3.4: Deriving a cost-sensitive Bayes decision

Notation for this result

X, x, Y, q	Random input, its fixed value, binary target, and actual positive-class probability at that input.
$r_1(x)$	Conditional expected cost of taking the positive action.
$r_0(x)$	Conditional expected cost of taking the negative action.
τ_*	Cost-minimizing probability threshold under the stated constant-cost decision model.

Assumptions. For the exact optimality statement, the probability supplied to the rule is the actual conditional positive-class probability for the decision population. Substituting an estimate gives an approximate decision rule whose quality requires evaluation. Correct decisions have zero cost. False-positive and false-negative costs are constant and nonnegative, with positive sum. There is no capacity constraint coupling decisions across cases.

Predicting positive minimizes conditional expected cost when the positive-class probability exceeds the false-positive cost divided by the sum of the two error costs. At equality either action has identical expected cost.

☐ Proof *Recommended*

At one input, compare actions before averaging over inputs. A positive action risks only the false-positive cost, weighted by the conditional probability of a negative outcome. A negative action risks only the false-negative cost, weighted by the positive-outcome probability. Comparing these two weighted costs produces the threshold; comparing the costs alone would ignore how likely each error is.

$$q = \Pr(Y = 1 \mid X = x) \tag{3.T29}$$

$$r_1(x) = C_{FP}(1 - q) \tag{3.T30}$$

$$r_0(x) = C_{FN}\, q \tag{3.T31}$$

$$r_1(x) < r_0(x) \iff C_{FP} < (C_{FP} + C_{FN})q \tag{3.T32}$$

$$q > \tau_* \tag{3.T33}$$

$$\tau_* = \frac{C_{FP}}{C_{FP} + C_{FN}} \tag{3.T34}$$

Because the cost denominator is positive, the comparison gives the stated probability threshold without reversing the inequality. Choosing the lower expected cost separately at every input also minimizes their population average by the law of total expectation. That final step fails for coupled decisions: a fixed total alert budget links cases, so independent pointwise choices need not solve the constrained problem.

If a missed positive costs four times a false alarm, the threshold is one fifth. Its justification needs probabilities, not arbitrary ranking scores. Broad-bin agreement is insufficient. Exact score calibration requires the positive-class probability conditional on the reported score to equal that score almost surely. Under the same cost assumptions, the threshold is then optimal among rules using that score; a rule using additional informative features may do better.

Elkan [3.5], Sections 1.1-1.3, defines costs with predicted classes as rows and actual classes as columns, then compares conditional expected costs. Reversing those axes reverses the meaning of a false-positive cost. Before selecting a threshold, label every cell and check the cost of one deliberately wrong prediction by hand. The probabilities must describe the target population: a

ranking score or a score fitted after resampling is not automatically a calibrated probability.

When costs vary between cases, repeat the conditional comparison with those case-specific costs. When actions are coupled by a budget, solve the constrained decision problem. Empirically selecting a threshold on validation data is a development operation, so its selected cost must not be presented as an independent final evaluation.

□

A one-half threshold is justified by equal error costs in this particular setup, not by the name of the classifier. Probability estimation error, changing prevalence, and uncertain costs can all move realized performance away from this ideal rule.

▶ Example: A cost ratio suggests, but does not certify, a threshold

Let a false positive cost one unit and a false negative cost five. Under the stated assumptions, divide the false-positive cost by the sum: one sixth, approximately 0.167. At probability 0.2, the positive action costs one times 0.8 in expectation; the negative action costs five times 0.2, or one. Choose positive. An uncalibrated model may need an empirically selected threshold instead. The notebook selects from a declared grid using validation labels, freezes the choice, then evaluates on test data. A limit on review capacity couples decisions and requires an additional constraint; the two-cost threshold alone does not encode it.

Interpretation and limits

Lowering a threshold often increases recall but does not guarantee acceptable precision, cost or workload. Threshold selection on test labels invalidates the intended final evaluation.

Algorithm 3.2. Validation-only threshold selection

Source and adaptation: Fawcett [3.3] for the distinct-score threshold sweep, and Elkan [3.5] for conditional error-cost reasoning. This book adds explicit validation-only selection, immutable count records and a highest-threshold tie rule; empirical validation minimization is not an independent test claim.

Notation and assumptions

$s \in \mathbb{R}^n,\ y \in \{0,1\}^n,\ n > 0$	Finite validation scores and binary labels; larger scores favor class one.

$C_{FP}, C_{FN} \geq 0, \quad C_{FP} + C_{FN} > 0$

False-positive and false-negative costs in one common unit.

$\tau_0 = +\infty > \tau_1 > \cdots > \tau_K$ All-negative sentinel followed by distinct decreasing validation scores.

$G_k = \{i : s_i = \tau_k\}, \quad a_k, b_k$ Tie-block indices and its positive/negative counts; n_+, n_- are total class counts.

$c = (TP, FP, FN, TN)$ Current true-positive, false-positive, false-negative and true-negative counts; c_k is their stored copy at threshold τ_k.

$q(c) = \dfrac{C_{FP}FP + C_{FN}FN}{n}$ Mean empirical cost per validation observation; q_k stores this cost at threshold τ_k.

$Q,\ k^*, \tau^*, \hat{f}$ Ordered threshold/count/cost table, selected row, selected threshold, and model fitted before using these validation labels.

Predict positive exactly when score is at least the threshold. Group uses exact score ties without label-based ordering. Append stores a threshold, its counts and cost. Decreasing threshold order makes the smallest minimizing row index the highest-threshold tie rule.

Algorithm 3.2

Input: Frozen model and its validation scores/labels. $\hat{f}, s, y$

Nonnegative error costs with positive sum. C_{FP}, C_{FN}

Output: Selected threshold, highest-threshold tie rule and complete selection table.

Selected validation confusion counts; no independent test-performance claim.

Group $(\tau_k, G_k)_{k=1}^{K} \leftarrow \mathrm{Group}(s)$

$n_+ \leftarrow \sum_{i=1}^{n} y_i$

$n_- \leftarrow n - n_+$

$c \leftarrow (0, 0, n_+, n_-)$

$Q \leftarrow [(\tau_0, c, q(c))]$

For $k = 1, \ldots, K$

  $a_k \leftarrow \sum_{i \in G_k} y_i$

  $b_k \leftarrow |G_k| - a_k$

  $c \leftarrow c + (a_k, b_k, -a_k, -b_k)$

Algorithm 3.2 (continued)

$Q \leftarrow \mathrm{Append}(Q, (\tau_k, c, q(c)))$

End for

$k^* \leftarrow \min \arg \min_{0 \le k \le K} q_k$

$\tau^* \leftarrow \tau_{k^*}$

Freeze Preserve model parameters and the score-equality rule. $(\hat{f}, \tau^*)$

Return Include the declared highest-threshold tie rule. τ^*, Q, c_{k^*}

Why this works. The table stores copies of the counts, not aliases to mutable c. Every distinct deterministic decision is represented, including all-negative and all-positive. Sorting costs O(n log n); the sweep costs O(n). After all development choices are frozen, evaluate once on a separate final population. Empirical cost minimization is not an exact Bayes-optimality claim without true conditional probabilities.

Essential

3.5 Handle imbalance without rewriting the evaluation population

Start an imbalanced task with counts, not a balancing operation. Unequal target frequencies may correctly represent the application. Report prevalence, establish a majority or class-frequency baseline, and choose metrics exposing the relevant errors. One percent positives among a million examples is not the same estimation problem as one percent among a hundred. Label quality and subgroup coverage may matter more than the overall ratio.

Weighting and resampling change fitting, not just the displayed class balance. Weights alter observations' loss contributions; oversampling repeats or generates minority examples, while undersampling removes majority examples. A relevant decision metric may improve, but variance, information loss and probability meaning can also change. In particular, weighted logistic fitting targets altered class contributions. Its raw outputs need not be probabilities for the original population prevalence.

Apply resampling only within the training part of each evaluation split. Duplicating examples before splitting can place related copies on both sides of the boundary and create misleading validation success. Synthetic resampling can leak information in a similar way if it uses evaluation neighbors. Leave validation and test prevalence representative of the intended use unless a separate sampling design and correction are explicitly justified. Compare

weighting, resampling and threshold adjustment as distinct interventions, documenting which combination was selected.

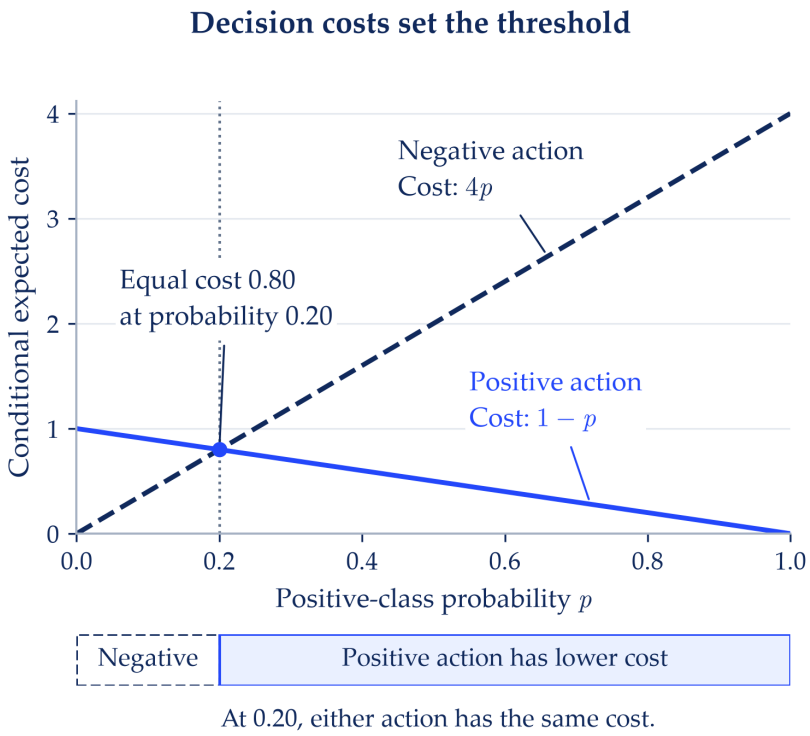


Figure 3.4. A positive action incurs false-positive cost 1; a negative action incurs false-negative cost 4; correct decisions cost zero. Conditional expected costs cross at positive-class probability 0.20. The threshold is optimal under these stated probabilities and constant costs, not under every deployment objective.

Chawla and colleagues [3.6] introduced the synthetic minority over-sampling technique (SMOTE) as a training-data intervention. The distinction from evaluation is essential: synthetic observations may alter how a model is fitted, but the held-out assessment must still represent the population in which the model will be used. Saerens, Latinne and Decaestecker derive a probability-adjustment procedure for changed class proportions when the input distribution within each class remains unchanged [3.9]. That restricted prior-probability-shift model explains why resampled-training probabilities

may need correction, but does not license the same correction when class-conditional feature distributions also change.

Notation for equation (3.23)

J_{weighted}	Weighted mean binary cross-entropy using the displayed sample-count normalization and no explicit regularization term.
$y_i,\ p_i$	Observed zero-or-one target and modeled positive-class probability for training observation i.
a_{y_i}	Strictly positive class weight chosen by the observation's target: one weight for each of the two classes.
$\log$	Natural logarithm used to evaluate each observation's binary cross-entropy contribution.

$$J_{\text{weighted}} = -\frac{1}{n}\sum_i a_{y_i}[y_i \log p_i + (1-y_i)\log(1-p_i)] \tag{3.23}$$

Positive class-specific weights a alter the mean-loss objective. Some implementations normalize weights differently, so inspect their convention and any interaction with regularization.

Class weighting changes the fitting objective. Raw fitted probabilities need not retain the original population's prevalence interpretation.

◇ Proposition 3.5: Class weighting changes the probability being fitted

Notation for this result

q, p	Actual conditional positive-class probability and the proposed fitted probability at a fixed input.
a_1, a_0	Strictly positive weights assigned to positive and negative outcomes, respectively.
r_w	Conditional expected class-weighted binary log loss; the subscript denotes weighting, not a coefficient-vector argument.
p_w	Unique minimizer of conditional weighted log loss for the actual probability and supplied class weights.
π	Positive-class prevalence in the evaluation population, not the circle constant in this formula.
t, f	True-positive and false-positive rates of a fixed decision rule conditional on their respective actual classes.
PPV	Positive predictive value, also called precision; defined when predicted positives have positive probability.

Assumptions. Consider unrestricted minimization of expected weighted binary log loss at one fixed input. Both class weights are positive, and the actual positive probability is strictly between zero and one. The exact probability correction below concerns this population optimum. Finite samples, regularization, misspecification, and optimization error can prevent a fitted model from attaining it.

The minimizer of class-weighted log loss has odds multiplied by the positive-to-negative weight ratio. Thus its raw output does not generally estimate the original population probability.

☐ **Proof** *Recommended*

$$r_w(p; q) = -a_1\, q \log p - a_0 (1 - q) \log(1 - p) \tag{3.T35}$$

Differentiate with respect to the proposed probability and set the result to zero. Multiplying by the positive product of the probability and its complement yields a linear equation in the proposed probability.

$$-\frac{a_1\, q}{p} + \frac{a_0 (1 - q)}{1 - p} = 0 \tag{3.T36}$$

$$p_w = \frac{a_1\, q}{a_1\, q + a_0 (1 - q)} \tag{3.T37}$$

$$\frac{p_w}{1 - p_w} = \frac{a_1}{a_0} \frac{q}{1 - q} \tag{3.T38}$$

The positive second-derivative terms show that the stationary probability is the unique minimum. But it minimizes the weighted objective, whose odds differ from the original population odds. Inverting the displayed odds relationship recovers the population probability in this idealized calculation. Applying that inverse to a finite fitted model is not, by itself, evidence of calibration.

$$q = \frac{a_0 p_w}{a_1 (1 - p_w) + a_0 p_w} \tag{3.T39}$$

A related prevalence effect appears in precision even if the class-conditional detector behavior stays fixed. Apply Bayes' rule to the event that the decision is positive; the denominator includes positives detected and negatives falsely flagged.

$$\mathrm{PPV} = \frac{\pi t}{\pi t + (1-\pi) f} \tag{3.T40}$$

This identity holds within each population. Precision can change with prevalence even when the rates stay fixed. Using the old rates to compute precision in a new population requires both rates to remain stable.

□

Weighting changes the fitting target; calibration checks probability forecasts; thresholding chooses actions. None automatically repairs the others. Resample only within fitting folds and keep evaluation prevalence unchanged. Check probability corrections on representative independent data: finite sampling, regularization, model restriction and imperfect optimization can all break exact agreement with the population identity.

▶ Example: Oversample training, not the entire table

Start with 900 negative and 100 positive development observations. Assign permanent original-row identifiers, then split before copying any minority rows. Otherwise two copies of one observation can cross the boundary while their new row positions appear disjoint. Resample identifiers only from the training minority subset; verify that every selected identifier belongs to training and none belongs to validation or test. Evaluate on the unchanged validation distribution, as the notebook does. A balanced fitting sample is a training choice, not evidence of better cost or calibrated probabilities. Original identities, rather than current row positions, make this leakage check meaningful.

Interpretation and limits

Balancing the validation or test set changes the question and can make precision misleading for deployment. Preserve the target population or explicitly account for the altered sampling design.

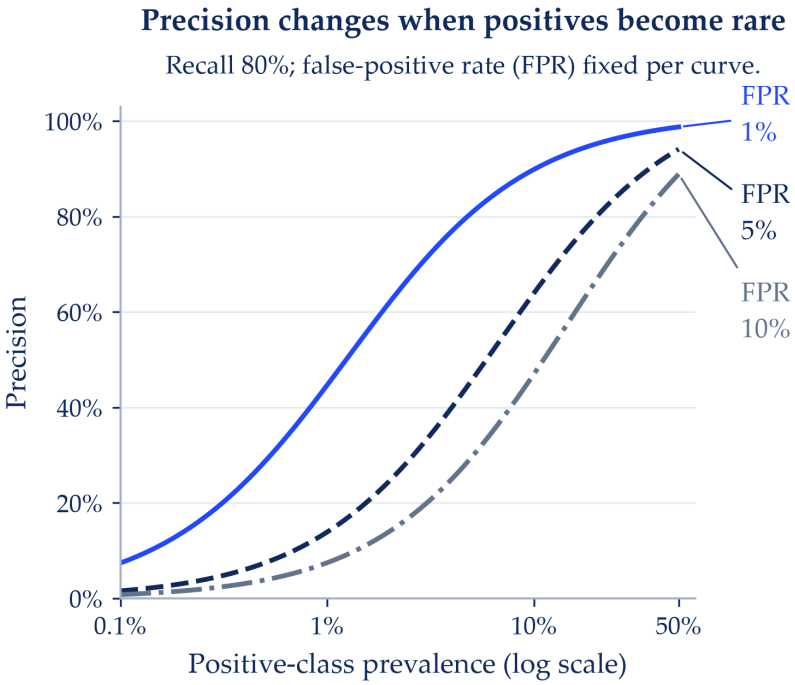


Figure 3.5. Bayes-rule precision as prevalence varies, with recall fixed at 0.80 and the indicated false-positive rate fixed for each curve. Precision decreases sharply for rare positives. Transporting these curves requires stable class-conditional detection and false-alarm rates.

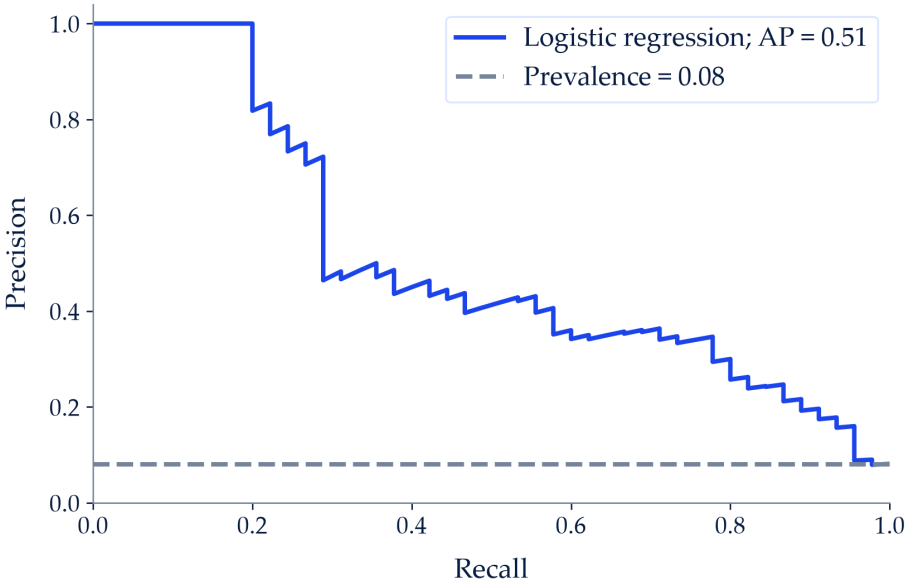


Figure 3.6. Actual output from a fixed synthetic, imbalanced classification experiment. Precision and recall change with the threshold. Average precision summarizes ranking but does not choose a threshold or encode the cost of a missed event.

Table 3.5. Classification evaluation answers different questions

Quantity	Question answered	What it does not establish
Log loss / Brier	Log loss averages logarithmic penalties; the Brier score averages squared probability errors.	A single aggregate cannot identify which subpopulation is miscalibrated.
ROC AUC	Do positive cases rank above negative cases?	It does not choose a deployment threshold or prove probability calibration.

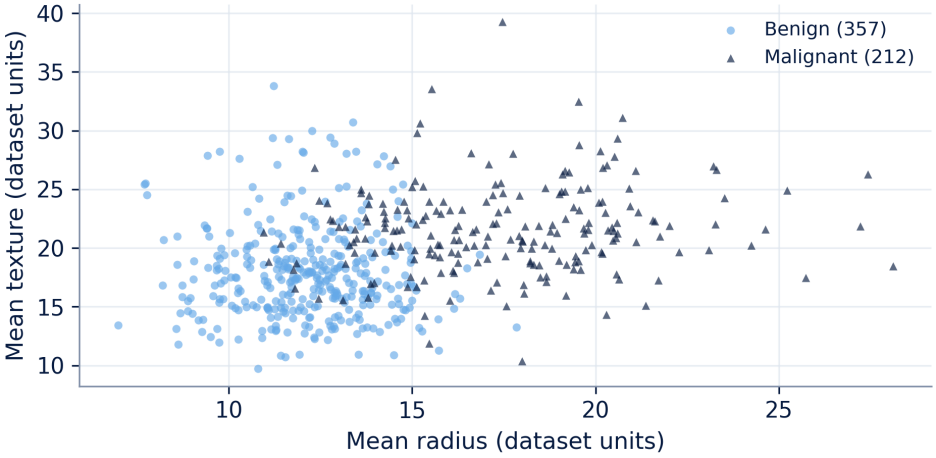
Quantity	Question answered	What it does not establish
Precision / recall	What happens at a stated operating threshold?	Prevalence can change precision. Other shifts can affect both.
PR / average precision	How does retrieval precision trade against recall?	Comparisons need the same positive class and evaluation population.

Table 3.6. Chapter synthesis

Concept	Main conclusion	Engineering check
Logistic model	An affine log-odds model produces a probability.	Separate the score, sigmoid output and final class decision.
Cross-entropy	The Bernoulli likelihood gives a proper probability loss.	Use a stable logits implementation, not a log of rounded probabilities.
Metrics	Every confusion-matrix ratio has a specific denominator.	Report support counts, positive-class definition and zero-denominator handling.
Imbalance	The evaluation population should reflect the stated use.	Resample only within fitting; select thresholds on development evidence.

Wisconsin Diagnostic Breast Cancer (WDBC)

569	30	2
samples	measurements	diagnoses



Two image-derived measurements from all 569 records. These are feature values, not the original microscopic images.

WDBC contains 569 breast-mass samples described by 30 numerical summaries of cell nuclei in digitized fine-needle-aspirate images. Its labels are benign or malignant. These are image-derived measurements, not raw mammograms; retrospective classification does not establish clinical safety or screening-population performance. [3.10]

Here: malignant is the positive label, unlike the original scikit-learn encoding. Later cases and laboratories use different input subsets and disjoint partitions. These are nucleus measurements, not raw mammograms.

Wolberg and colleagues / UCI, CC BY 4.0. Recorded labels shown descriptively. Source / terms.

Case Study: WDBC: separate ranking, probability quality and decision cost

Problem and data

The Wisconsin Diagnostic Breast Cancer collection supplies thirty measurements for 569 historical records. Audit a classifier with malignant as the positive class, explicitly reversing the library's original target convention. Separate three questions: whether scores rank cases, whether probability estimates match outcomes, and whether a threshold produces acceptable errors under a declared cost model. Costs here are instructional assumptions, not clinical valuations; the dataset and experiment cannot validate a screening system or support treatment decisions.

Experimental design

Create stratified training, validation and final-test partitions. Fit ordinary and class-weighted logistic pipelines using training-only standardization. Compare their validation ranking and log-loss diagnostics, then evaluate four predeclared thresholds under a false-negative cost of five and false-positive cost of one. Select the minimum-cost model-threshold pair with a deterministic tie rule and retain its original training fit. Evaluate the frozen policy, its half-threshold comparator and the training-prevalence reference once on the final sample.

Measured Reference Run

Malignant was recoded as positive. The stratified split contained 341 fitting, 114 validation and 114 final records. Ordinary and balanced-weight logistic pipelines were compared at four thresholds: 1/6, 0.3, 0.5 and 0.7. Validation minimized an illustrative cost of five per false negative and one per false positive, selecting ordinary logistic at 1/6 without refitting.

Table 3.7. Measured case-study evidence

Final policy	False negatives	False positives	Mean cost
Selected threshold 1/6	0	6	0.053
Same model, threshold 0.5	3	2	0.149
Prevalence rule at 0.5	42	0	1.842

Lowering the threshold changed decisions, not fitted scores: more false positives bought fewer false negatives and lower declared cost. Final ROC AUC 0.998 and log loss 0.065 describe the ordinary model's scores, not its operating point.

Failure Analysis And Limits

The high-AUC model still missed three malignant records at threshold 0.5. Ranking quality therefore did not eliminate operating-point mistakes. The selected threshold produced six false positives, and zero observed false negatives does not guarantee zero future misses. These teaching costs are not medical utilities; the historical benchmark does not establish screening safety.

Historical saved run with seed 2027. Preserve the positive-class recoding, equality-as-positive rule and frozen original training fit; do not infer clinical performance from this table.

▶ Example: inspecting the real data

The library's target 0 means malignant and becomes positive label 1 here. An always-benign rule correctly labels 357 records but misses all 212 malignant records: accuracy 357/569, about 0.627, yet malignant recall 0/212 = 0. This full-collection arithmetic exposes the majority-class trap; it is not a final-test result.

Dataset source and documentation

Companion: Week 03, Real-Data Case Study notebook. Reproduce the declared run, then investigate a changed condition. No fixed benchmark score is promised.

What Zero Observed Misses Can Still Hide

Zero observed false negatives is a count, not proof of zero future miss probability. To quantify the distinction, consider a fixed classifier evaluated on 42 independent positive-class observations from one unchanged population, with no misses. This binomial calculation illustrates the scale of uncertainty; it does not validate a medical application or repair dependence, selection bias, or population mismatch.

p, n_+, M Population miss probability for the fixed classifier within the positive class; positive test count; number of observed misses. The outcomes are assumed independent with the same p.

$$\Pr(M = 0 \mid p) = (1 - p)^{n_+} \tag{3.B1}$$

For a one-sided confidence bound, find the miss probability under which an all-zero count is just a five-percent event. Inverting the binomial tail gives the exact upper endpoint for this observed count.

α, p_{upper} Chosen tail probability, here 0.05, and the one-sided 95-percent binomial upper confidence endpoint after zero misses.

$$p_{\text{upper}} = 1 - \alpha^{1/n_+} \tag{3.B2}$$

For 42 positives, the upper endpoint is approximately 0.069, or 6.884 percent. Thus a perfect observed recall does not tightly establish a negligible population miss rate. The confidence interpretation concerns the coverage of this procedure under repeated samples satisfying its assumptions; it is not a posterior probability assigned to this particular interval. Choosing a decision threshold after inspecting these same outcomes would invalidate the fixed-rule setup. Larger representative positive samples and an appropriate sampling design matter more than extra displayed decimal places.

A Controlled Learning Cycle

The WDBC case distinguishes ranking, probability quality and decision cost under an explicit positive-class convention. Week 3 notebooks audit stable losses, prevalence changes and validation-selected thresholds. The single-feature laboratory exposes how fixed scores become confusion counts and hypothetical costs. Record one controlled comparison with its data roles and explanation; retain the nonclinical limits below rather than treating a favorable count as a safety claim.

Matched practice and simulator protocol · Packaged edition; launch instructions in the guide.

Laboratory 03 | Classification And Imbalanced Data

Logistic scores, thresholds, confusion counts and hypothetical costs.

A fitted logistic model separates probability scores from threshold-dependent decisions on historical breast-cancer observations.

Open Laboratory 03 | Use the live view for readable controls.

Transfer: Thirty-feature WDBC classification → Single-feature WDBC classification. Not clinical validation; no deployed prevalence or treatment-effect claim.

Chapter Capability

Select a threshold with development data and explain a confusion matrix, a ranking curve, and a probability score without treating them as interchangeable.

Week 03: separate practice, solutions and assessment

References

[3.1] Cox, D. R. (1958). The Regression Analysis of Binary Sequences. Journal of the Royal Statistical Society: Series B 20(2), 215-232. Source / DOI

[3.2] Gneiting, T.; Raftery, A. E. (2007). Strictly Proper Scoring Rules, Prediction, and Estimation. Journal of the American Statistical Association 102(477), 359-378. Source / DOI

[3.3] Fawcett, T. (2006). An introduction to ROC analysis. Pattern Recognition Letters 27(8), 861-874. Source / DOI

[3.4] Davis, J.; Goadrich, M. (2006). The Relationship Between Precision-Recall and ROC Curves. Proceedings of the 23rd International Conference on Machine Learning. Source / DOI

[3.5] Elkan, C. (2001). The Foundations of Cost-Sensitive Learning. Proceedings of the 17th International Joint Conference on Artificial Intelligence. Source / DOI

[3.6] Chawla, N. V.; Bowyer, K. W.; Hall, L. O.; Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique. Journal of Artificial Intelligence Research 16, 321-357. Source / DOI

[3.7] Brier, G. W. (1950). Verification of Forecasts Expressed in Terms of Probability. Monthly Weather Review 78(1), 1-3. Source / DOI

[3.8] Saito, T.; Rehmsmeier, M. (2015). The Precision-Recall Plot Is More Informative than the ROC Plot When Evaluating Binary Classifiers on Imbalanced Datasets. PLOS ONE 10(3), e0118432. Source / DOI

[3.9] Saerens, M.; Latinne, P.; Decaestecker, C. (2002). Adjusting the Outputs of a Classifier to New a Priori Probabilities: A Simple Procedure. Neural Computation 14(1), 21-41. Source / DOI

[3.10] Wolberg, W.; Mangasarian, O.; Street, N.; Street, W. (1993). Breast Cancer Wisconsin (Diagnostic). UCI Machine Learning Repository. Source / DOI

[3.11] Manning, C. D.; Raghavan, P.; Schütze, H. (2008). Introduction to Information Retrieval. Cambridge University Press. Source / DOI

Advanced

ROC Area Is Pairwise Ranking Accuracy With Half-Credit For Ties

Supporting proof for Section 3.3.

Notation For This Result

K, k, t_k	Number of distinct scores, group index from one through that number, and the score of group k.
a_k	Number of actual positive observations whose score equals the score of group k.
n_+	Strictly positive count of actual positive observations in the evaluation set.
b_k	Number of actual negative observations whose score equals the score of group k.
n_-	Strictly positive count of actual negative observations in the evaluation set.
A_k	Cumulative positive count after admitting the first k score groups; the count is zero before the first group.
ℓ	Summation index over score groups already admitted, running from one through k.
B_k	Cumulative negative count after admitting the first k score groups; the count is zero before the first group.
u_k	False-positive rate after admitting group k; its initial value before the first group is zero.
v_k	True-positive rate after admitting group k; its initial value before the first group is zero.
Δ_k	ROC area contributed by the straight-line segment from the preceding operating point to the point after group k.
$\widehat{\mathrm{AUC}}$	Empirical area under the piecewise-linear ROC curve formed from the observed evaluation scores.
r, j	Indices over positive and negative observations, respectively; their ranges are their corresponding class counts.

s_r^+, s_j^-	Scores of the indexed actual positive and actual negative observations; superscripts denote classes, not arithmetic signs.
$\mathbf{1}\{\cdot\}$	Indicator equal to one when the enclosed comparison is true and zero otherwise.
S^+, S^-	Independent scores from the positive and negative class-conditional populations under one fixed scoring rule.
$\mathrm{AUC}_{\mathrm{pop}}$	Population ROC area using the same linear interpolation convention at score-distribution atoms.
$\Pr$	Probability over the specified pair of independent class-conditional score draws.

Assumptions. Consider a finite binary evaluation set containing at least one positive and one negative observation. Each observation has one finite real-valued score; larger scores favor the positive class. No probability calibration is assumed. A threshold predicts positive when the score is at least that threshold. Sweep downward through distinct score values, admitting every member of a tied-score group together. Include the no-positive-predictions endpoint, and calculate ROC area by straight-line interpolation between successive operating points.

The empirical area under this ROC curve equals the fraction of positive-negative pairs ranked in the correct order, counting an equal-score pair as one half. This is an exact identity for the observed scores, not a statement that the fitted model has the same performance on new data.

□ Proof *Advanced*

Group observations by distinct score, from highest to lowest. Within a group, every observation crosses the threshold simultaneously. Record positive and negative group counts, and then accumulate them to obtain the ROC coordinates.

$$t_1 > t_2 > \cdots > t_K \tag{3.T17}$$

$$n_+ = \sum_{k=1}^{K} a_k \tag{3.T18}$$

$$n_- = \sum_{k=1}^{K} b_k \tag{3.T19}$$

$$A_k = \sum_{\ell=1}^{k} a_\ell \tag{3.T20}$$

$$B_k = \sum_{\ell=1}^{k} b_\ell \tag{3.T21}$$

Before the first score group is admitted, both cumulative counts are zero. Divide each cumulative count by its corresponding class total to obtain the operating point.

$$u_k = \frac{B_k}{n_-} \tag{3.T22}$$

$$v_k = \frac{A_k}{n_+} \tag{3.T23}$$

The initial point predicts every observation negative and has zero false-positive and zero true-positive rate.

A score group increases the horizontal coordinate by its negative count divided by the total negative count. The mean of the two vertical endpoint coordinates includes every higher-scoring positive and half of the positives in the current group. The trapezoidal contribution is therefore the following exact quantity. A group containing no negatives has zero horizontal width.

$$\Delta_k = (u_k - u_{k-1}) \frac{v_{k-1} + v_k}{2} \tag{3.T24}$$

$$\Delta_k = \frac{b_k A_{k-1} + \frac{1}{2} a_k b_k}{n_+ n_-} \tag{3.T25}$$

For each negative observation in the current group, every positive in an earlier group has a strictly larger score. Thus the first numerator term counts exactly those correctly ordered pairs. The second term gives half-credit to each positive-negative pair tied in the current group. Every pair belongs to exactly one such comparison when its negative observation's group is processed. Lower-scoring positives contribute nothing. Summing over all groups proves the identity.

$$\widehat{\mathrm{AUC}} = \sum_{k=1}^{K} \Delta_k \tag{3.T26}$$

$$\widehat{\mathrm{AUC}} = \frac{1}{n_+ n_-} \sum_{r=1}^{n_+} \sum_{j=1}^{n_-} \left[\mathbf{1}\{s_r^+ > s_j^-\} + \frac{1}{2}\mathbf{1}\{s_r^+ = s_j^-\} \right] \tag{3.T27}$$

Equivalently, draw one positive and one negative independently and uniformly from their observed classes. The empirical area is the probability of a strict ranking win plus one half of the probability of a tie. For a fixed scoring rule evaluated on independent fresh class-conditional draws, the corresponding population quantity is the same expression with population scores. The empirical identity itself does not require independence between rows; such sampling assumptions matter when making population claims or calculating uncertainty.

$$\mathrm{AUC}_{\mathrm{pop}} = \Pr(S^+ > S^-) + \frac{1}{2}\Pr(S^+ = S^-) \tag{3.T28}$$

A straight segment across a tied group describes interpolation, not additional deterministic thresholds that separate equal scores. It can be interpreted as the expected operating rates of a rule that admits the tied group with a chosen probability. Ordering tied observations by their true labels instead would invent ranking information and change the area.

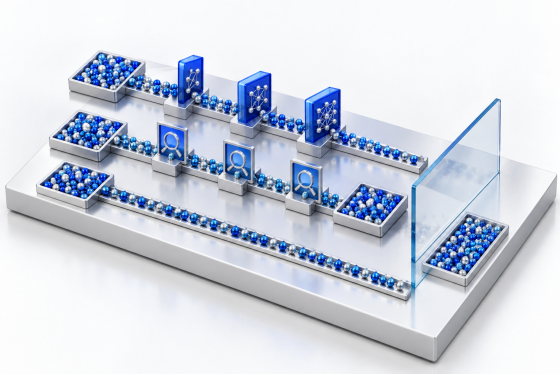
□

ROC AUC assesses ordering across positive-negative pairs. It does not measure calibration, specify a deployment threshold, or encode an application's error costs. A strictly increasing score transformation preserves every pairwise comparison and therefore preserves AUC. An area of one half describes chance-level pairwise ranking; it does not prove that the model was generated by random guessing.

Return to the chapter and its references

CHAPTER 04

Validation And Model Selection



Reading path

Start with Essential. The other levels offer more depth.

Essential
4.1 4.2 4.3 4.5

Recommended
4.4

Advanced
Proofs 4.2 4.3 4.4

Worked example Lab task

The same algorithm can look excellent or disappointing depending on which observations are withheld. Design the split to represent the intended prediction problem, not to maximize the reported score.

What You Will Be Able To Do

- Choose holdout, stratified, grouped or temporal evaluation to match a prediction contract.

- Place preprocessing and every data-dependent selection step inside the correct training boundary.

- Implement nested cross-validation that evaluates a tuning procedure rather than its best development score.

- Report final held-out results with reproducibility information and appropriately limited uncertainty claims.

Before you begin: Chapters 1-3: learning objectives and metrics; Scikit-learn estimators and pipelines; Indexing arrays and interpreting sample variability.

Draw each information boundary, then inspect the case; return for selection-optimism and uncertainty arguments.

Algorithm scope: Complete evaluation orchestration: Algorithm 4.1 specifies nested data roles and pipeline selection. Fit_h and Refit delegate estimator implementations while preserving preprocessing, stopping and calibration boundaries.

Chapter notation

n	Positive observation count; development and bootstrap-evaluation roles are specified locally.
$K,\ V$	Fold count: K in the conventional name, V in the equations; an integer at least two.

A learning experiment includes every choice that can adapt to observed results: features, preprocessing, algorithms, regularization, thresholds and stopping rules. Credible evaluation accounts for that whole sequence. An inappropriate split can make a carefully tuned model less informative than a simple baseline evaluated under the intended conditions of use.

This chapter makes model development inspectable and repeatable. It compares cross-validation designs, demonstrates apparent signal from random-label leakage, and separates inner tuning from outer evaluation. No split is universally correct: each reported score must identify its population, the learning procedure it evaluates, and the uncertainty that remains.

Essential

4.1 Holdout, K-fold, stratification and leave-one-out

A holdout split asks one direct question: how does a model fitted on one set perform on reserved observations? This is often adequate if those observations are representative and informative enough. Results still depend on the split, especially when few minority-class positives remain. Choose its size from data availability, dependence, decision stakes and required precision; fixed percentages are examples, not statistical laws.

K-fold cross-validation partitions development observations into a chosen number of folds, denoted V in our equations. Each observation is evaluated by a model fitted without its fold, and the fold scores summarize that procedure's performance. The fitted models generally use less data than a final model trained on all development observations. Stratification approximately preserves class proportions across folds but does not remove grouping, temporal dependence or selection bias.

Leave-one-out fits n models to n development examples, withholding one example each time. Training sets overlap heavily, so results are dependent; one binary example cannot define a fold ROC AUC. Computation can be expensive, and smaller holdouts do not make estimates universally unbiased or superior. Match the estimator, metric, data size and dependence structure to the intended deployment.

Stone [4.1] established cross-validation as a framework for choosing and assessing statistical predictors. Kohavi's empirical comparison [4.2] illustrates

how fold construction changes estimation and selection behavior; its findings are useful evidence, not a universal prescription for one fold count.

Notation for equation (4.1)

D	Cross-validation development data, excluding any separately reserved final test set.
$V,\ v$	Fold count (at least two) and index from one through V.
D_v	Held-out fold-v records, used only for evaluation in that fit.
$D \setminus D_v$	Training records: development data minus the held-out fold.
$\hat{f}_{D \setminus D_v}$	Complete predictor and preprocessing fitted only to the fold's training records.
$\widehat{R}_{D_v}$	Held-out-fold empirical risk using its targets and independently fitted predictor.
$\widehat{R}_{\mathrm{CV}}$	Arithmetic mean of V held-out risks for the specified fitting procedure.

$$\widehat{R}_{\mathrm{CV}} = \frac{1}{V} \sum_{v=1}^{V} \widehat{R}_{D_v}\left(\hat{f}_{D \setminus D_v}\right) \tag{4.1}$$

Pooling mean losses gives each observation equal weight, not each fold. With fold sizes two and eight and mean losses zero and one, the unweighted fold average is 0.5, whereas the pooled loss is 0.8. Equal fold weights are appropriate when sizes are equal. This identity relies on additive per-observation loss: a nonlinear metric computed from pooled predictions need not equal even the size-weighted fold metrics.

Distinguish fold averaging from pooling predictions. Pooled AUC also compares scores produced by different fitted models, whose scales need not match; it is not simply mean fold AUC. With unequal fold sizes, additionally state whether folds or observations receive equal weight.

◇ Theorem 4.1: What an independent holdout estimates

The bounded-loss concentration step is Hoeffding's inequality [4.12]; the conditional expectation, variance and finite-family union-bound arguments are developed below.

Notation for this result

f, L	Complete frozen predictor and declared per-observation loss.
$m, i, (X_i,\ Y_i)$	Evaluation count, index 1 through m, and independent target-population input-target pair.
$Z_i, R(f), \widehat{R}$	Individual loss, conditional population risk of the fixed predictor, and holdout mean loss.

Z	Generic evaluation loss conditional on the frozen predictor, between zero and one.
s, g	Real exponential-tilt parameter and log centered-loss moment-generating function.
$\mathbb{E}_s, h$	Expectation under exponential tilting and a bounded real loss test function.
ϵ	Positive holdout-versus-population-risk tolerance.
$M, j, f_j, \widehat{R}_j$	Positive predeclared predictor count, index 1 through M, predictor, and common-holdout risk estimate.

Assumptions. Condition on the complete fitted predictor, including preprocessing, hyperparameters, and any independent training randomness. Evaluation pairs are independent draws from the named population and are independent of this fitted predictor. Finite first loss moments are required for unbiasedness, and finite second moments for the variance identity. The exponential bound additionally assumes each individual loss lies between zero and one; it does not directly cover unbounded squared error.

Condition on the information that produced the fitted predictor, then regard that predictor as fixed. Independent holdout observations estimate its population risk without conditional bias; bounded losses additionally give a finite-sample deviation bound. If holdout results helped choose the predictor, this fixed-predictor argument no longer applies to the selected score. Different row identifiers alone do not establish independence.

☐ **Proof** *Recommended*

$$Z_i = L(Y_i, f(X_i)) \tag{4.T1}$$

$$R(f) = \mathbb{E}[Z_i \mid f] \tag{4.T2}$$

$$\widehat{R} = \frac{1}{m} \sum_{i=1}^{m} Z_i \tag{4.T3}$$

$$\mathbb{E}[\widehat{R} \mid f] = R(f) \tag{4.T4}$$

$$\mathrm{Var}(\widehat{R} \mid f) = \frac{\mathrm{Var}(Z_1 \mid f)}{m} \tag{4.T5}$$

Separate the proof into three steps. Linearity gives the holdout mean's expectation; independence removes covariance terms in its variance. For the tail bound, examine the log moment-generating function of a centered bounded loss. Its second derivative is a variance under an exponentially tilted distribution, which reweights outcomes without moving them outside zero to one. Any distribution on that interval has variance at most one quarter, giving the bound used next.

$$g(s) = \log \mathbb{E}[e^{s(Z-\mathbb{E}Z)}] \tag{4.T6}$$

$$g(0) = g^{'}(0) = 0 \tag{4.T7}$$

$$\mathbb{E}_s[h(Z)] = \frac{\mathbb{E}[h(Z)e^{sZ}]}{\mathbb{E}[e^{sZ}]} \tag{4.T8}$$

$$g^{''}(s) = \mathbb{E}_s[Z^2] - (\mathbb{E}_s[Z])^2 \leq \tfrac{1}{4} \tag{4.T9}$$

Variance is no larger than mean squared distance to any chosen constant. Choosing one half bounds it by one quarter for a variable between zero and one. Integrate this curvature bound twice from zero to bound the log moment-generating function. Independence combines the bounds across observations; Markov's inequality then converts the exponential-moment bound into an upper-tail probability.

$$g(s) \leq \frac{s^2}{8} \tag{4.T10}$$

$$\Pr(\widehat{R} - R(f) \geq \epsilon \mid f) \leq e^{-sm\epsilon + ms^2/8} \quad (s > 0) \tag{4.T11}$$

Minimizing the exponent gives a tilting parameter equal to four times the tolerance. Apply the same argument to the negative centered losses and add the two tail probabilities.

$$\Pr(|\widehat{R} - R(f)| \geq \epsilon \mid f) \leq 2e^{-2m\epsilon^2} \tag{4.T12}$$

Apply the tail bound to each predictor, then add the probabilities that any predictor exceeds it. This union-bound step does not require independent candidate scores; they may use the same holdout. What must be fixed independently of that holdout is the candidate family itself. Repeatedly

inventing candidates from its results changes the information available to the search and is not covered automatically.

$$\Pr\left(\max_{1\le j\le M} |\widehat{R}_j - R(f_j)| \ge \epsilon\right) \le 2M e^{-2m\epsilon^2} \tag{4.T13}$$

□

This limited theorem explains why evaluation sample size and the number of inspected candidates both matter. It is not a guarantee for arbitrary adaptive searches, correlated data, shifted populations, or unbounded losses. Even a valid risk bound addresses the declared loss, not every safety or business requirement.

◇ Proposition 4.2: The estimand and dependence structure of cross-validation

Notation for this result

n, V, v	Development count, equal-sized fold count dividing it, and fold index 1 through V.
$D_v, D_{-v}, \lvert D_v\rvert$	Held-out index set, complementary training records, and held-out count.
A, n_{fit}, S_v	Fixed learner, fold-training count, and held-out fold-v empirical loss.
i, X_i, Y_i, L	Held-out index, random input, observed target, and declared observation loss.
$D_{n_{\text{fit}}}, R$	Fresh training sample of that size and fitted-predictor population risk; expectation also averages the fixed learner's independent randomness.
$\widehat{R}_{CV}$	Mean held-out loss across equal-sized folds.
u, Cov	Second fold index 1 through V and fold-score covariance; count each pair once, smaller index first.

Assumptions. Observations are independent and identically distributed. Fold assignment is fixed or random independently of their values, with equally sized folds. The fold count is at least two and divides the sample count. Finite first loss moments are required for expectation identities, and finite second moments for the displayed variance and covariance identity. The learning algorithm is fixed in advance and treats training observations symmetrically. Every learned transformation and selection performed by that algorithm uses only its fold's training data. The score is an average of individual losses.

The expected cross-validation loss equals the expected population risk of that learning algorithm trained at the fold-training sample size. Fold scores are

generally dependent, so their empirical dispersion does not automatically provide the standard error of the mean.

Read the complete supporting proof . *Advanced*

Cross-validation evaluates a learning procedure at specified training sizes. Use its variability descriptively unless an uncertainty method accounts for dependence. Leave-one-out changes training size but does not eliminate dependence or make every metric meaningful on a one-observation fold.

▶ Example: Count what is fitted and what is evaluated

With 500 observations in five equal folds, each of five fits trains on 400 and evaluates on 100; every observation is evaluated once. Leave-one-out instead makes 500 fits on 499 observations each and evaluates one at a time. The training sizes and evaluation dependence differ, so these are not merely coarse and precise versions of one estimate. For classification, check each evaluation fold's class counts before choosing a metric: a single-observation fold cannot support ROC AUC. The notebook verifies coverage and disjointness, then compares observed scores without assuming that the more expensive design is more trustworthy.

Interpretation and limits

Stratification balances class proportions, not independence. It cannot make repeated users independent or repair a split that trains on information from the future.

Essential

4.2 Group and temporal splits must match the intended use

Notation for this section

t	Integer time step in the three-step target example; labels arrive three steps after their row. Not an optimization iteration.

Decide whether the prediction target is a new row from a known group or a genuinely new group. People, organizations, devices, experiments and source documents can link rows. For new-group prediction, withhold entire groups. This prevents shared group identifiers across fitting and evaluation, though other dependencies may remain. A known-user score can be useful, but is not evidence about entirely new users.

Roberts and colleagues [4.3] tie splitting to the intended generalization task: a new site differs from another observation at a known site. Bengio and Grandvalet [4.8], Theorem 6, establish that ordinary K-fold errors do not yield a

universally unbiased variance estimator. Repeating partitions of the same dataset does not supply independent datasets. Report fold dispersion descriptively; a confidence interval requires a justified uncertainty procedure, not automatic division by the square root of the fold count.

Temporal evaluation trains on earlier information and evaluates later predictions. Expanding-window designs progressively add past observations; rolling-window designs retain a bounded history when that matches the intended training policy. A random split may mix different regimes and allow future information into training. Even a chronological row split can leak if target labels span future intervals, aggregated features use later data, or preprocessing is fitted beyond the training cutoff.

Choose a temporal gap by tracing when training information becomes available. A row recorded before evaluation may still have an outcome measured afterward; that label cannot train the earlier predictor. Overlapping windows, reporting delays and related entities can require several restrictions at once. A gap is only one way to enforce them. Validate the complete data-availability and update policy, not merely the fact that the splitter orders rows by time.

Roberts and colleagues [4.3] show why temporal, spatial, and hierarchical dependence can invalidate a randomly shuffled assessment. A split should reproduce the intended form of generalization, such as a new group or a genuinely later observation.

Notation for equations (4.2), (4.3)

G_{train}	Training group identifiers when evaluating unseen-group performance.
$G_{\text{validation}}$	Validation group identifiers; disjoint from training for unseen-group evaluation.
$\cap,\ \varnothing$	Intersection and empty set: no group may occur in both roles.
$i,\ j$	Indices over all training records used by a fit and all validation predictions it produces.
$t_{\text{label available},i}$	Earliest time training observation i's complete target is available to the learner.
$t_{\text{prediction},j}$	Required production time of validation prediction j; use only information already available.

$$G_{\text{train}} \cap G_{\text{validation}} = \varnothing \tag{4.2}$$

$$t_{\text{label available},i} < t_{\text{prediction},j} \tag{4.3}$$

The first condition applies when evaluating new groups. The second states that each used training label must already be available before the corresponding evaluation prediction time. The exact inequalities depend on the application's timestamp convention.

Require label availability for every training record used by each evaluated fit. Timestamp conventions determine whether equality is permitted.

▶ Example: A three-step target needs more than row order

A row at integer time t has a three-step outcome available at t plus three. For a prediction at time 40 requiring labels strictly before that instant, train only through time 36. Time 37 is admissible only if its label at time 40 is available before prediction under the declared ordering. Time 39 is inadmissible either way: its label arrives at 42. A justified gap can enforce this cutoff on a regular index. If deployment also concerns new devices, withhold entire devices. Chronology alone does not enforce entity separation; both boundaries come from the prediction contract.

Interpretation and limits

Sorting rows by time is not sufficient when labels or features use future windows. Check the availability of every piece of training information, not just the observation timestamp.

- *Essential*

4.3 The validation boundary surrounds the entire pipeline

Trace the information used by the entire learning procedure, not only the estimator's fit call. Leakage occurs when inadmissible information affects either the predictor or a development choice. Scaling all observations, selecting features against all targets, building a vocabulary before splitting, or aggregating future outcomes can leak information. Direct access to validation labels during fitting is not required.

Cross-validation must receive the whole fitting procedure, not a matrix already transformed using all rows. Fit imputation, scaling, dimension reduction and the estimator separately inside each training fold; target-based selection and PLS require the same boundary. Test it by changing only held-out values or labels: that fold's fitted preprocessing must not change. A pipeline enforces

included operations, but cannot recognize a future-outcome column or choose the correct grouping for you.

The same principle extends beyond preprocessing. Resampling belongs within the training fold; calibration needs predictions on data not used to fit its underlying classifier; stacking trains its meta-model on out-of-fold base predictions; early stopping uses a development partition, not the final test. Thresholds and hyperparameters are choices too. Draw the full sequence of fits and information flows, then identify which data are available to each operation before trusting a single aggregate score.

Varma and Simon [4.4] demonstrate the optimism created when information used for model selection also influences error estimation. This applies to learned preprocessing and feature selection as well as to the final predictor: the fitted pipeline must remain inside the training boundary. Ambroise and McLachlan demonstrate selection bias in microarray classifiers when evaluation observations help choose the genes before cross-validation [4.7]. Their correction repeats feature selection within each training partition, providing a real-data counterpart to this chapter's random-label negative control.

Notation for equations (4.4), (4.5), (4.6)

k	Fold index 1 through V; refit preprocessing and estimator independently per fold.
d	Input-column count before fitted preprocessing; output feature count may differ.
D_{-k}	Training inputs and targets outside held-out fold k.
$X_{-k},\ y_{-k}$	Training input matrix (d columns, one row per training record) and aligned target column.
$X_k,\ \hat{y}_k$	Held-out input matrix (d columns, one row per held-out record) and aligned prediction column.
$\hat{T}_k$	Stored training-only preprocessing, reused without refitting.
$\hat{f}_k$	Estimator fitted to transformed training inputs and their targets for fold k.
$\mathrm{fitPreprocessor}$	Returns a fitted transformation; supervised preprocessing may use training targets, never held-out targets.
$\mathrm{fitModel}$	Fits an estimator to transformed training inputs and aligned targets.

$$\hat{T}_k = \mathrm{fitPreprocessor}(D_{-k}) \tag{4.4}$$

$$\hat{f}_k = \text{fitModel}(\hat{T}_k(X_{-k}), y_{-k}) \tag{4.5}$$

$$\hat{y}_k = \hat{f}_k(\hat{T}_k(X_k)) \tag{4.6}$$

Fit preprocessing on the training partition, fit the estimator on transformed training inputs and training targets, then transform held-out inputs using the stored preprocessing state. Held-out targets do not enter these fitting operations.

Input matrices have one row per observation; transformed feature counts must agree between training and held-out inputs within each fold.

▶ Example: Find impressive patterns in random labels

Use random features independent of a random binary target as a leakage diagnostic. Selecting the twenty features most correlated with all labels lets held-out targets shape the representation before cross-validation begins. Repeating selection within each training fold removes that path. The notebook labels whole-dataset selection an invalid negative control and reports both scores. A valid finite score can differ from chance, and a leaky run need not win every time. The information path establishes the defect; the observed score difference only illustrates its possible consequence.

Interpretation and limits

A pipeline is not a universal leakage detector. It protects the operations placed inside it, but not future-derived raw features, shared entities or test-driven manual decisions.

Recommended

4.4 Search selects a model; nested validation evaluates that selection

Hyperparameter search compares complete fitting choices, such as penalty strength, neighborhood size or tree depth. Grid search evaluates a declared Cartesian product; randomized search samples specified distributions within a computational budget. The search space is itself a modeling choice. Use sensible scales, consistent splits and a documented budget. Unequal tuning effort is not evidence of an intrinsic algorithmic advantage.

The best cross-validation score observed during a search is used to choose a candidate and is therefore a selection result. Searching many weakly differentiated candidates can exploit chance variation in that score. An untouched final test can evaluate the selected procedure, or nested cross-validation can create outer held-out folds. In each outer training partition,

inner validation performs the complete search; the outer fold evaluates the resulting selected model without contributing to its choice.

Nested validation estimates the performance of a specified selection procedure trained on the outer-training sample sizes. It does not guarantee the same selected hyperparameter in every fold, and differing selections can be informative. Group or temporal restrictions must be honored at both levels, with correctly passed metadata and availability constraints. After the protocol is fixed, perform the declared search on all development data, refit as specified and evaluate once on the final test if one was reserved.

Cawley and Talbot [4.5] explain how the selection criterion itself can be overfitted. Varma and Simon [4.4] motivate the nested construction used here, with an inner selection process and an outer assessment of that entire process.

Notation for equations (4.7), (4.8)

k	Outer-split index; each split has its own inner model-selection experiment.
$\mathcal{H},\ h$	Predeclared hyperparameter search space and one candidate configuration.
$\widehat{R}_{\text{inner},k}(h)$	Candidate inner-validation risk using only outer split k's training partition.
$\hat{h}_k$	Configuration minimizing inner-validation risk in the current outer split.
argmin	Set of minimizing configurations; select one by a predeclared deterministic tie rule.
$\hat{f}_{\hat{h}_k}$	Predictor/preprocessing refitted on outer-training data with the inner-selected configuration.
$\widehat{R}_{\text{outer},k}$	Empirical-risk evaluator on untouched outer held-out observations.
s_k	Outer score evaluating the entire inner-selection-and-refit procedure.

$$\hat{h}_k \in \operatorname*{argmin}_{h \in \mathcal{H}} \widehat{R}_{\text{inner},k}(h) \tag{4.7}$$

$$s_k = \widehat{R}_{\text{outer},k}(\hat{f}_{\hat{h}_k}) \tag{4.8}$$

For each outer split k, the hyperparameter h is selected using inner validation contained entirely within outer training data. The selected fit is then evaluated on the untouched outer fold.

The predictor also depends on outer split k through its training data, although that dependence is suppressed in its subscript.

◇ Proposition 4.3: Why selecting the lowest noisy score is optimistic

Notation for this result

j, M	Candidate index from one through a finite number of candidate predictors.
$R_j, \widehat{R}_j, E_j$	Candidate's true risk, observed evaluation estimate, and zero-mean score estimation error.
s	Strictly positive standard deviation of each Gaussian score error in this two-candidate example, measured in loss units.
s^2	Common positive variance of the independent Gaussian score errors, equal to the square of their standard deviation.
$\mathcal{N}$	Normal distribution with the displayed mean and variance.
t, π	Nonnegative integration variable for the Gaussian absolute moment and the circle constant, respectively.
R	Shared true risk of the two candidates in the equal-risk illustration.

Assumptions. Each candidate's evaluation score is an unbiased estimate of its fixed population risk, with finite expectation. Candidates have been specified without using these evaluation scores. The exact two-candidate example additionally assumes equal true risks and independent Gaussian score errors with a common positive variance. It is an explanatory noise model, not an exact model of every validation metric.

The expected lowest observed score is no larger than the lowest true risk. Even when two candidates have identical quality, selecting by evaluation noise creates a downward bias in the reported winning score.

□ Proof *Advanced*

$$\widehat{R}_j = R_j + E_j \tag{4.T20}$$

$$\mathbb{E}[E_j] = 0 \tag{4.T21}$$

Before seeing evaluation noise, fix any one candidate. The winning observed score is no greater than that candidate's observed score, on every realization. Taking expectations preserves the inequality. Then choose the fixed candidate with smallest true risk. This order proves optimism without assuming independent errors between candidates; choosing the comparator after seeing the noise would change the argument.

$$\mathbb{E}[\min_j \widehat{R}_j] \leq \min_j \mathbb{E}[\widehat{R}_j] = \min_j R_j \tag{4.T22}$$

For equally good candidates, the winning score differs only because of its error. Write the smaller of two errors as half their sum minus half their absolute difference. The sum averages to zero, but the absolute difference is nonnegative: selecting the smaller error leaves a negative contribution. Under the stated independent Gaussian-error model, the difference is Gaussian with twice the individual error variance.

$$\min(E_1, E_2) = \frac{E_1 + E_2 - |E_1 - E_2|}{2} \tag{4.T23}$$

$$E_1 - E_2 \sim \mathcal{N}(0, 2s^2) \tag{4.T24}$$

Integrating the positive half of a centered normal density and doubling gives its expected absolute value. Substitution provides an explicit amount of selection optimism, even though neither candidate is genuinely better.

$$\mathbb{E}|E_1 - E_2| = \frac{1}{s\sqrt{\pi}} \int_0^\infty t e^{-t^2/(4s^2)}\, dt = \frac{2s}{\sqrt{\pi}} \tag{4.T25}$$

$$\mathbb{E}[\min_j \widehat{R}_j] = R - \frac{s}{\sqrt{\pi}} \tag{4.T26}$$

This calculation concerns the score used for selection, not a necessary improvement or deterioration in the selected model's true risk. Here both true risks are identical. Correlation between score errors changes the amount, but does not rescue reporting the selected inner score as untouched evidence. Repeatedly redesigning the search after seeing results compounds the information reuse.

□

A favorable search result can reflect successful exploitation of measurement noise. Reserve independent evaluation for the selected procedure, and report search space and budget. The effect becomes more consequential when validation scores are noisy or many alternatives are inspected.

◇ Proposition 4.4: Nested evaluation protects the entire selection procedure

Notation for this result

$D_{\text{in}}, D_{\text{out}}$	Outer training and independent outer evaluation records; inner validation stays within the former.
$B, \widehat{h}$	Complete inner selection and chosen configuration, including tuned thresholds or representations.
$A, \hat{f}$	Refitting procedure and complete predictor; retain fitted preprocessing for inference.
U	All independent selection/refitting randomness, included in conditioning.
$\widehat{R}_{D_{\text{out}}}, R$	Outer mean loss and population risk under the same input-target distribution.

Assumptions. An outer evaluation set is independent of its outer training data under the declared sampling model. Every inner operation, including preprocessing, feature selection, resampling, thresholding, and tuning, is a function of outer training data and independent algorithm randomness only. The evaluation loss has finite expectation. The procedure and reporting rule are fixed without reacting to outer evaluation results.

Conditioned on the outer training information, the outer mean loss estimates the population risk of the complete selected-and-refitted predictor. Information isolation, not the mere presence of nested loops, supplies this property.

□ Proof *Advanced*

$$\widehat{h} = B(D_{\text{in}}) \tag{4.T27}$$

$$\hat{f} = A(D_{\text{in}}, \widehat{h}) \tag{4.T28}$$

For the outer-fold argument, imagine freezing everything determined by outer training: records, selection randomness, chosen configuration and refitted predictor. Conditional on that information, the predictor is fixed and the untouched outer records remain fresh observations. The independent-holdout expectation result therefore applies. The essential separation is that outer evaluation outcomes did not help create the predictor being scored.

Kohavi [4.2], Sections 4-6, studies cross-validation empirically with decision trees and naive Bayes; the recommendation for stratified ten-fold validation is conditional on the experiments, not a universal theorem. Varma and Simon

[4.4] address a different issue: tuning and evaluating on the same folds induces selection bias. Repeat learned preprocessing and tuning inside each outer training fold. First choose a split that respects future use; neither ten folds nor nesting repairs a subject or time leak.

$$\mathbb{E}[\widehat{R}_{D_{\text{out}}}(\hat{f}) \mid D_{\text{in}}, U] = R(\hat{f}) \tag{4.T29}$$

Now average over possible outer training samples. The result evaluates the whole selection-and-refitting procedure at that training size, not a single fixed hyperparameter. If feature selection inspects outer labels, conditioning on the training information no longer fixes the predictor independently of outer outcomes; the preceding expectation argument breaks. Putting later operations in a pipeline cannot undo a target-derived feature already prepared outside it.

An outer result can still reveal a defect requiring further development. Once it influences a new search or a choice between competing selection procedures, it has become development information. A new independent test is then needed for a new final claim. Likewise, temporal or group-based deployment requires outer and inner designs that respect those dependencies rather than borrowing an independent-row argument.

□

The evaluated object is the whole learning algorithm, not just its final estimator class. After choosing the protocol, refit on the permitted development data and keep any final test outside that process. Statistical separation supports an interpretable claim; it does not certify deployment readiness by itself.

▶ Example: Count the fits in a nested experiment

With three regularization candidates and three inner folds, each outer iteration makes three times three, or nine, scoring fits. Refit the inner-selected candidate once on the outer training partition, then evaluate on its untouched outer fold. Three outer folds therefore require 27 scoring fits plus three refits: 30 fits before any later full-development search. Different outer folds may select different values; that is consistent with evaluating a selection procedure. The notebook records each choice and outer score. Do not replace this evaluation with the best inner score, which helped select the candidate and is not independent evidence.

Interpretation and limits

Reporting the maximum search score as if it were an untouched evaluation ignores selection. Nested validation must repeat the complete tuning process inside each outer training partition.

Algorithm 4.1. Nested validation of a complete pipeline

Algorithm references. Nested evaluation of the complete selection procedure follows Varma and Simon [4.4] and Cawley and Talbot [4.5]. The reporting and failure-handling contract is an explicit teaching specification.

Notation and assumptions

$D,\ \mathcal{H},\ h$	Development data excluding any final test, nonempty finite candidate set, and candidate configuration. D_I, X_I and y_I restrict data, input rows and targets to indices I.
$(T_k, E_k), \quad k = 1, \dots, K$	Nonempty disjoint outer index sets and a positive finite split count.
$(I_{kv}, V_{kv}), \quad v = 1, \dots, m_k$	Inner-training/validation indices contained in T_k, and positive fold count.
$f_{khv}, \hat{f}_k, \widehat{h}_k$	Inner fitted pipeline, outer-training refit, and selected configuration.
$q_{khv}, \bar{q}_{kh}, s_k$	Inner scalar score, aggregated candidate score, and untouched outer score.
$M, \mathrm{Agg}, \mathrm{Select}$	Declared evaluation metric, fold weighting/aggregation rule, and selection rule respecting metric direction and deterministic ties.
$\widehat{y}_k,\ \mathcal{R},\ \mathcal{P}$	Outer-fold prediction vector, fold-indexed result records, and complete search/split/seed/version specification.

Fit_h clones the entire pipeline and learns preprocessing, feature selection, resampling and estimator state only on its supplied training indices. InnerSplit uses only outer-training records. Refit preserves internal stopping/calibration holdouts required by the selected procedure. Check disjoint nonempty splits, group/time and metric support. Log failures and apply policy. If none qualify, record a failed fold. Record immutable copies indexed by fold.

Algorithm 4.1

Input: Development data and finite nested split plans. $D,\ (T_k, E_k)$

Candidates, metric, aggregation and tie rule. $\mathcal{H}, M, \mathrm{Agg}, \mathrm{Select}$

Seeds, per-fit budgets and declared failure/undefined-metric policy.

Output: Outer scores, indexed out-of-fold predictions and selected configurations.

$\mathcal{R}$

Complete evaluated search specification. $\mathcal{P}$

Check Both-level data boundaries, metric support and finite fit budgets.

Initialize the immutable protocol record from the candidate set, split plans, seeds, metric direction, aggregation, tie rule, budgets, failure policy and software versions; initialize the results list:

$\mathcal{R} \leftarrow [\,]$

For $k = 1, \dots, K$

  Split $(I_{kv}, V_{kv})_{v=1}^{m_k} \leftarrow \mathrm{InnerSplit}(T_k)$

  For $(h, v) \in \mathcal{H} \times \{1, \dots, m_k\}$

    $f_{khv} \leftarrow \mathrm{Fit}_h(D_{I_{kv}})$

    $q_{khv} \leftarrow M(y_{V_{kv}}, f_{khv}(X_{V_{kv}}))$

  End for

  Aggregate For each candidate h. $\bar{q}_{kh} \leftarrow \mathrm{Agg}_v(q_{khv})$

  $\widehat{h}_k \leftarrow \mathrm{Select}_{h \in \mathcal{H}}\ \bar{q}_{kh}$

  $\hat{f}_k \leftarrow \mathrm{Refit}_{\widehat{h}_k}(D_{T_k})$

  $\widehat{y}_k \leftarrow \hat{f}_k(X_{E_k})$

  $s_k \leftarrow M(y_{E_k}, \widehat{y}_k)$

  $\mathcal{R}_k \leftarrow (E_k, \widehat{h}_k, s_k, \widehat{y}_k)$

End for

Return $\mathcal{R}, \mathcal{P}$

Why this works. Outer scores are generally dependent and describe outer-training sample sizes. Do not redesign the procedure using these scores and call them untouched evaluation. A later all-development fit is a separate operation.

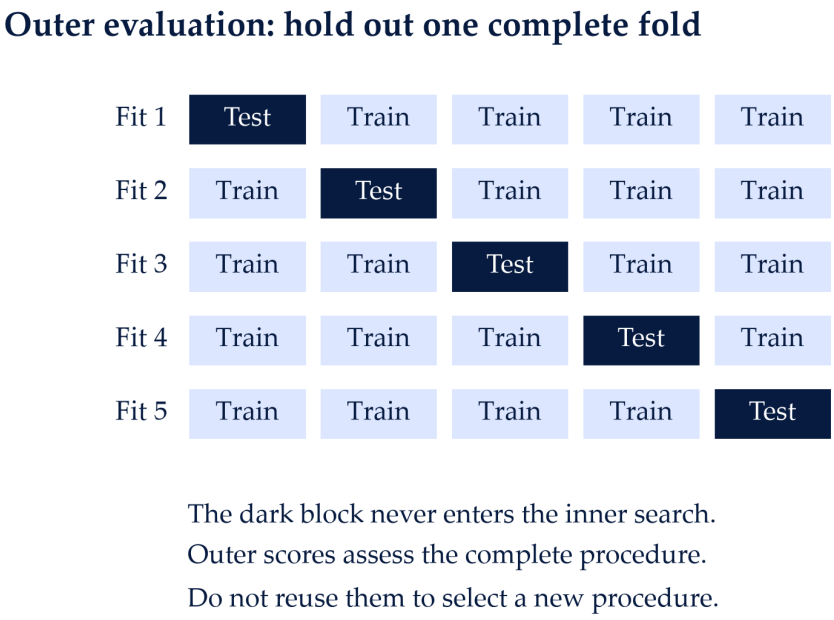


Figure 4.1. Read each row as a separate outer fit: its dark fold is excluded from the entire search performed on the light folds. Across five rows every fold is evaluated once. Scores assess the selection procedure; redesigning it from these results consumes their untouched-evaluation role.

Essential

4.5 Uncertainty, reproducibility and the final test lock

Before reporting uncertainty, name what varies: evaluation cases, training samples or stochastic fitting. Cross-validation folds share training observations, making their scores generally dependent. Their standard deviation measures observed spread. Dividing it by the square root of the fold count neglects covariance unless independence is justified. Even identical fold scores do not establish zero uncertainty about future performance.

A paired bootstrap keeps each evaluation target attached to its prediction and resamples those pairs. With independent examples and a fixed fitted model, this approximates variation from drawing a different evaluation sample. It does not rerun training or selection and cannot capture uncertainty from those stages or future shift. For groups or time dependence, preserve the relevant

structure when resampling. Rare outcomes can leave some resamples without the cases needed to define the metric, making intervals unstable or unavailable.

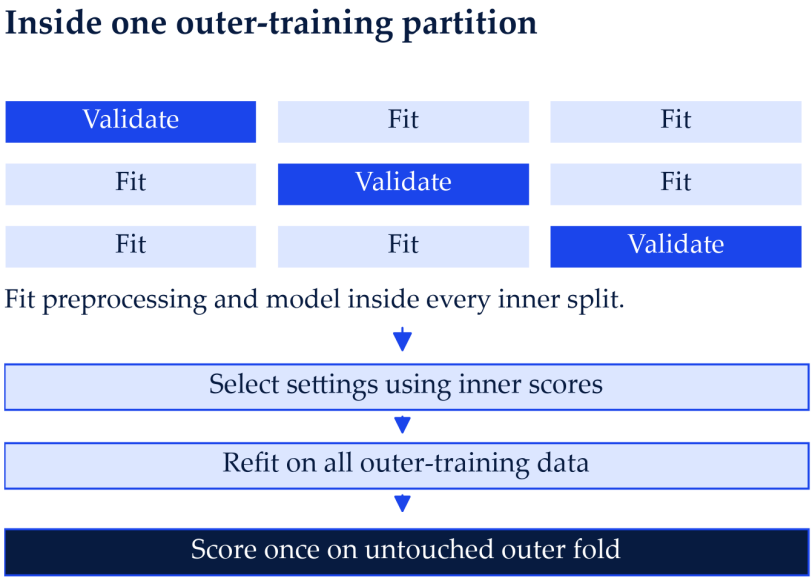


Figure 4.2. Inside one outer-training partition, three-fold inner validation selects settings. Every split fits preprocessing and the estimator using its own fitting rows only. The selected settings are refitted on all outer-training rows, then scored once on the untouched outer fold.

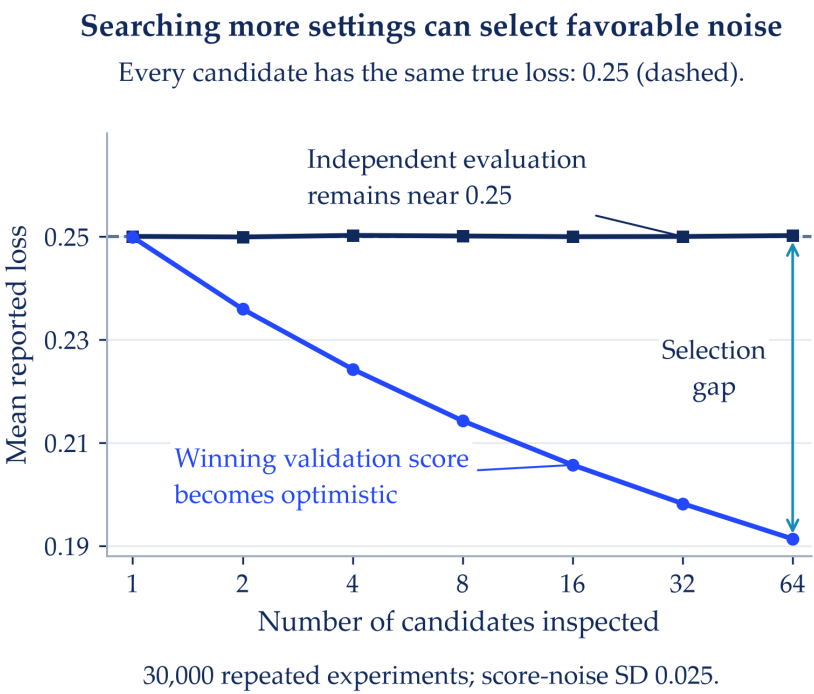


Figure 4.3. In 30,000 simulated experiments, every candidate has identical true loss 0.25 and independent Gaussian score noise with standard deviation 0.025. Searching more candidates lowers the winning validation score but not its independent evaluation score. Selection rewards favorable noise even when underlying quality is unchanged.

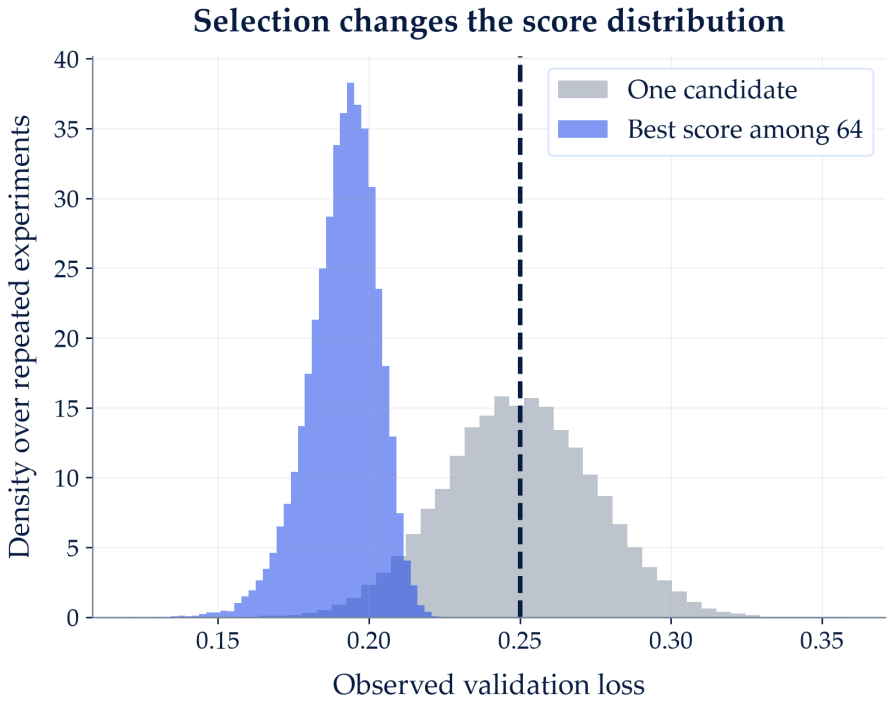


Figure 4.4. Score distributions for one candidate and the best validation score among 64 equally good candidates. Both come from the same 30,000-experiment score-noise simulation. The minimum is systematically optimistic. This controlled illustration does not claim that real candidate scores are independent.

Reproducibility requires more than a random seed. Record data provenance, split identifiers, feature definitions, metric configuration, dependency versions, search settings and the selected procedure. Test the inference interface on new batches and malformed inputs. A final test lock means no features, thresholds or hyperparameters are selected using its outcomes. When performance is disappointing, report it, investigate using development data and seek new independent evaluation for revised claims rather than silently reusing the same test.

Efron and Tibshirani [4.6] develop bootstrap reasoning for sampling uncertainty. Such resampling is conditional on an appropriate sampling design; it does not repair dependent observations, repeated test-set decisions, or a test population different from the intended use. Bengio and Grandvalet show that the usual K-fold experiment does not supply a universally unbiased variance

estimator valid for all data distributions [4.8]. This supports treating fold-score dispersion as descriptive unless an uncertainty method's additional assumptions are justified, rather than automatically converting it into a confidence interval. The reusable-holdout work of Dwork and colleagues shows that adaptive reuse can be controlled by specially designed mechanisms that restrict the information released from the holdout [4.9]. It is not permission to inspect an ordinary test score repeatedly while tuning; the simple workflow here retains its final test lock.

Notation for equations (4.9), (4.10)

n	Independent original-evaluation count and size of every bootstrap resample.
$b,\ \hat{s}^{(b)}$	Replicate index 1 through the declared count and its resampled-data metric.
$j,\ i_j$	Resample position 1 through n and original index drawn from 1 through n for it.
$y_{i_j},\ \hat{y}_{i_j}$	Selected observation's target and fixed-model prediction, kept paired.
M	Metric applied to the entire resampled target-prediction sequence.
$\mathrm{Uniform}\{1,\dots,n\}$	Uniform distribution over original indices; independent draws with replacement.

$$i_j \overset{\text{iid}}{\sim} \mathrm{Uniform}\{1,\dots,n\}, \qquad j = 1,\dots,n \tag{4.9}$$

$$\hat{s}^{(b)} = M\big((y_{i_j}, \hat{y}_{i_j})_{j=1}^{n}\big) \tag{4.10}$$

Keep the fitted predictor fixed and resample whole independent evaluation observations, retaining each observation's target and prediction together. The paired bootstrap varies which evaluated cases are represented; it does not retrain the model. Its percentile interval is approximate under the stated sampling assumptions. Grouping, temporal dependence and population shift require more than this independent-row resampling scheme.

The ordered collection retains repeated sampled indices. Draw indices independently with replacement within each replicate and independently across replicates; iid means independent and identically distributed. The fitted predictor stays fixed.

▶ Example: Resample four paired outcomes

Suppose a frozen classifier is correct on evaluation rows 1, 2 and 4 and wrong on row 3: accuracy is 3/4. A bootstrap draw of indices (1, 1, 3, 4) gives accuracy

3/4; (3, 3, 2, 4) gives 1/2. Copy each selected row's target and prediction together, including duplicates. Across all 4^4 = 256 equally likely ordered draws, mean bootstrap accuracy is 0.75 and its standard deviation is about 0.217. Four cases make the resampling visible; they do not support a precise population-accuracy claim. No classifier is refitted in this calculation.

▶ Example: What a bootstrap interval does and does not say

Freeze the classifier and development choices before predicting an independent test sample. Bootstrap observation indices, retaining each label-probability pair and every repeated index. This approximates test-sample variability for that fixed model, not uncertainty from retraining, search or shift. Compute ROC AUC only for resamples containing both classes, and count the excluded one-class samples. Discarding them conditions the retained distribution on both classes being present; frequent exclusions undermine an ordinary unconditional percentile-interval interpretation. A class-stratified bootstrap instead fixes class counts and answers a different conditional question. The notebook states this scope and uses neither interval nor resamples to select models or thresholds. Define the pairing, resampling unit and conditioning before interpreting an interval.

Interpretation and limits

A seed does not imply validity, and a narrow interval does not protect against leakage or distribution shift. Reproducible wrong experiments remain wrong.

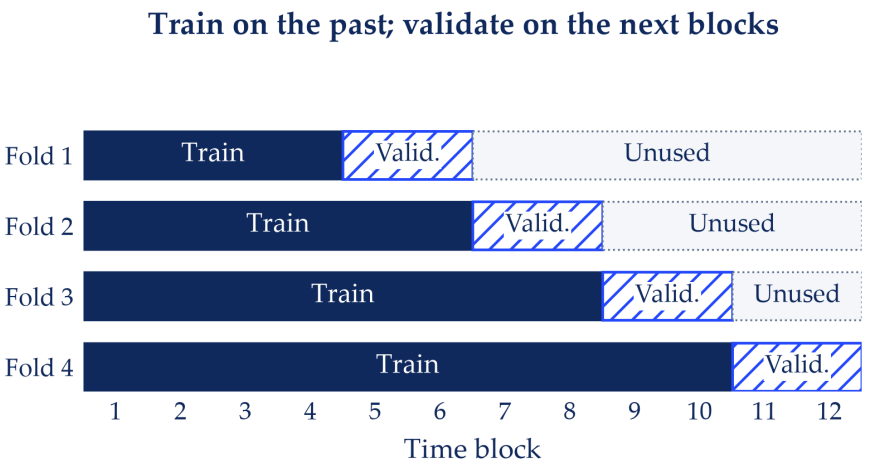


Figure 4.5. A schematic expanding-window split. Each validation block occurs after its training blocks. Real applications may also need a gap, grouped entities, and a separate final test period. Random row splitting answers a different question.

Table 4.1. Choose the split before comparing algorithms

Design	Appropriate sampling structure	Common failure
Random / stratified	Approximately independent cases; stratification preserves class proportions.	Related cases on both sides create overly optimistic evidence.
Grouped	Several rows belong to the same person, device or source.	Random row splits leak group-specific information.

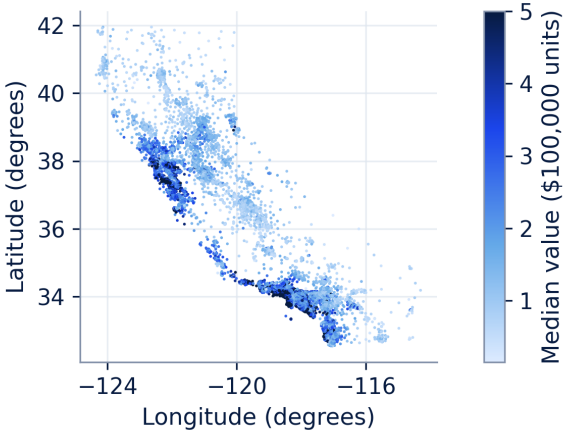
Design	Appropriate sampling structure	Common failure
Temporal	Past observations predict a future period.	Training on future information breaks the intended deployment boundary.
Nested validation	A fixed search procedure needs outer evaluation isolated from its tuning decisions.	Reporting the best inner score as independent performance.

Table 4.2. Chapter synthesis

Concept	Main conclusion	Engineering check
Fold boundaries	Preprocessing and model fitting share the same training boundary.	No full-dataset scaling, vocabulary or feature selection before a fold.
Search	More configurations create more opportunities for selection optimism.	Record the complete search space and selection rule.
Nested evaluation	Outer folds evaluate the full selection procedure.	Inner folds must not use outer-test outcomes.
Final test	A frozen procedure earns an independent final measurement.	Do not adjust the model after observing the final score.

California Housing

20,640 block groups	**8** predictors	**1990** census data



All 20,640 census block groups, located by their recorded coordinates. Color shows the historical median-value target, including its upper cap.

California Housing describes 20,640 census block groups from the 1990 US census. Eight predictors summarize income, housing, population and location; the target is median house value in units of $100,000. Each row is an area, not an individual home, making geographic evaluation a distinct prediction question. [4.10]

Here: the case selects 10,000 source rows and compares random splitting with a predeclared geographic boundary. The response is in $100,000 units and has an upper cap; a row is an area, not one home.

Pace and Barry; census-derived StatLib / scikit-learn distribution. Historical values only. Source / terms.

Case Study: California Housing: random interpolation is not northward transfer

Problem and data

California Housing records eight predictors and median house values for census block groups observed in 1990. Compare two contracts: predicting randomly withheld block groups from the observed region, and predicting northern block groups after learning only from observations south of latitude thirty-eight degrees. These are different populations, even when evaluated with the same loss. Fix the geography rule and a ten-thousand-row computational subset before modeling, without selecting a boundary that creates an attractive result.

Experimental design

Give the random and geographic protocols equal development and final-test sample counts. For each, compare four declared scaling-and-estimator pipelines using three development folds. Random folds shuffle rows; geographic folds withhold complete half-degree coordinate cells inside the southern development region. Select by mean fold squared error, refit the selected pipeline within its development boundary, and evaluate it and a development-mean reference once on that protocol's test. Preserve source-row identifiers, fold membership, seeds and dollar-unit metrics.

Measured Reference Run

A fixed 10,000-row subset produced two predeclared protocols, each with 8,537 development and 1,463 test rows. The spatial test held out latitude at or above 38 degrees. Four standardized pipelines used either random three-fold validation or three folds grouped by half-degree spatial cells. Both searches selected ten-neighbor regression. Dollar metrics below concern the historical capped target.

Table 4.3. Measured case-study evidence

Test protocol	Procedure	MAE ($)	RMSE ($)
Random rows	Ten neighbors	39603.638	56835.263
Random rows	Development mean	91625.475	114634.94

Test protocol	Procedure	MAE ($)	RMSE ($)
Northward transfer	Ten neighbors	52336.033	64479.838
Northward transfer	Development mean	96092.706	106107.844

The selected rule beat its mean reference under both contracts. Its northern test error rose while baseline RMSE fell: changing the evaluation population changed even the reference difficulty. The random-row and geographic scores answer different prediction questions, not two interchangeable estimates of one error.

Failure Analysis And Limits

The random-row protocol predicts within the sampled geographic mixture; the northern holdout tests transfer from the southern development region. Their error contrast is not a causal leakage estimate: source records overlap and development geography and targets differ. Spatial cells need not be independent. The target cap near $500,000 also limits these dollar errors to the stored response, not uncapped prices or a future market.

Historical saved run; retain seed 2027, source-row identities, latitude boundary, both fold plans and the identical four-candidate budgets.

▶ Example: inspecting the real data

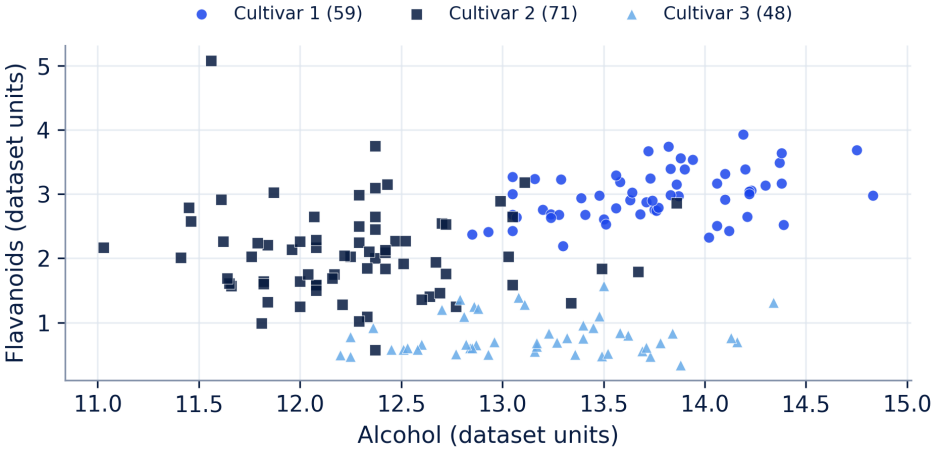
The first source record has latitude 37.88, longitude -122.23 and target 4.526. Its recorded median house value is therefore 4.526 times $100,000 = $452,600. Under the declared geographic rule it is south of the final-test boundary, if selected into the computational subset; this conversion is observed-data arithmetic, not a model performance claim.

Dataset source and documentation

Companion: Week 04, Real-Data Case Study notebook. Reproduce the declared run, then investigate a changed condition. No fixed benchmark score is promised.

Wine Recognition

178	13	3
wine analyses	measurements	cultivars



All 178 wines in two chemical coordinates. Cultivar labels describe the dataset; clustering exercises must not use them to choose settings.

Wine Recognition records 13 chemical measurements for 178 wines from three cultivars grown in one Italian region. It supports classification and scale-sensitive clustering. Its cultivar labels describe origin, not quality scores; it is a different dataset from Portuguese red Wine Quality. [4.11]

Here: cross-validation first uses these measurements for cultivar classification. Chapter 8 reuses them for clustering with labels withheld from model selection. This is not the separate Wine Quality rating dataset.

Aeberhard and Forina / UCI, CC BY 4.0. Source / terms.

A Controlled Learning Cycle

The California Housing case makes geographic transfer part of the evaluation question. Week 4 notebooks connect that boundary to leakage, delayed labels and nested selection. The Wine laboratory isolates five-fold membership, fold-local scaling and candidate evidence, not the broader split designs listed below. Record a baseline, one controlled change, the metric and fitting roles; explain which generalization question the resulting comparison actually answers.

Matched practice and simulator protocol · Packaged edition; launch instructions in the guide.

Laboratory 04 | Validation And Model Selection

Five-fold membership, fold-local scaling and candidate evidence.

Five-fold cross-validation exposes the actual Wine training rows and the scaling parameters fitted inside each fold.

Open Laboratory 04 | Use the live view for readable controls.

Transfer: California Housing geographic transfer → Wine KNN classification. No temporal or grouped split, nested CV, or geographic-transfer simulation.

Chapter Capability

Evaluate the complete selection procedure without allowing preprocessing, tuning, or repeated test inspection to cross the evaluation boundary.

Week 04: separate practice, solutions and assessment

References

[4.1] Stone, M. (1974). Cross-Validatory Choice and Assessment of Statistical Predictions. Journal of the Royal Statistical Society: Series B 36(2), 111-133. Source / DOI

[4.2] Kohavi, R. (1995). A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. Proceedings of the 14th International Joint Conference on Artificial Intelligence. Source / DOI

[4.3] Roberts, D. R.; et al. (2017). Cross-validation strategies for data with temporal, spatial, hierarchical, or phylogenetic structure. Ecography 40(8), 913-929. Source / DOI

[4.4] Varma, S.; Simon, R. (2006). Bias in error estimation when using cross-validation for model selection. BMC Bioinformatics 7, Article 91. Source / DOI

[4.5] Cawley, G. C.; Talbot, N. L. C. (2010). On Over-fitting in Model Selection and Subsequent Selection Bias in Performance Evaluation. Journal of Machine Learning Research 11, 2079-2107. Source / DOI

[4.6] Efron, B.; Tibshirani, R. J. (1994). An Introduction to the Bootstrap. 1st ed., paperback edition, Chapman & Hall. Source / DOI

[4.7] Ambroise, C.; McLachlan, G. J. (2002). Selection bias in gene extraction on the basis of microarray gene-expression data. Proceedings of the National Academy of Sciences 99(10), 6562-6566. Source / DOI

[4.8] Bengio, Y.; Grandvalet, Y. (2004). No Unbiased Estimator of the Variance of K-Fold Cross-Validation. Journal of Machine Learning Research 5, 1089-1105. Source / DOI

[4.9] Dwork, C.; Feldman, V.; Hardt, M.; Pitassi, T.; Reingold, O.; Roth, A. (2015). The reusable holdout: Preserving validity in adaptive data analysis. Science 349(6248), 636-638. Source / DOI

[4.10] Pace, R. K.; Barry, R. (1997). Sparse spatial autoregressions. Statistics & Probability Letters, 33(3), 291-297. Source / DOI

[4.11] Aeberhard, S.; Forina, M. (1992). Wine. UCI Machine Learning Repository. Source / DOI

[4.12] Hoeffding, W. (1963). Probability Inequalities for Sums of Bounded Random Variables. Journal of the American Statistical Association 58(301), 13-30. Source / DOI

Advanced

The Estimand And Dependence Structure Of Cross-Validation

Supporting proof for Section 4.1.

Notation For This Result

n, V, v	Development sample count, number of equal-sized folds dividing that count, and fold index from one through V.
$D_v, D_{-v}, \lvert D_v \rvert$	Held-out observation-index set, complementary training records, and number of held-out observations.
A, n_{fit}, S_v	Fixed learning procedure, its fold-training sample count, and empirical loss on held-out fold v.
i, X_i, Y_i, L	Held-out index, random input, observed target, and the declared per-observation loss.
$D_{n_{\text{fit}}}, R$	Fresh training sample of the indicated size and population risk of the fitted predictor. The displayed expectation also averages any independent randomness used by the fixed learning procedure.
$\widehat{R}_{CV}$	Mean of the held-out loss estimates across the equally sized folds.
u, Cov	Second fold index from one through V and covariance between fold scores; each pair appears once, with the first index smaller.

Assumptions. Observations are independent and identically distributed. Fold assignment is fixed or random independently of their values, with equally sized folds. The fold count is at least two and divides the sample count. Finite first loss moments are required for expectation identities, and finite second moments for the displayed variance and covariance identity. The learning algorithm is fixed in advance and treats training observations symmetrically. Every learned transformation and selection performed by that algorithm uses only its fold's training data. The score is an average of individual losses.

The expected cross-validation loss equals the expected population risk of that learning algorithm trained at the fold-training sample size. Fold scores are generally dependent, so their empirical dispersion does not automatically provide the standard error of the mean.

□ **Proof** *Advanced*

$$n_{\text{fit}} = n\frac{V-1}{V} \tag{4.T14}$$

$$S_v = \frac{1}{|D_v|}\sum_{i\in D_v} L(Y_i, A(D_{-v})(X_i)) \tag{4.T15}$$

Condition on one fold's training records and training randomness. Its held-out examples are independent fresh observations for the resulting predictor. Their conditional mean loss is therefore that predictor's population risk. Taking expectation over training data produces the same algorithm-risk quantity for every equal-sized fold.

$$\mathbb{E}[S_v] = \mathbb{E}_{D_{n_{\text{fit}}}}[R(A(D_{n_{\text{fit}}}))] \tag{4.T16}$$

$$\widehat{R}_{CV} = \frac{1}{V}\sum_{v=1}^{V} S_v \tag{4.T17}$$

$$\mathbb{E}[\widehat{R}_{CV}] = \mathbb{E}_{D_{n_{\text{fit}}}}[R(A(D_{n_{\text{fit}}}))] \tag{4.T18}$$

This identity is not an unbiasedness statement for the particular final model trained on every development observation. Nor does it imply independent fold scores: two fits share many training records, and records held out for one score are used to train other predictors. Expanding the variance of a sum retains their covariance terms.

$$\operatorname{Var}(\widehat{R}_{CV}) = \frac{1}{V^2}\left[\sum_v \operatorname{Var}(S_v) + 2\sum_{u<v}\operatorname{Cov}(S_u, S_v)\right] \tag{4.T19}$$

Replacing this expression by the fold variance divided by the fold count silently removes those covariances. Unequal folds require observation weighting for pooled mean loss. Nonlinear metrics, including F1 and AUC, are not covered by the additive-loss identity and can differ between fold averaging and pooled out-of-fold predictions. Stratification based on targets also changes the simple independent-fold argument and requires its own sampling interpretation.

□

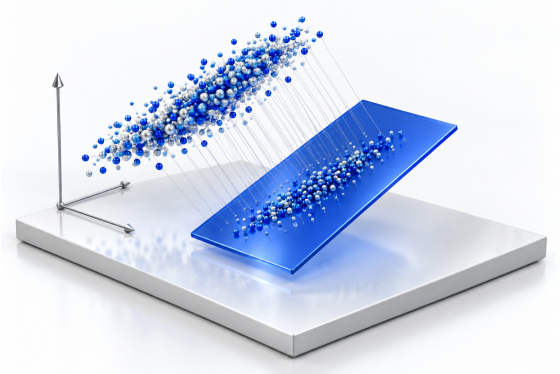
Cross-validation evaluates a learning procedure at specified training sizes. Use its variability descriptively unless an uncertainty method accounts for

dependence. Leave-one-out changes training size but does not eliminate dependence or make every metric meaningful on a one-observation fold.

Return to the chapter and its references

CHAPTER 05

Representations, PCA, And PLS



Reading path

Start with Essential. The other levels offer more depth.

Essential
5.1 5.2 5.3

Recommended
5.4 5.5

Advanced
Proofs 5.1 5.2 5.3 5.4

Worked example Lab task

A representation can discard useful information or make it easier to learn. Compare compression, prediction, and visualization as different objectives before interpreting a low-dimensional plot.

What You Will Be Able To Do

- Derive and implement a principal-component projection using the singular value decomposition.

- Select representation dimension inside validation folds and distinguish reconstruction from prediction.

- Compare principal-component regression with partial least squares on a controlled target-dependent example.

- Interpret a t-SNE visualization without treating its geometry as proof of clusters.

Before you begin: Matrix multiplication and orthogonal vectors; Linear regression and regularization; Train, validation, and test separation.

Reconstruct an observation with PCA before comparing predictive dimensions; then revisit PLS1 and nonlinear visualization.

Algorithm scope: Complete teaching procedures: PCA with an SVD primitive and regression-deflation PLS1. t-SNE is interpreted, not implemented; representation selection remains part of the training procedure.

Shared chapter notation

n, d	Positive observation and measured-feature counts, respectively, for the training or centered data specified locally.

X	Real training matrix (n observation rows, d feature columns), in the chosen measurement units.
X_c	Column-centered training/data matrix (n by d), obtained by subtracting each feature mean; n counts observations and d counts features.
X, X_c	Original and column-centered training matrices (n by d), with observation rows and feature columns, after any training-fitted feature scaling.
r	Integer retained-principal-component count, from zero through the smaller of n and d.
Z	Training principal-score matrix (n by r): centered observation rows multiplied by the retained right singular vectors.
S	For $n > 1$, symmetric positive-semidefinite sample covariance of the centered features (d by d), with divisor n minus one. Not Algorithm 5.1's singular-value matrix.
v	Candidate d-entry projection vector of unit Euclidean length; projection gives one scalar score per observation. A variance-maximizing direction is a leading covariance eigenvector.
Cov	Sample covariance across paired centered score vectors for $n > 1$, with divisor n minus one.
$\lVert \cdot \rVert_2$	Euclidean vector norm, including the score-direction vector in the partial least squares argument.

Chapter notation

PCA	Principal component analysis: orthogonal directions retaining input variance in a declared feature scale.
PLS	Partial least squares: supervised latent directions using input-target relationships.
$\mathrm{t-SNE}$	t-distributed stochastic neighbor embedding: nonlinear neighborhood visualization, not supervised prediction.
SVD	Singular value decomposition: orthogonal real-matrix factorization into singular vectors and values.

Prediction uses numerical representations of measurements, categories, text, images or earlier model outputs. Changing coordinates can simplify optimization, expose structure or discard information the target needs. Representation learning is therefore a modeling choice, not cosmetic preprocessing. Linear projections provide a first setting in which assumptions and information losses can be stated explicitly.

The chapter connects correlated features to numerical conditioning, derives principal component analysis (PCA), and separates useful compression from useful prediction. Partial least squares (PLS) incorporates target information; t-SNE instead visualizes local neighborhoods. Fit learned transformations within the training procedure: held-out inputs can leak distributional information even when their targets remain hidden.

Essential

5.1 Representations, scale, and numerical conditioning

Changing a representation changes the geometry seen by the model. Let X have n observation rows and d feature columns. Standardization changes units; rotation changes axes; projection discards directions and may reduce dimension. Use standardization when relative deviations matter, but recognize that it can amplify unreliable low-variance measurements. Choose the resulting metric for the task, not by a universal preprocessing recipe.

Nearly duplicate columns allow very different regression coefficients to produce similar fitted values. A small data perturbation can therefore change coefficients without making all predictions unstable. Diagnose the weak directions with an SVD; compare ridge regularization, improved measurements, and additional data under realistic perturbations. These remedies answer different questions. Solve with stable factorizations rather than explicitly inverting the normal-equation matrix.

A latent coordinate summarizes variation; its name does not establish a hidden cause. Shared variation could be signal, batch effects, drift or nuisance. PCA eigenvector scores are uncorrelated on the centered training sample because of the fitted covariance eigenbasis; rotating to arbitrary orthogonal axes need not decorrelate observations. Uncorrelated also does not mean independent, nor must training covariance persist after deployment. Interpret coordinates through external evidence, repeatability and known quantities, not labels invented from a plot.

Golub and Van Loan [5.1] provide the matrix-factorization and conditioning tools behind numerical representations. Jolliffe [5.2] explains why centering and the choice between covariance and correlation PCA change the scientific meaning of the resulting components. Halko, Martinsson, and Tropp [5.8] develop randomized subspace methods for approximate matrix factorization. They provide a computational route to large-scale low-rank representations, with approximation error to assess separately from downstream prediction error.

Notation for equations (5.1), (5.2)

μ	Feature-mean column (d entries), estimated only from training data.
$\mathbf{1}$	Ones column (n entries); outer product with the mean repeats it across rows.
U	Left singular vectors (n by q), with orthonormal columns.
Σ	Diagonal nonnegative singular values (q by q), in nonincreasing order.
V	Right singular vectors (d by q), with orthonormal columns.
q	Economy-decomposition size: min(n,d).

$$X_c = X - \mathbf{1}\mu^\top \tag{5.1}$$

$$X_c = U\Sigma V^\top \tag{5.2}$$

The training mean vector is mu, X_c is the centered n by d matrix, and the singular values on the diagonal of Sigma describe variation along orthogonal directions. A small singular value indicates a weakly observed direction; zero singular values identify exact rank deficiency.

This economy SVD retains zero singular values when present; q is not necessarily the matrix rank. Transpose converts each column direction into a row.

◇ Proposition 5.1: Why weak singular directions destabilize coefficients

Notation for this result

X_c	Centered real design (n by d); full column rank after centering requires $n > d$.
U, V	Orthonormal-column matrices (n by d and d by d).
Σ	Diagonal positive singular values, largest first.
$y_c, \widehat{\beta}$	Centered targets (n entries) and least-squares coefficients (d entries).
δ	Additive zero-mean target perturbation (n entries); fixed design.
$\Delta\widehat{\beta}$	Perturbed minus original coefficient vector.
σ_d	Smallest singular value of the full-column-rank centered design.
$\Delta\widehat{y}$	Change in the n fitted training responses.

Assumptions. The centered design has full column rank, with fixed feature and target scaling. Centering has handled the intercept. Only the target changes, by a perturbation whose entries sum to zero, so the centering convention is preserved; no random-noise assumption is needed. The design remains fixed. The result concerns coefficient sensitivity, not a universal prediction-error claim.

Least squares amplifies target perturbations most strongly along the left singular vector associated with the smallest singular value. Training fitted values, by contrast, are an orthogonal projection and do not amplify Euclidean perturbations.

Proof *Advanced*

Write the economy singular-value decomposition. Because every retained singular value is positive, the normal equations can be solved in orthogonal coordinates without forming a numerical matrix inverse.

$$X_c = U\Sigma V^\top \tag{5.T1}$$

$$\widehat{\beta} = V\Sigma^{-1}U^\top y_c \tag{5.T2}$$

Resolve the target perturbation along the left singular vectors. Solving for coefficients divides each resolved component by its corresponding positive singular value. A small singular value therefore magnifies that component. The largest possible Euclidean amplification is the reciprocal of the smallest design singular value, which gives the bound below; not every perturbation attains that worst direction.

$$\Delta\widehat{\beta} = V\Sigma^{-1}U^\top \delta \tag{5.T3}$$

$$\|\Delta\widehat{\beta}\|_2 \le \frac{\|\delta\|_2}{\sigma_d} \tag{5.T4}$$

A perturbation in the weakest left singular direction attains the coefficient bound because inversion divides by the smallest singular value. To examine fitted values, multiply those changed coefficients back by the design: the singular-value multiplication cancels the division. What remains is a projection, which cannot enlarge Euclidean perturbations. Thus the same perturbation can greatly change coefficients while only modestly changing fitted values on the training design.

$$\Delta\widehat{y} = X_c\Delta\widehat{\beta} = UU^\top\delta \tag{5.T5}$$

$$\|\Delta\widehat{y}\|_2 \le \|\delta\|_2 \tag{5.T6}$$

□

Rank deficiency makes the unregularized coefficient solution nonunique; it does not make every prediction meaningless. New inputs can expose weakly constrained directions even when training fitted values remain stable. This distinction motivates regularization, stable solvers, and assessment under realistic input perturbations rather than simply deleting correlated measurements.

▶ Example: Two almost redundant measurements

Let the second measurement equal the first plus independent measurement error. At values 2 and 2.01, coefficients (8, -7) predict 16 minus 14.07, or 1.93; coefficients (3, -2) predict 6 minus 4.02, or 1.98. Both coefficient sums equal one, so the predictions agree when the error is zero. Large coefficient changes can coexist with small prediction changes near the observed relationship. Check singular values, realistic perturbations, and a ridge baseline. Do not discard the difference between measurements if that difference carries target information.

Interpretation and limits

Small singular values do not prove that measurements are meaningless. They identify weak variation in this sample, which may still matter for prediction or diagnosis.

Essential

5.2 PCA as variance maximization and reconstruction

Notation for this section

n, d	Observation count greater than one and feature count for the centered training data in this section.

PCA asks which orthogonal directions retain the most sample variance after centering. Along a unit direction v, variance is v transpose S v, with S the sample covariance matrix. A multiplier enforces unit length; setting the derivative to zero yields the eigenvector equation. The leading eigenvector maximizes variance. Later directions must also be orthogonal to earlier ones. Their eigenvalues measure variance, not semantic importance.

PCA minimizes squared reconstruction error among rank-r orthogonal linear projections. Multiply centered observations by retained loading directions to

obtain scores Z. Multiply Z by the loading transpose and restore the training mean to reconstruct. PCA thus describes dominant variation and compresses linearly under the chosen squared-error metric. Neither role optimizes target prediction, classification accuracy or fairness.

Compute PCA from the centered matrix's SVD; no covariance inverse is needed. Distinguish an unstable coordinate convention from an unstable representation. Reversing a loading and its scores leaves reconstruction unchanged. Ties allow rotated axes or, across the cutoff, a nonunique subspace. Near ties can make fitted axes unstable. Compare the retained subspace or reconstruction before treating a changed loading plot as evidence that the underlying system changed.

Jolliffe [5.2] develops PCA as a variance-maximizing projection. Eckart and Young's low-rank approximation result [5.3] supplies the algebraic foundation for its reconstruction interpretation under squared Euclidean error. Tipping and Bishop [5.7] recover principal axes through maximum likelihood in a latent-variable model. This probabilistic extension adds modeling assumptions; ordinary PCA's reconstruction identity alone does not supply an uncertainty model.

Notation for equations (5.3), (5.4)

n	Number of training observations, greater than one for this sample-covariance formula.
ℓ	Covariance eigenvalue associated with the direction; its value equals that direction's sample variance.
$\max$	Optimization over candidate directions satisfying the displayed unit-length constraint.

$$S = \frac{X_c^\top X_c}{n-1} \tag{5.3}$$

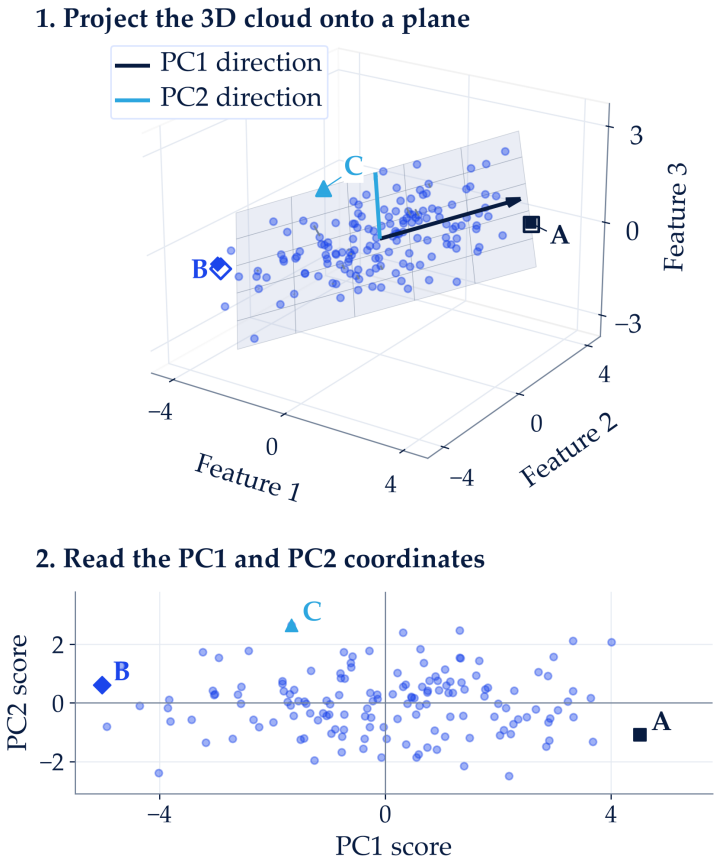


Figure 5.1. Follow A, B and C from filled 3D observations to hollow projections on the pale plane, then locate their unchanged IDs, colors and shapes below in PC-score coordinates. PCA uses 150 centered synthetic observations (seed 202705), without targets. Navy PC1 and cyan PC2 are orthonormal directions; short residual segments follow discarded PC3. The retained two-dimensional subspace preserves 99.4% of sample variance. Squared reconstruction error equals the third squared singular value, verified numerically. The lower panel changes coordinates, not observations.

$$\max_{v^\top v=1} v^\top S v \quad \Longrightarrow \quad Sv = \ell v \tag{5.4}$$

S is the sample covariance matrix. The multiplier ell becomes an eigenvalue and equals variance along its unit eigenvector. We use ell here to distinguish covariance eigenvalues from the regularization parameter lambda.

The stationarity equation also holds for nonleading eigenvectors. Selecting the largest eigenvalue gives the variance maximum; ell is not a regularization parameter.

Notation for equations (5.5), (5.6), (5.7)

V_r	Leading r right singular vectors as columns (d by r).
$\widehat{X}$	Reconstructed training matrix (n by d), including restored training means.
$\mu, \mathbf{1}$	Training mean (d entries) and ones column (n entries).
σ_j	Centered-design singular value j, ordered largest to smallest.
$j > r$	Discarded indices r+1 through min(n,d).
$\|\cdot\|_F^2$	Sum of squared reconstruction errors over every observation and feature.

$$Z = X_c V_r \tag{5.5}$$

$$\widehat{X} = Z V_r^\top + \mathbf{1}\mu^\top \tag{5.6}$$

$$\|X - \widehat{X}\|_F^2 = \sum_{j>r} \sigma_j^2 \tag{5.7}$$

V_r retains r right singular vectors. The discarded squared singular values sum to training reconstruction error. The identity assumes ordinary centered PCA with its unweighted Euclidean reconstruction metric.

This is the centered PCA reconstruction identity in the coordinates actually fitted. For standardized inputs, the identity measures standardized-coordinate error.

◇ Theorem 5.2: The global variance maximum is a leading eigenvector

Notation for this result

X_c, n, d	Centered n by d data matrix, number of observations, and number of features.
Var	Sample variance of the centered projected observations, using denominator n minus one.
$\mathcal{L}, \ell$	Constraint Lagrangian and scalar multiplier, respectively.
∇_v	Gradient with respect to the candidate direction's d coordinates.
v_j, ℓ_j	The jth unit covariance eigenvector and its eigenvalue, ordered nonincreasingly.

a_j, j Coefficient of the candidate in the eigenbasis and index from one through d.

Assumptions. There are at least two observations, all columns have been centered, and variance is measured in the fitted feature coordinates. The covariance matrix is real and symmetric; eigenvalues are ordered from largest to smallest.

A unit leading eigenvector maximizes projected sample variance. Stationarity identifies every eigenvector; the eigenvalue ordering is the additional argument that identifies a global maximum.

□ **Proof** *Advanced*

$$S = \frac{X_c^\top X_c}{n-1} \tag{5.T7}$$

$$\mathrm{Var}(X_c v) = v^\top S v \tag{5.T8}$$

Introduce a multiplier for the unit-length constraint and differentiate. Symmetry gives twice the matrix-vector product. Multiplying stationarity on the left by the direction shows that its multiplier equals captured variance.

$$\mathcal{L}(v, \ell) = v^\top S v - \ell(v^\top v - 1) \tag{5.T9}$$

$$\nabla_v \mathcal{L} = 2Sv - 2\ell v = 0, \qquad Sv = \ell v \tag{5.T10}$$

Stationarity finds every covariance eigenvector, not just the maximum. To identify the maximum, expand a unit direction in an orthonormal eigenbasis. Its squared expansion coefficients are nonnegative and sum to one, so its variance is a weighted average of eigenvalues. Such an average cannot exceed the largest eigenvalue, which identifies the optimal value.

$$v = \sum_{j=1}^{d} a_j v_j, \qquad \sum_{j=1}^{d} a_j^2 = 1 \tag{5.T11}$$

$$v^\top S v = \sum_{j=1}^{d} \ell_j a_j^2 \le \ell_1 \tag{5.T12}$$

The leading eigenvector realizes that largest variance. For the next direction, require orthogonality to those already chosen and repeat the same maximization in the remaining subspace. Tied eigenvalues leave a freedom: rotating the basis within their eigenspace changes the displayed axes but not optimality. Therefore a tied principal axis is not uniquely identified by the data. □

Principal scores are uncorrelated on the fitted sample, not generally statistically independent. Neither an eigenvector nor a variance percentage establishes a causal or physical state. Standardizing features changes the covariance matrix and therefore changes the optimization problem.

◇ Theorem 5.3: Optimal low-rank reconstruction and discarded variance

Notation for this result

W, r	Orthonormal basis (d by r) and integer retained dimension 0 through d.
P, I_r	Orthogonal projector (d by d); identity (r by r).
$\|\cdot\|_F, \mathrm{tr}$	Frobenius norm and trace (sum of diagonal entries).
σ_j	Descending singular value j, zero-padded to length d.
h_j	Squared projection length of right singular vector j onto the retained subspace.
j	Direction index 1 through d.
V_r	Leading r right singular vectors as columns.
$\widehat{X}_c$	Optimal centered reconstruction, rank at most r.
A	Candidate reconstruction (n by d), rank at most r.
P_A, I	Candidate row-space projector and identity (d by d).

Assumptions. The centered real data matrix is reconstructed using ordinary unweighted squared Euclidean error. The retained dimension is between zero and the feature count. Singular values beyond the matrix rank are zero.

Projection onto the leading principal subspace minimizes reconstruction error among orthogonal projections of that dimension. The same reconstruction is optimal among all matrices of at most that rank.

□ Proof *Advanced*

$$P = WW^\top, \qquad W^\top W = I_r \tag{5.T13}$$

$$\|X_c - X_c P\|_F^2 = \operatorname{tr}(X_c^\top X_c) - \operatorname{tr}(W^\top X_c^\top X_c W) \tag{5.T14}$$

An orthogonal projector splits each centered observation into a retained component and a perpendicular residual. Their squared lengths add, so total sum of squares equals captured sum of squares plus reconstruction error. Algebraically, symmetry and idempotence cancel the cross terms. Since the total is fixed, minimizing reconstruction error is equivalent to maximizing the captured part.

$$\operatorname{tr}(W^\top X_c^\top X_c W) = \sum_{j=1}^{d} \sigma_j^2 h_j, \quad 0 \le h_j \le 1, \quad \sum_{j=1}^{d} h_j = r \tag{5.T15}$$

Transferring weight from a smaller squared singular value to a larger one cannot decrease the sum. The maximum therefore places unit weight on the largest retained values. Their eigenvectors realize that allocation.

$$\widehat{X}_c = X_c V_r V_r^\top \tag{5.T16}$$

$$\|X_c - \widehat{X}_c\|_F^2 = \sum_{j=r+1}^{d} \sigma_j^2 \tag{5.T17}$$

The stronger claim allows any candidate matrix of the permitted rank, not just a projection. Project onto that candidate's row space, then split its reconstruction error into orthogonal row-space and perpendicular parts. Pythagoras makes their squared errors add. The projection error cannot beat the best rank-constrained projection, and the remaining term is nonnegative; a more general candidate therefore cannot improve on PCA's optimum.

Eckart and Young [5.3] concern low-rank reconstruction, not prediction of a response. Rosipal and Trejo [5.9], Sections 3-4, construct partial least-squares directions using both inputs and outputs, with centering carried into kernel space. This distinction explains why a high-variance direction need not predict the target and a supervised direction cannot be fitted using held-out responses. Compare reconstruction and prediction separately; choose the retained dimension within training-side validation rather than from a pleasing projection.

$$\|X_c - A\|_F^2 = \|X_c(I - P_A)\|_F^2 + \|X_c P_A - A\|_F^2 \tag{5.T18}$$

□

The discarded sum is a precise compression cost in the coordinates used for fitting. It is not a bound on target-prediction loss. Adding back the training feature means restores the original coordinates; a variance threshold alone cannot certify that discarded features are irrelevant.

▶ Example: A covariance matrix with a clear direction

Consider centered sample covariance with diagonal entries 4 and off-diagonal entries 3. It sends (1, 1) to (7, 7) and (1, -1) to (1, -1): the common-motion direction has eigenvalue 7 and the difference direction has eigenvalue 1. Normalize each direction by the square root of two. Of total variance 8, the first component retains 7/8, or 87.5 percent, and discards 12.5 percent. This gives optimal rank-one reconstruction in these units; a target depending on the difference may still need the discarded direction.

Interpretation and limits

A component's sign and a high explained-variance ratio do not establish a physical mechanism, causal direction, or adequate predictive information.

Algorithm 5.1. PCA by thin SVD

Algorithm references. SVD computation and numerical conditioning: Golub and Van Loan [5.1]; PCA construction: Jolliffe [5.2]; rank-constrained reconstruction: Eckart and Young [5.3].

Notation and assumptions

$X, \widetilde{X}, X_c \in \mathbb{R}^{n \times d}$	Raw, scaled and centered training matrices; n rows and d features.
$\mathcal{T}, \mu, \mathbf{1}_n$	Training-fitted scaling map, d-entry mean column, and n-entry ones column.
q, U, Σ, V	q=min(n,d); thin-SVD factors have shapes n by q, q by q, and d by q.
$\sigma_1, \dots, \sigma_q$	Nonnegative singular values, sorted largest first; diagonal entries of Sigma.
r, V_r, I_r	Retained count, first r columns of V, and r-by-r identity.
$Z \in \mathbb{R}^{n \times r}, z(x)$	Training scores and r-entry score column for a new d-entry input x.
$\widehat{X}, \|\cdot\|_F$	Reconstruction in scaled coordinates and Frobenius norm.

FitScale estimates only training-coordinate scales; use scale one for a constant feature, retaining its column. Fit any imputation on training data before this algorithm. All later rows use the saved transformations. A failed shape, finite-value or SVD check returns numerical-failure status.

Algorithm 5.1

Input: Finite training matrix and retained count.

$$n \geq 2, \quad d \geq 1, \quad 0 \leq r \leq q$$

Integer r; fixed scaling and imputation conventions.

Output: Frozen transform, mean, loadings, singular values and training scores.

$$(\mathcal{T}, \mu, V_r, \Sigma, Z)$$

Require Valid shapes, finite data and integer r in [0,q].

Training rows only. $\mathcal{T} \leftarrow \mathrm{FitScale}(X)$

$\widetilde{X} \leftarrow \mathcal{T}(X)$

$\mu \leftarrow n^{-1} \widetilde{X}^{\top} \mathbf{1}_n$

$X_c \leftarrow \widetilde{X} - \mathbf{1}_n \mu^{\top}$

Factor Sorted thin singular-value decomposition. $X_c = U \Sigma V^{\top}$

$V_r \leftarrow V_{[:,1:r]}$

$Z \leftarrow X_c V_r$

Define Frozen new-input map. $z(x) = V_r^{\top}(\mathcal{T}(x) - \mu)$

In scaled coordinates, for the residual check below.

$\widehat{X} \leftarrow Z V_r^{\top} + \mathbf{1}_n \mu^{\top}$

Check Within declared numerical tolerance. $V_r^{\top} V_r = I_r$

Check $\|\widetilde{X} - \widehat{X}\|_F^2 = \sum_{j=r+1}^{q} \sigma_j^2$

Return Fitted state and success status. $(\mathcal{T}, \mu, V_r, \Sigma, Z)$

Why this works. The empty loading matrix at r=0 reconstructs the training mean. Invert the saved scaling to recover original units; imputation is not generally invertible. Reconstruction error is measured in fitted coordinates. If all singular values vanish, explained-variance ratios are undefined. Component signs and tied-subspace bases need not be unique.

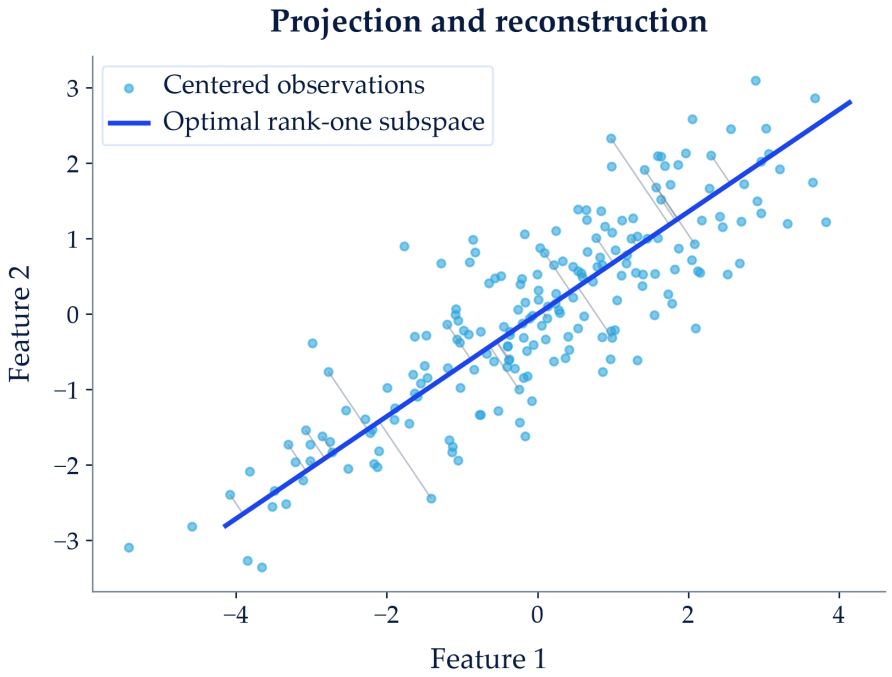


Figure 5.2. Rank-one PCA on 200 centered two-dimensional observations. The line is the first principal subspace; thin segments show discarded orthogonal residuals. The squared reconstruction residual equals the discarded squared singular value, checked numerically. This small example isolates the reconstruction geometry.

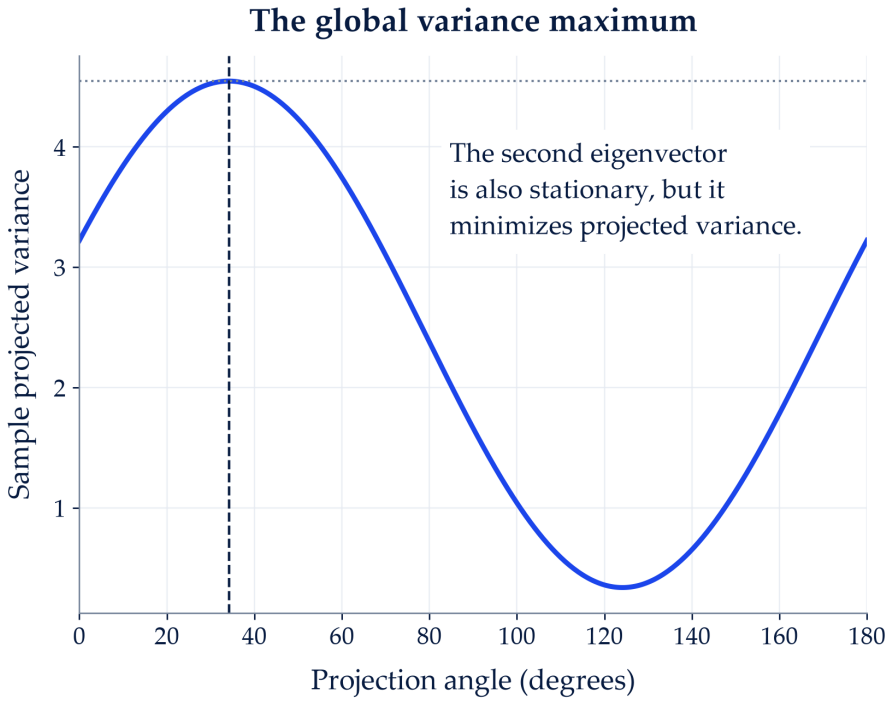


Figure 5.3. Projected sample variance is evaluated over a dense grid of direction angles for the same centered observations. Analytically, its global maximum is the largest covariance eigenvalue and occurs at the principal direction. The orthogonal second eigenvector is also stationary but minimizes, rather than maximizes, projected variance.

Essential

5.3 Choosing dimension for reconstruction or prediction

Notation for this section

u, v	Independent centered scalar random variables with unit variance in the counterexample; v here is not a projection vector.
x_1, x_2	The counterexample's two scalar measured features: ten times u and one tenth of v, respectively.

Choose the number of retained dimensions for the task, not for the plot alone. For compression, a scree plot or variance threshold suggests a rank; assess it through held-out reconstruction error, storage and feature-specific distortion. For prediction, use the downstream validation metric. A direction with little variance may carry target information, so the best compression rank need not be the best prediction rank.

Principal-component regression fits PCA, then regresses the target on retained scores. In each validation split, first estimate means, scales, and PCA directions on the training fold; then fit regression there and score the untouched fold. Compare the complete procedures, including rank and regression regularization. Refit the selected procedure on development data before one final test evaluation. Choosing rank after inspecting test error consumes that test set as validation data.

PCA ranks directions by input variation, not usefulness for a target. A low-variance measurement can predict the target accurately while a high-variance measurement is irrelevant. Scaling can change that ranking, but correlated nuisance features may still occupy the leading components. Compare principal-component regression with an all-input baseline. Ridge provides a useful contrast: it shrinks uncertain directions continuously, whereas truncating PCA removes every direction beyond the selected rank, including any target information there.

Hastie, Tibshirani, and Friedman [5.4] distinguish dimension reduction from predictive modeling. Explained variance answers a reconstruction question; held-out prediction must determine whether retaining a particular number of components is useful for a supervised task.

Notation for equation (5.8)

r, λ	Candidate component count and nonnegative regression penalty.
$\hat{r}, \widehat{\lambda}$	Component count and penalty jointly selected by cross-validation.
F, f	Validation-fold count and index 1 through F.
X_f, y_f	Held-out inputs (one column per feature) and scalar targets (one per row).
$\widehat{g}_{r,\lambda}^{(-f)}$	Complete preprocessing/PCA/regression procedure trained on all development folds except f.
MSE	Mean squared observed-versus-predicted target difference in the held-out fold.
$\arg\min$	Select the candidate pair with smallest mean validation loss.

$$\hat{r}, \widehat{\lambda} = \arg\min_{r,\lambda} \frac{1}{F} \sum_{f=1}^{F} \mathrm{MSE}\Big(y_f, \widehat{g}_{r,\lambda}^{(-f)}(X_f)\Big) \tag{5.8}$$

Each fitted procedure g includes training-fold centering, optional scaling, PCA, and regression. F is the number of validation folds. MSE is the mean squared error on a held-out fold, not the sum of squared errors.

Each fold contributes one mean loss with equal weight. With unequal fold sizes, this differs from pooling all held-out squared errors before averaging.

▶ Example: Small variation, large predictive value

Let independent centered variables u and v have unit variance. Measure x1 = 10u and x2 = 0.1v, with target v. Their population variances are 100 and 0.01. Raw rank-one PCA keeps x1 and approximately 99.99 percent of input variance. Yet independence makes its best squared-error target prediction zero, with mean squared error one. Using x2 instead gives the exact prediction 10x2 = v. A finite random sample need not have zero cross-covariance; a balanced Cartesian grid realizes exact orthogonality. This controlled counterexample separates reconstruction from prediction, not the performance of methods on a real dataset.

Interpretation and limits

Fitting PCA once before cross-validation leaks validation-distribution information. Put all learned representation steps inside the cross-validated pipeline.

Recommended

5.4 Partial least squares uses target information

PLS uses the target to choose a representation; PCA does not. A common first PLS component selects normalized input and output directions with large covariance between their projected scores. A single target has a scalar output direction; multiple targets can be combined. Because covariance depends on units, centering and scaling are model choices. PLS can therefore retain a low-variance input direction that is relevant to prediction.

After extracting one PLS component, subtract its explained contributions from the residual matrices before finding the next. Repeating the first covariance calculation on unchanged data would keep revisiting the same information, not implement multicomponent PLS. Keep the roles distinct: weights define projection directions, scores locate observations, loadings reconstruct data, and

regression coefficients predict targets. Record the implementation, normalization and deflation, because variants make different choices.

Target-aware construction can improve prediction when inputs are correlated, but it also introduces another opportunity to fit chance associations. Choose the component count through cross-validation and fit PLS separately in every training fold. The target used by a validation fold must not influence representation learning. Compare PLS against principal-component regression and a regularized all-feature baseline using the same splits and metric. No representation method is universally best, and high training covariance does not guarantee reliable out-of-sample prediction.

Wold, Sjöström, and Eriksson [5.5] describe PLS regression as a method for extracting predictive latent structure from correlated variables. Unlike PCA, its directions depend on the response, which makes fitting them within each training split essential. Rosipal and Trejo [5.9] extend target-aware PLS to nonlinear kernel representations and compare it with kernel PCR and ridge regression. Their construction is a distinct extension, not the linear PLS procedure used in this chapter; the kernel viewpoint is developed in Chapter 6.

Notation for equation (5.9)

n, d	Training count greater than one and input-feature count.
X_c	Centered inputs (n by d), after any declared training-only scaling.
Y_c	Centered targets (n by q); q here counts target variables.
w	Unit input-weight vector (d entries), defining the first input score per observation.
c	Unit output-weight vector (q entries), defining the first output score per observation.
$X_c w,\ Y_c c$	Paired score vectors (n entries) whose sample-covariance magnitude is maximized.
$\lVert\cdot\rVert$	Euclidean norm; both weight vectors have unit length.
$\max$	Joint optimization of input/output weights under the displayed normalizations.

$$\max_{\lVert w\rVert=\lVert c\rVert=1} \operatorname{Cov}(X_c w,\ Y_c c)^2 \tag{5.9}$$

X_c and Y_c are centered training inputs and outputs. The vectors w and c define the first input and output scores. Squaring covariance removes an arbitrary sign choice. Subsequent PLS components require a specified deflation algorithm.

This describes a common first-component objective, not the entire PLS algorithm. The local symbol q counts outputs, unlike the SVD dimension in Section 5.1.

◇ Derivation 5.4: The first PLS covariance direction is a singular-vector problem

Notation for this result

X_c, Y_c	Centered matrices of sizes n by d and n by q for inputs and outputs.
n, d, q	Observation count greater than one, input dimension, and output dimension.
M	The d by q sample cross-covariance matrix.
w, c	Unit input and output directions with d and q components, respectively.
s_1	Largest singular value of the sample cross-covariance matrix.
y_c	The single centered target column, with nonzero input-target cross-product.

Assumptions. Input and target matrices are centered and have the same observation rows; any scaling is fixed before this calculation. Both direction vectors have unit Euclidean norm. The cross-covariance matrix is nonzero.

The maximum squared covariance of the first pair of scores equals the largest squared singular value of the cross-covariance matrix. A corresponding left-right singular-vector pair attains it.

Read the complete supporting proof . *Advanced*

PLS uses target information that PCA ignores, so fitting it outside a validation fold leaks that fold's outcomes. A large training covariance establishes only this sample objective, not reliable prediction or unique physical interpretation.

▶ Example: Choosing a supervised latent dimension

Consider a controlled limit: six standardized features all equal a centered unit-variance nuisance u; two all equal an independent centered unit-variance target v. The covariance blocks have leading eigenvalues 6 and 2, so one-component PCR retains only u. Input-target covariances are zero for the first six features and one for the last two. The normalized first PLS weight therefore combines the last pair, producing the score square root of two times v and permitting exact prediction. With noisy measurements, compare fold-local PLS, PCR, and ridge on identical splits; this calculation is not a universal performance ranking.

Interpretation and limits

PLS fitted before splitting leaks labels directly. A compact supervised representation can overfit even when the final regression model is simple.

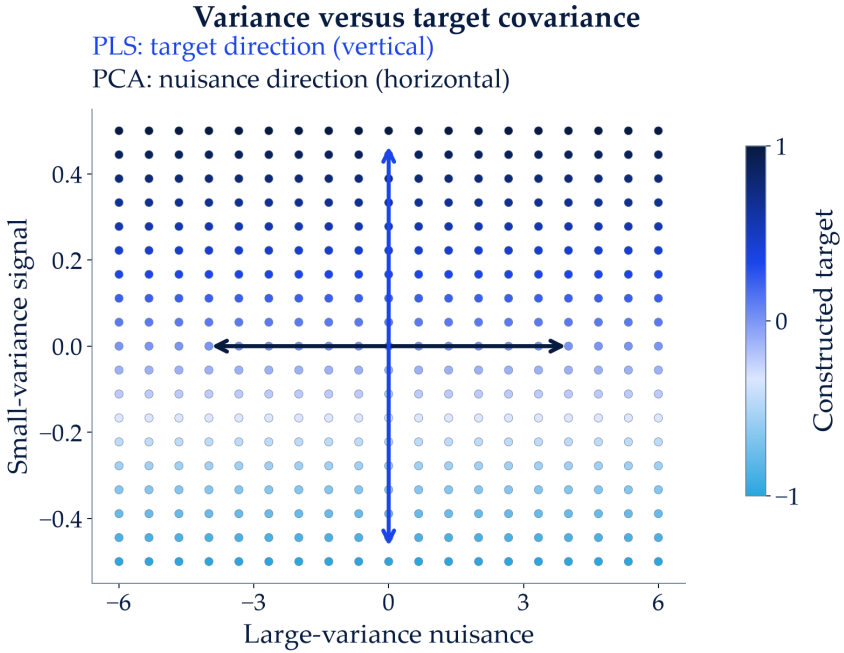


Figure 5.4. A balanced input grid makes the large-variance nuisance coordinate orthogonal to the small-variance target-bearing coordinate. PCA follows the nuisance axis; the normalized input-target cross-product identifies the first PLS direction. Colors encode the constructed target. This is an algebraic counterexample, not a benchmark.

PLS1 From Residual Directions To New Predictions

One direction maximizing input-target covariance does not yet specify how to predict with several components. We use a complete convention: single-response PLS with unit input weights and regression deflation, the PLS1 specialization of NIPALS, or nonlinear iterative partial least squares. At each stage, deflation supplies residual inputs and a residual response for the next component. With one response their cross-covariance is a vector, so its direction can be obtained directly; no iterative singular-vector calculation is needed.

Let X contain $n \geq 2$ finite training rows and $d \geq 1$ columns, and let y be its scalar-target column. Store the training feature mean μ and target mean $\bar{y}$.

Write $E_0 = X - \mathbf{1}_n \mu^T$ and $f_0 = y - \bar{y}\mathbf{1}_n$ for the centered residual matrix and residual target; $\mathbf{1}_n$ is the length-n ones column. The displayed variant centers but does not standardize. If units require scaling, fit that scaling within training, retain it for prediction, and apply the formulas in those scaled coordinates.

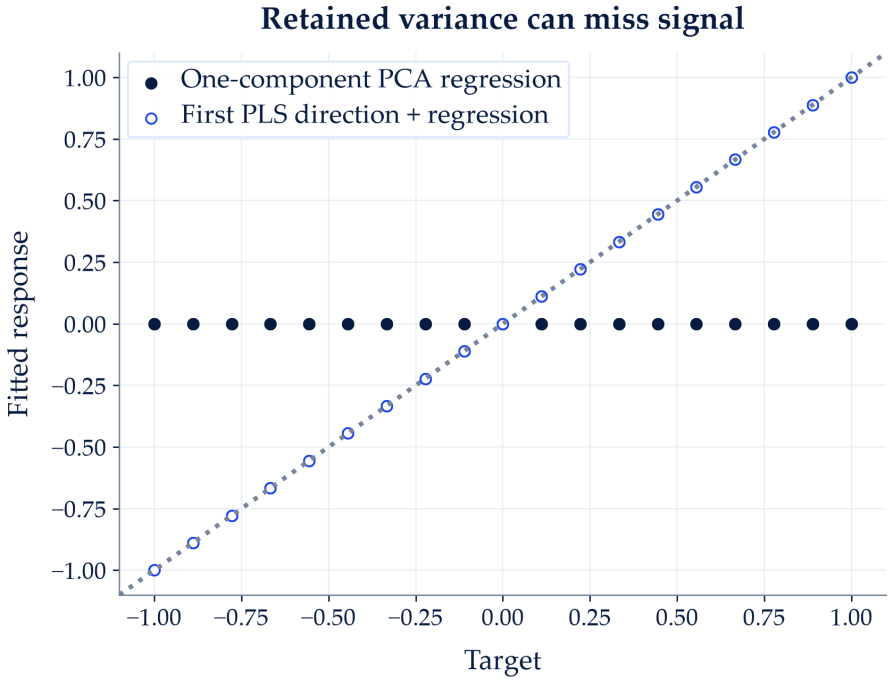


Figure 5.5. One-component PCA retains 99.3% of input variance in the orthogonal-grid example yet discards its target signal. Regression on the first PLS direction recovers the target exactly on this constructed grid. Fitted responses illustrate the different objectives; they do not estimate generalization performance.

Choose a positive maximum component count $H \leq \min(n-1, d)$ by validation. At component index h, form a unit weight w_h from the remaining input-target cross-product. A zero cross-product ends extraction: it does not justify dividing by zero or inventing another direction.

w_h Length-d input weight; the denominator is required to be positive.

E_{h-1}, f_{h-1} Residual inputs (n by d) and residual targets (length n) before component h.

$$w_h = \frac{E_{h-1}^T f_{h-1}}{\|E_{h-1}^T f_{h-1}\|_2} \tag{5.B1}$$

t_h Length-n training score column; its squared length must be positive.

$$t_h = E_{h-1} w_h \tag{5.B2}$$

Regress each residual input column and the residual target on this score. The resulting input loading p_h reconstructs features; the scalar target loading q_h predicts the target contribution. A weight chooses a score, whereas a loading maps that score back to a measured quantity.

p_h Length-d least-squares input loading for component h.

$$p_h = \frac{E_{h-1}^T t_h}{t_h^T t_h} \tag{5.B3}$$

q_h Scalar least-squares target loading for component h.

$$q_h = \frac{t_h^T f_{h-1}}{t_h^T t_h} \tag{5.B4}$$

Before extracting the next component, subtract the fitted rank-one part from both the input matrix and the target matrix. Both fitted parts use the same input score, so the remaining target is a regression residual, not a separate target-score projection. The next component is then sought in what the previous input score did not explain.

E_h Residual input matrix after removing component h.

$$E_h = E_{h-1} - t_h p_h^T \tag{5.B5}$$

f_h Residual target column after removing component h.

$$f_h = f_{h-1} - t_h q_h \tag{5.B6}$$

Suppose r nondegenerate components were retained. Collect their weights and loadings as the d-by-r matrices $W = [w_1, \dots, w_r]$ and $P = [p_1, \dots, p_r]$, their

scores as $T = [t_1, \ldots, t_r]$, and their target loadings as the length-r column q. The score directions acting on the original centered inputs are the columns of a rotation matrix R. They are not generally the residual weights in W.

G The r-by-r weight-loading product; it is upper triangular with unit diagonal in nondegenerate exact arithmetic.

$$G = P^T W \tag{5.B7}$$

R The d-by-r rotation from the original centered input coordinates to retained scores; compute it by a linear solve, not an explicit inverse.

$$R = W G^{-1} \tag{5.B8}$$

Why this recovery works. Each step projects out its score, preserving earlier zero residual-weight products. A loading has dot product one with its own weight and zero with earlier weights, so G is unit upper triangular and invertible. Thus the final residual obeys $E_r W = 0$. Multiplying the decomposition $E_0 = T P^T + E_r$ by W gives $E_0 W = TG$, and hence $T = E_0 R$. The fitted centered target is Tq, which supplies the regression coefficients.

β Length-d coefficient column in the fitted input coordinates.

$$\beta = Rq \tag{5.B9}$$

b Intercept in the same input coordinates; it restores the saved training means.

$$b = \bar{y} - \mu^T \beta \tag{5.B10}$$

$x, \widehat{y}(x)$ New input column with the original d-feature ordering, and its scalar prediction. Apply saved scaling first if scaling was used.

$$\widehat{y}(x) = b + x^T \beta \tag{5.B11}$$

PLS1 With Regression Deflation

Algorithm references. PLS regression: Wold, Sjöström and Eriksson [5.5]. The specific regression-deflation and prediction conventions agree with the official PLSRegression description [5.12]; the one-response derivation is given here explicitly.

01. Save the training means. Initialize the residual inputs and target as defined above; retain no components yet.

02. For h from 1 through H, calculate the residual input-target cross-product. Stop if it is zero; in floating point use a declared scale-aware tolerance.

03. Compute the normalized weight and score using (5.B1)-(5.B2). Reject nonfinite values or a numerically degenerate score.

04. Compute the input and target loadings using (5.B3)-(5.B4). Store the weight, score and both loadings.

05. Deflate both residuals using (5.B5)-(5.B6); then continue to the next component.

06. If no component was retained, return the training-target mean as a constant predictor.

07. Otherwise form G and solve its transposed linear system for the rotation matrix; do not form the inverse.

$$G^T R^T = W^T$$

08. Check the solve residual and conditioning; report numerical failure if unreliable. Recover beta and b using (5.B9)-(5.B10), and save all preprocessing and the retained count r.

Example: two components are genuinely needed. Consider the following four centered observations, with no feature scaling and target equal to the sum of the two inputs. Here x_1 and x_2 label the two measured coordinates, not observation indices.

Table 5.B1. A Four-Observation PLS1 Calculation

Observation	x1	x2	Target
1	-2	-1	-3
2	-1	1	0
3	1	1	2
4	2	-1	1

The first cross-product is $(10, 4)^T$, so $w_1 = (5, 2)^T/\sqrt{29}$. The first score does not capture the whole target: after its regression is subtracted, a nonzero target residual remains. The second residual direction is $w_2 = (-2, 5)^T/\sqrt{29}$. Their weights and loadings are shown separately below. Values in the table are rounded; computations use the unrounded quantities.

Table 5.B2. Residual Weights Are Not Reconstruction Loadings

Step	Weight w	Input loading p	Target loading q
1	(0.928, 0.371)	(1.012, 0.162)	1.174
2	(-0.371, 0.928)	(-0.371, 0.928)	0.557

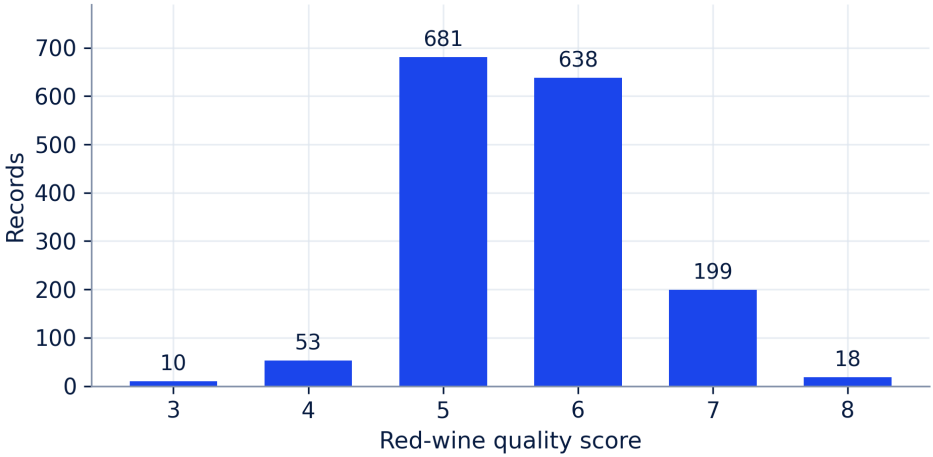
Coefficient recovery gives $\beta = (1, 1)^T$ and $b = 0$. At a new input $(1, 2)^T$, the two-component predictor returns 3; the one-component predictor instead returns $261/133$, approximately 1.962. Both residual blocks vanish after the second component in this exact constructed example. This is an algebra check, not an estimate of generalization or a reason to select two components on every dataset.

Red Wine Quality

1,599
red-wine records

11
predictors

0-10
rating scale



All 1,599 red-wine records before duplicate handling. Scores 5 and 6 dominate; the response is an ordinal rating.

The red Wine Quality file contains 1,599 Portuguese vinho verde records with 11 physicochemical predictors and an ordered sensory quality score. The repository also supplies a separate white-wine file. Quality is not cultivar identity, and duplicate measurement vectors must be handled before evaluating generalization. [5.10]

Here: only the red file is used. Removing 240 repeated predictor vectors leaves 1,359 for the case. The chart deliberately shows the original 1,599 records. The separate white file contains 4,898 records.

Cortez and colleagues / UCI, CC BY 4.0. Red-wine source file. Source / terms.

For direct library comparison, use PLSRegression(n_components=2, scale=False), whose centering and regression-deflation conventions match this example. Component signs can differ without changing predictions. Numerical verification compares coefficient-based predictions with sequential residual transformation and with the library on new inputs. The Wine Quality experiment can then be read as the same fitted method inside validation, rather than as an unexplained call to a library.

Scikit-learn developers, Cross Decomposition and PLSRegression: implementation conventions for regression deflation, rotations and scaling. Source

Recommended

5.5 t-SNE is a neighborhood visualization

Read t-SNE as a neighborhood map, not a ruler for global distances. It converts high-dimensional similarities into probabilities and seeks a low-dimensional arrangement with compatible neighborhoods. A heavy-tailed embedding distribution reduces crowding. Unlike PCA, t-SNE is nonlinear and has no ordered variance-explaining components. Its local objective does not generally preserve distances between remote islands or make visual gap sizes faithful measures of original-space separation.

The t-SNE map [5.6] fits neighborhood probabilities, not every original-space distance; island separation is not a calibrated distance. Halko, Martinsson and Tropp [5.8], Algorithms 4.1 and 4.4, use random projections and orthogonalization to approximate a matrix range. Power iterations can improve approximation but need intermediate orthogonalization for numerical stability. Check reconstruction error separately from predictive performance; neither a fast factorization nor a pleasing map validates a scientific interpretation.

Perplexity specifies a neighborhood scale, not a requested number of clusters. To test an embedding, vary defensible settings and compare which original neighbors remain neighbors, rather than which picture looks most persuasive. Record initialization, optimization settings, seed and sample composition. Trustworthiness checks one aspect of local preservation; it cannot validate a scientific grouping. Keep coloring labels out of fitting and disclose any label-informed selection.

The standard scikit-learn t-SNE estimator embeds the supplied observations and does not provide a general transform method for arbitrary future

observations. It should therefore not be dropped into a deployment pipeline as though it were PCA. Use it for exploratory analysis, error investigation, or hypothesis generation, then test hypotheses in the original feature space or with a suitable predictive representation. If preprocessing includes PCA for speed, describe that step and ensure it has not already removed the structure being investigated.

Van der Maaten and Hinton [5.6] introduced t-SNE to preserve local neighborhood relationships in a low-dimensional display. Its attractive visual separation should not be mistaken for a test of cluster membership or a faithful representation of all global distances.

Worked example: the same flowers, two different objectives

Use all 150 Iris observations and all four measurements [5.11], with each column centered and divided by its population standard deviation. This is the corrected Fisher version supplied by scikit-learn [5.13]. Fit without species labels; add species colors and shapes afterward. This is descriptive visualization, not a held-out prediction experiment.

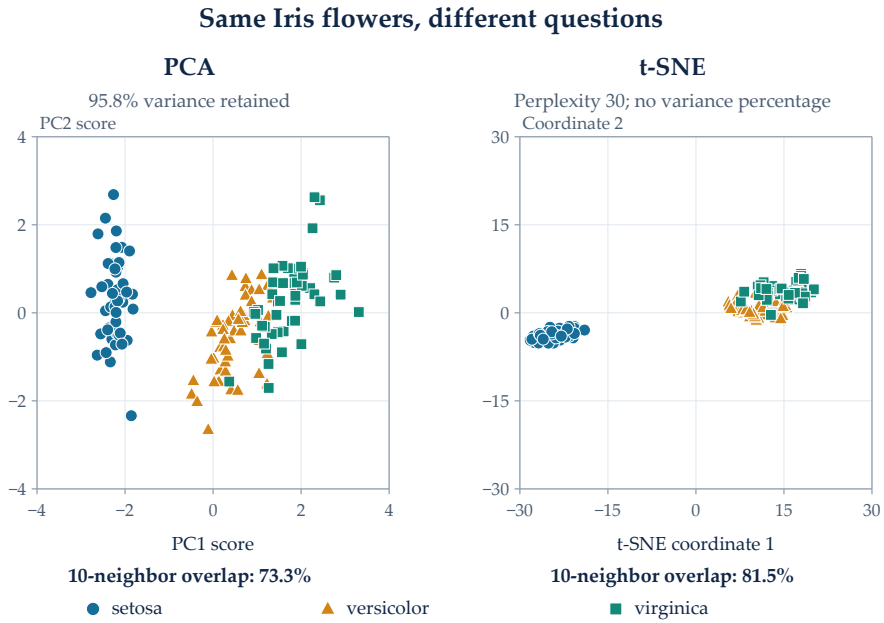


Figure 5.6. Actual PCA and t-SNE fits to the same standardized Iris measurements; species only color and shape the points. PCA retains 95.8% of input variance. The t-SNE fit uses scikit-learn 1.9.1 [5.14], exact Euclidean affinities, perplexity 30, PCA initialization, learning rate 50, early exaggeration 12, 2,000 iterations and seed 202705. Axes have different meanings and scales; t-SNE gaps and island areas are not calibrated input-space distances or densities.

For a small numerical check, find each flower's 10 nearest neighbors in the standardized four-dimensional input, then count how many remain among its 10 display neighbors. There are 150 x 10 = 1,500 directed neighbor links. PCA retains 1,099, giving 1,099/1,500 = 73.3%; this t-SNE fit retains 1,222, giving 81.5%. Exact-distance ties use observation order. This average overlap is not trustworthiness, classification accuracy or evidence of three distinct populations. Repeat across settings before drawing a scientific conclusion.

PCA versus t-SNE: a practical comparison

Question	PCA	t-SNE
Objective	Linear projection maximizing retained variance; equivalently, minimum squared reconstruction error.	Nonlinear map minimizing mismatch between input and display neighborhood probabilities.
Meaning of axes	Orthogonal scores, ordered by variance. Here PC1 explains 73.0% and PC2 22.9%.	Display coordinates have no explained-variance percentages; remote gaps and group areas are not calibrated.
Future observations	Apply the fitted centering, scaling and loading matrix.	Standard scikit-learn TSNE has no general transform for new observations [5.14].
Important choices	Feature units/scaling, retained rank and solver.	Input metric, perplexity, initialization, learning rate, iterations and seed.
This Iris run	95.8% retained variance; 73.3% mean 10-neighbor overlap.	81.5% mean 10-neighbor overlap; no corresponding retained-variance score.
Typical use	Compression, denoising, reconstruction or a validated predictive representation.	Exploratory neighborhood visualization and hypothesis generation; test conclusions in the input space.

Comparison sources: Jolliffe [5.2], van der Maaten and Hinton [5.6], and the official TSNE implementation documentation [5.14]. The Iris percentages are computed for this declared run; they are not universal rankings of the methods.

Notation for equations (5.10), (5.11)

n	Observation count whose ordered nonself pairs define both similarity distributions.
P, p_{ij}	Input-space joint pair distribution and probability of ordered pair i,j.
Q, q_{ij}	Embedding-space joint pair distribution and probability of the same ordered pair.
i, j	Observation indices 1 through n; exclude equal-index self-pairs.
z_i, z_j	Embedding coordinates, usually two-entry visualization vectors.
$\mathrm{KL}(P\|\|Q)$	Kullback-Leibler divergence from input-pair to embedding-pair probabilities.
$\log$	Natural log; zero input-probability pairs contribute zero by the limiting convention.

$\propto$ Proportionality before normalizing positive weights over all ordered nonself pairs.

$\|z_i - z_j\|^2$ Squared Euclidean embedding distance, not original-input distance.

$$\mathrm{KL}(P\|Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{5.10}$$

$$q_{ij} \propto (1 + \|z_i - z_j\|^2)^{-1} \tag{5.11}$$

P contains normalized high-dimensional pair similarities; Q contains normalized embedding similarities. The low-dimensional vectors z_i are optimized to reduce this divergence. Neighborhood preservation, not global distance preservation, drives the interpretation.

Both probability distributions sum to one over nonself pairs. The embedding coordinates are optimized; global inter-island distances are not guaranteed faithful.

▶ Example: Investigating islands without naming clusters

Three islands in a t-SNE display are a hypothesis, not three confirmed populations. Suppose six of an observation's ten original-space nearest neighbors remain among its ten display neighbors: overlap is 6/10, or 60 percent, not trustworthiness. Repeat the comparison across observations, seeds and neighborhood scales. Check original coordinates and acquisition batches: a continuous gradient can fragment, and sampling can hide a rare group. Report settings and use the display to propose a test, such as comparing subgroup errors, rather than treating appearance as confirmation.

Interpretation and limits

Do not count t-SNE islands as confirmed clusters, compare their areas as population densities, or interpret far-apart groups using literal global distances.

Table 5.1. Dimension reduction is not one problem

Method	Information used and objective	Interpretation limit
PCA	Inputs only; maximum retained variance / minimum rank-limited squared reconstruction error.	High variance can be irrelevant to the prediction target.
PLS	Inputs and targets; supervised latent directions related to cross-covariance.	Targets are part of fitting, so the representation must stay inside validation folds.
t-SNE	Input neighborhoods; nonlinear low-dimensional visualization.	Global distance, empty space and cluster area are not directly interpretable.
Frozen embedding	An existing learned representation from a separately identified model.	Its pretraining objective and domain may not match the current task.

Table 5.2. Chapter synthesis

Concept	Main conclusion	Engineering check
Scale	Changing units changes variance and distance.	Decide whether standardization matches the scientific meaning.
PCA	Orthogonal scores preserve selected variation, not all information.	State the fitted mean, retained components and reconstruction error.
Rank choice	Reconstruction and prediction can prefer different dimensions.	Choose rank using the objective relevant to the application.
Visualization	A low-dimensional map is a diagnostic, not a proof of clusters.	Check robustness and return to original-space evidence.

Case Study: Red Wine Quality: reconstruction variance versus predictive signal

Problem and data

The red Wine Quality data pair eleven physicochemical measurements with ordered sensory ratings. Ask whether a compact representation supports prediction, while distinguishing input reconstruction from target prediction. The supplied file has 1,599 records, including exact repeated measurement vectors. Remove those repeats before splitting and state that the resulting experiment weights unique vectors equally. Producer and batch identifiers are absent, so this safeguard does not establish independence across all wines or transferability to new vintages.

Experimental design

Keep the first source row for each unique feature vector after verifying consistent duplicate targets. Reserve a final test and fit standardized PCA using development inputs only. Verify reconstruction error from the omitted squared singular values, including the rank-zero mean reconstruction. Compare four complete procedures through three-fold development validation: unreduced ridge, two-component PCR, six-component PCR and two-component PLS. Refit these declared procedures on development observations, freeze the primary selection and report their predeclared final comparison with a mean reference.

Measured Reference Run

Removing 240 repeated feature vectors from 1,599 records left 1,359 equally weighted unique vectors: 1,087 development and 272 test. Three-fold validation compared all-feature ridge, principal-component regression (PCR) at two and six components, and two-component partial least squares (PLS). Scaling and representations were fitted inside each fold. Ridge was the frozen primary selection; all four predeclared procedures and the mean reference were evaluated once. Errors use rating-score units.

Table 5.3. Measured case-study evidence

Procedure	MAE	RMSE
Development mean	0.699	0.826
Ridge, all 11 features	0.529	0.677
PCR, two components	0.611	0.755

Procedure	MAE	RMSE
PCR, six components	0.545	0.697
PLS, two components	0.537	0.687

Two-component PLS outperformed two-component PCR; all-feature ridge had the lowest test RMSE. Equal dimension did not mean equal information: PLS used targets, PCA retained input variation. The reconstruction identity verifies compression, not predictive ordering.

Failure Analysis And Limits

Two-component PCR produced more error than six-component PCR or full-feature ridge in this run: variance-maximizing compression did not ensure accurate prediction. This is evidence about these fits, not proof that PLS or ridge universally wins. Removing exact duplicates changed record weights but did not reveal unknown producer or batch relationships. Squared error also treats the ordinal rating steps as equally spaced by modeling choice.

Historical saved run with seed 2027. Preserve first-occurrence deduplication, original source indices, fold-local fitting and the predeclared primary selection.

▶ Example: inspecting the real data

At PCA rank two, the saved omitted singular energy is 6,491.660. Dividing by 1,087 development rows times 11 standardized features gives reconstruction MSE about 0.543; rank-zero reconstruction has MSE 1.000. This is squared input error per matrix entry, not wine-quality prediction error. The notebook checks the identity using unrounded values.

Dataset source and documentation

Companion: Week 05, Real-Data Case Study notebook. Reproduce the declared run, then investigate a changed condition. No fixed benchmark score is promised.

A Controlled Learning Cycle

The Wine Quality case separates reconstruction quality from predictive usefulness. Week 5 notebooks test SVD boundaries, low-variance predictive directions and fold-local representation selection. The Iris laboratory traces PCA/PLS1 weights, scores, reconstruction and held-out predictions. Record one controlled change with its criterion, data roles and explanation; the limits below distinguish explained input variance from accuracy and keep t-SNE interpretation outside this laboratory.

Matched practice and simulator protocol · Packaged edition; launch instructions in the guide.

Laboratory 05 | Representations, PCA, And PLS

PCA/PLS1 weights, scores, reconstruction and held-out predictions.

Browser laboratory dataset: Iris. See its first-use portrait [5.11].

PCA and PLS1 expose feature weights, latent scores and input reconstruction while predicting measured Iris petal width from three other features.

Open Laboratory 05 | Use the live view for readable controls.

Transfer: Red Wine Quality regression → Iris petal-width regression from three other measurements. No t-SNE; explained input variance is not predictive accuracy.

Chapter Capability

Choose a representation using its intended objective, distinguish PCA from supervised PLS, and identify what a t-SNE plot cannot establish.

Week 05: separate practice, solutions and assessment

References

[5.1] Golub, G. H.; Van Loan, C. F. (2013). Matrix Computations. 4th ed., Johns Hopkins University Press. Source / DOI

[5.2] Jolliffe, I. T. (2002). Principal Component Analysis. 2nd ed., Springer. Source / DOI

[5.3] Eckart, C.; Young, G. (1936). The approximation of one matrix by another of lower rank. Psychometrika 1, 211-218. Source / DOI

[5.4] Hastie, T.; Tibshirani, R.; Friedman, J. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction. 2nd ed., Springer. Source / DOI

[5.5] Wold, S.; Sjöström, M.; Eriksson, L. (2001). PLS-regression: a basic tool of chemometrics. Chemometrics and Intelligent Laboratory Systems 58(2), 109-130. Source / DOI

[5.6] van der Maaten, L.; Hinton, G. (2008). Visualizing Data using t-SNE. Journal of Machine Learning Research 9, 2579-2605. Source / DOI

[5.7] Tipping, M. E.; Bishop, C. M. (1999). Probabilistic Principal Component Analysis. Journal of the Royal Statistical Society: Series B (Statistical Methodology) 61(3), 611-622. Source / DOI

[5.8] Halko, N.; Martinsson, P. G.; Tropp, J. A. (2011). Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions. SIAM Review 53(2), 217-288. Source / DOI

[5.9] Rosipal, R.; Trejo, L. J. (2001). Kernel Partial Least Squares Regression in Reproducing Kernel Hilbert Space. Journal of Machine Learning Research 2, 97-123. Source / DOI

[5.10] Cortez, P.; Cerdeira, A.; Almeida, F.; Matos, T.; Reis, J. (2009). Wine Quality. UCI Machine Learning Repository. Source / DOI

[5.11] Fisher, R. A. (1936). Iris. UCI Machine Learning Repository. Source / DOI

[5.12] Scikit-learn developers. Cross decomposition, Section 1.8.3: PLSRegression. Official user guide; accessed 30 October 2026. Source / DOI

[5.13] Scikit-learn developers (2026). load_iris: corrected Fisher Iris data, API reference. Version 1.9.1; accessed 31 October 2026. Source / DOI

[5.14] Scikit-learn developers (2026). TSNE: exact t-SNE implementation and parameter conventions, API reference. Version 1.9.1; accessed 31 October 2026. Source / DOI

Advanced

The First PLS Covariance Direction Is A Singular-Vector Problem

Supporting proof for Section 5.4.

Notation For This Result

X_c, Y_c	Centered matrices of sizes n by d and n by q for inputs and outputs.
n, d, q	Observation count greater than one, input dimension, and output dimension.
M	The d by q sample cross-covariance matrix.
w, c	Unit input and output directions with d and q components, respectively.
Cov	Sample covariance of paired centered score vectors with denominator n minus one.
s_1	Largest singular value of the sample cross-covariance matrix.
$\|\cdot\|_2$	Euclidean norm of a score-direction vector.
y_c	The single centered target column, with nonzero input-target cross-product.

Assumptions. Input and target matrices are centered and have the same observation rows; any scaling is fixed before this calculation. Both direction vectors have unit Euclidean norm. The cross-covariance matrix is nonzero.

The maximum squared covariance of the first pair of scores equals the largest squared singular value of the cross-covariance matrix. A corresponding left-right singular-vector pair attains it.

☐ Derivation *Advanced*

$$M = \frac{X_c^\top Y_c}{n-1} \tag{5.T19}$$

$$\mathrm{Cov}(X_c w,\, Y_c c) = w^\top M c \tag{5.T20}$$

For a fixed output direction, Cauchy-Schwarz bounds the covariance by the length of its cross-covariance image. A parallel input direction reaches that bound whenever the image is nonzero. The remaining maximization is another Rayleigh-quotient problem.

$$|w^\top M c| \leq \|Mc\|_2 \tag{5.T21}$$

$$\max_{\|c\|_2=1} c^\top M^\top M c = s_1^2 \tag{5.T22}$$

Choose a leading right singular vector for the output direction and normalize its cross-covariance image to obtain the input direction. This proves global optimality for the stated first-component problem. With one centered output, the input direction simplifies to the normalized input-target cross-product.

$$w = \frac{X_c^\top y_c}{\|X_c^\top y_c\|_2} \quad \text{when } q = 1 \tag{5.T23}$$

If the entire cross-covariance is zero, every admissible pair has zero objective and no preferred direction follows. Later components require an explicit deflation and normalization algorithm. This first-step derivation does not prove that repeated unmodified maximizations yield the full PLS regression model.

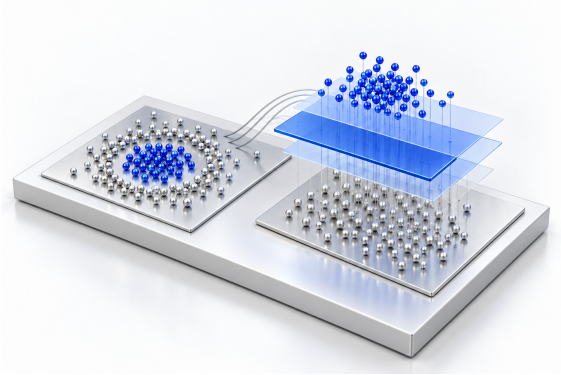
□

PLS uses target information that PCA ignores, so fitting it outside a validation fold leaks that fold's outcomes. A large training covariance establishes only this sample objective, not reliable prediction or unique physical interpretation.

Return to the chapter and its references

CHAPTER 06

Nearest Neighbors and Support Vector Machines



Reading path

Start with Essential. The other levels offer more depth.

Essential
6.1 6.2

Recommended
6.4 6.5

Advanced
6.3 See tagged proofs

Worked example Lab task

A notion of similarity is already a modeling decision. Follow that decision from nearest neighbors to margins and kernels, then compare its consequences on observations the fitting process never saw.

What You Will Be Able To Do

- Implement nearest-neighbor prediction and evaluate the effect of scaling and neighborhood size.

- Derive soft-margin SVM constraints and connect support vectors to dual variables and KKT conditions.

- Tune linear and RBF support-vector classifiers without using test data.

- Fit epsilon-insensitive support-vector regression and explain its computational tradeoffs.

Before you begin: Vector norms and inner products; Convex losses and regularization; Pipelines and cross-validation.

Trace neighbors and margin scores first; treat the kernel solver interface as an abstraction until the dual revisit.

Algorithm scope: Solver interface: Algorithm 6.1 states and verifies the kernel-SVM dual but delegates its solution to DualQP. The analytic pair example is not a general SMO implementation.

Shared chapter notation

k	Positive integer neighbor count, no greater than the available training-observation count.
C	Strictly positive coefficient on summed classification slacks (equivalently, hinge losses), also the upper bound on each dual margin multiplier. Larger C raises the cost of violations; this is not a mean-loss coefficient. Sections 6.4-6.5 specify their tuning or regression roles locally.
b	Unpenalized scalar intercept: shifts the linear decision boundary without changing its normal direction.
d	Positive number of real input-feature coordinates in the linear classification representation.
n, i	Positive training-observation count and index from one through n; unbounded observation sums include every training observation.
x_i, y_i	Training input vector and corresponding binary class label for observation i, encoded as minus one or plus one.
γ	Positive Gaussian-kernel inverse squared-distance scale in the fitted input coordinates; controls the locality of similarity and is called gamma in prose.
ϕ	Feature map into a real inner-product space, possibly infinite-dimensional.

Chapter notation

$\mathrm{kNN}, \mathrm{KNN}$	k-nearest neighbors: local prediction from a specified positive integer count of neighboring training observations.
$\mathrm{SVM}, \mathrm{SVC}$	Support vector machine and support-vector classifier; SVC names the classification estimator.
KKT	Karush-Kuhn-Tucker conditions: stationarity, primal/dual feasibility, and complementary slackness.
RBF	Radial basis function; here, the Gaussian distance-based kernel.
SVR	Support-vector regression: target-unit no-penalty residual tube, not binary margins.
$\mathrm{MAE}, \mathrm{RMSE}$	Mean absolute and root mean squared error; both in target units.

Similarity connects geometry to prediction. Nearest neighbors uses nearby observations, support vector machines control a decision margin, and kernels replace ordinary inner products with richer similarities. Their behavior depends on representation, scale and local data support; geometric elegance does not prevent failure under distribution shift or a poorly chosen metric.

We move from local averaging to maximum-margin classification, dual variables, nonlinear kernels and support-vector regression. Validation selects preprocessing and model hyperparameters as a complete pipeline. Interpret margins in model coordinates, not as physical safety certificates, and distinguish classifier scores from calibrated probabilities.

Essential

6.1 Nearest neighbors and the meaning of distance

Notation for this section

A, B	Names of the two candidate observations in the unit-scaling example, not matrices or model parameters.

K-nearest neighbors predicts by consulting stored examples close to a query. Regression averages the k neighbors' targets; classification uses class frequencies or a weighted vote. Small k stays local but is noise-sensitive; large k smooths predictions and may cross real boundaries. Select k with validation. Distance weighting is an additional choice when nearer neighbors should contribute more strongly.

Euclidean distance treats coordinate differences as commensurable after any preprocessing. A large numerical scale can dominate the distance even if the associated feature is irrelevant. Standardization often helps, but it encodes the assumption that one training standard deviation has comparable importance across features. Categorical encodings, missingness indicators, text vectors, and correlated features may require other representations or metrics. Any imputation and scaling statistics must be learned on training data, including separately within cross-validation folds.

Irrelevant coordinates can dominate high-dimensional distances, and a query far from training support is not ordinary interpolation. Evaluate both neighborhood quality and deployment cost. Brute-force prediction compares a query with stored observations; tree indexes or approximate search help only under suitable geometry and accuracy requirements. Measure memory and latency at representative sample sizes: little fitting time does not imply cheap prediction.

Cover and Hart [6.1] established foundational risk results for nearest-neighbor classification. Their analysis concerns specified asymptotic conditions; finite-sample performance still depends on representation, scale, and whether the chosen distance preserves useful neighborhoods. Weinberger and Saul [6.7] learn a Mahalanobis distance from labels rather than treating raw Euclidean distance as fixed. Such metric learning is supervised preprocessing and must therefore remain inside the training boundary.

Notation for equations (6.1), (6.2)

x, x_i	Query and indexed training input vectors (d entries), in the same preprocessed coordinates.
$\widehat{y}(x)$	Query regression prediction: mean of selected neighbors' scalar targets.
y_i	Observation-i scalar regression target.
$N_k(x)$	Indices of k nearest training observations, with a specified distance-tie rule.
$d(x, x_i)$	Euclidean query-to-training-input distance.
d, j	Feature count and index 1 through d.
x_j, x_{ij}	Query and observation-i values of feature j.

$$\widehat{y}(x) = \frac{1}{k} \sum_{i \in N_k(x)} y_i \tag{6.1}$$

$$d(x, x_i) = \sqrt{\sum_{j=1}^{d} (x_j - x_{ij})^2} \tag{6.2}$$

N_k (x) indexes the k closest training observations under the chosen metric. The expression is unweighted regression; classification instead averages class indicators. Distance-weighted versions require care when one or more distances are exactly zero.

The letter d denotes a dimension when used alone and a distance function when given arguments. This equation is unweighted regression, not classification.

▶ Example: Changing units changes the neighbors

Compare coordinate contributions before taking square roots. Query-to-A differences (1, 0.1) give squared distance 1 + 0.01 = 1.01; query-to-B differences (0.2, 0.8) give 0.04 + 0.64 = 0.68, so B is nearer. Multiply the second coordinate by ten: the squared distances become 2 and 64.04, or distances about 1.414 and 8.002. A is now nearer because the changed metric emphasizes the coordinate on which it is closer. Fit a domain-appropriate scaling rule within training data and validate the complete neighbor procedure.

Interpretation and limits

Specify distance-tie and class-vote-tie rules before evaluation. Inverse-distance weights also need a zero-distance policy for duplicate inputs; one defensible choice uses only the zero-distance neighbors. The resulting class frequencies are not guaranteed calibrated probabilities, especially with sparse or imbalanced neighborhoods.

Essential

6.2 Maximum margins and soft constraints

Notation for this section

d, n	Feature dimension and training observation count in the linear classification problem.
x, w	Real d-dimensional input and coefficient vectors; their inner product contributes to the scalar decision score.

For labels minus one and plus one, classify by the sign of w transpose x plus b. Positive rescaling of w and b leaves that boundary unchanged, so score magnitude alone is not a geometric margin. With nonzero w, the score planes minus one and plus one are separated by two divided by the norm of w. Hard-margin SVM minimizes squared coefficient norm while keeping every labeled observation on or beyond its own margin plane.

Soft-margin SVM allows overlap through nonnegative slack variables. Eliminating them gives a norm penalty plus summed hinge losses. Larger C weights training violations more heavily relative to the norm penalty; it does not guarantee better test predictions. Because the objective sums losses, translating C to a mean-loss regularization parameter depends on sample count. Check both the loss normalization and any factor of one-half before transferring settings between formulations.

A wider margin may improve generalization under appropriate assumptions, but the statement concerns geometry in the chosen feature space. It does not certify robustness to arbitrary input corruption, distribution shift, or physical hazards. Feature scaling changes that geometry. Classification also does not require maximizing margin to be useful: regularized logistic regression is a valid high-dimensional comparator. Compare predictive metrics, calibration, computation, and relevant error costs rather than declaring one classifier inherently safe or optimal for a data regime.

Cortes and Vapnik [6.2] developed support-vector networks with a margin-based objective and controlled training violations. Their formulation explains the role of the penalty parameter as a trade-off within the fitted problem, not as a direct target for test accuracy.

Notation for equation (6.3)

w	Coefficient vector (d entries) defining the linear score and plane orientation.
ξ, ξ_i	Nonnegative margin-slack vector and observation-i component.
$\|w\|^2$	Squared Euclidean coefficient norm, inversely related to squared canonical margin width.

$\min, \text{s. t.}$ Minimize over listed variables subject to every observation-wise constraint.

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i \\ \text{subject to} \quad & y_i(w^\top x_i + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0 \quad (i = 1, \dots, n). \end{aligned} \tag{6.3}$$

Slack measures a margin violation, not simply a classification error. When C is strictly positive, optimal slack is the smallest feasible value: zero at or beyond the correct margin, between zero and one for a correct point inside it, one on the decision boundary, and greater than one for a wrong label. A merely feasible solution can use larger slack. Here C penalizes the sum of these violations.

Cortes and Vapnik [6.2] analyze margin-based classification; Cover and Hart [6.1] analyze a different object, the limiting error of nearest-neighbor classification. The latter bound is asymptotic and assumes the stated sampling and regularity conditions; it does not promise that a finite one-neighbor fit has at most twice the optimal error. Use these results to ask different questions about geometry and data density. Select scaling, neighborhood size and kernel parameters through the same training-side protocol.

With positive C, optimal slack equals the hinge loss. An arbitrary feasible slack may be larger than the smallest necessary violation.

Notation for equations (6.4), (6.5)

y	Binary class label for one observation, encoded as minus one or plus one.
f	Scalar decision score for that observation, before applying a sign-based classification rule.
ℓ_{hinge}	Hinge loss, measuring the positive shortfall of the signed score from the canonical unit margin.
n	Number of training observations used to convert the summed objective into a mean-loss objective.
λ	Coefficient multiplying the squared norm when the equivalent objective uses mean hinge loss without a half factor.

$$\ell_{\text{hinge}}(y, f) = \max(0, 1 - yf) \tag{6.4}$$

$$\lambda = \frac{1}{2Cn} \tag{6.5}$$

Dividing the soft-margin objective by Cn gives mean hinge loss plus lambda times the squared coefficient norm. If the mean-loss formulation instead uses lambda divided by two, the parameter mapping changes accordingly.

Before transferring C between solvers, compare their complete objectives: sum or mean loss, sample count, and the coefficient multiplying the squared norm.

◇ Derivation 6.1: From geometric margin to the soft-margin objective

Notation for this result

x, w	Input and normal vectors in a d-dimensional real feature space.
$b, f(x)$	Scalar intercept and decision score before taking its sign.
width	Euclidean distance between the two canonical unit-score margin planes, measured in fitted feature-coordinate units.
$\|\cdot\|_2$	Euclidean vector norm.
ξ, ξ_i	Vector of slack allowances and its ith component.
ξ_i^*	Smallest feasible and therefore optimal slack when the score function is held fixed.

Assumptions. Binary labels use minus one and plus one. The coefficient vector is nonzero when interpreting geometric distance. The soft-margin penalty is strictly positive. The intercept is not regularized.

Canonical unit-score constraints turn maximum-margin fitting into norm minimization. Eliminating each optimal slack variable gives exactly hinge loss, not merely an approximation.

□ Derivation *Recommended*

Move a point along the unit normal to the separating hyperplane. Each unit of movement changes its score by the coefficient norm. The negative and positive unit-score planes are separated by a score difference, not a fixed input-space distance; dividing that score difference by the norm gives the width below. This is why smaller norms mean wider canonically scaled margins.

$$f(x) = w^\top x + b \tag{6.T1}$$

$$\mathrm{width} = \frac{2}{\|w\|_2} \tag{6.T2}$$

For strictly separable finite data, scale a separating score until its smallest signed score is one. Maximizing the resulting width is equivalent to minimizing the squared norm. If separation fails, introduce an observation-specific nonnegative allowance.

$$\min_{w,b,\xi} \ \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{n}\xi_i \tag{6.T3}$$

$$y_i f(x_i) \geq 1 - \xi_i, \qquad \xi_i \geq 0 \tag{6.T4}$$

Hold the coefficients and intercept fixed and minimize over one slack variable. Feasibility requires it to be at least zero and at least the signed-score shortfall. Its positive cost prevents choosing any excess slack, so the optimum is the larger of those two lower bounds: exactly the hinge loss. Substituting these optimal slacks yields the summed-hinge objective.

$$\xi_i^* = \max\{0, 1 - y_i f(x_i)\} \tag{6.T5}$$

$$\min_{w,b} \ \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{n}\max\{0, 1 - y_i f(x_i)\} \tag{6.T6}$$

□

Slack equals one on the decision boundary and exceeds one for a wrong-sign score. An arbitrary feasible slack may be larger than the hinge loss. A margin is geometric in the chosen feature coordinates; it is not a certificate of safety under unrestricted perturbations or distribution shift.

▶ Example: Margin violation is not the same as error

For three positive labels with scores 1.4, 0.6, and -0.2, subtract each score from one and truncate below at zero. The hinge losses are 0, 0.4, and 1.2: total 1.6. Only the last prediction is misclassified, so the classification error is 1/3. The middle point is correctly classified but still incurs margin loss. Increasing C emphasizes these losses relative to the coefficient norm and can also emphasize label noise. Select C on development validation, not by minimizing training violations alone.

Interpretation and limits

The SVM margin is measured in transformed feature coordinates. Calling it a physical safety buffer or a calibrated probability confuses different quantities.

Advanced

6.3 Duality, KKT conditions, and support vectors

Notation for this section

n, d	Training observation count and input dimension for the dual and optimality discussion.
α, β	Length-n nonnegative multiplier vectors for margin and slack-nonnegativity constraints, respectively.
w, b	Length-d coefficient vector and scalar intercept of the primal linear classifier.

The dual follows by eliminating primal variables from the Lagrangian. Assign multipliers alpha to margin constraints and beta to nonnegative slack. Stationarity expresses w as a weighted sum of labeled training vectors, makes signed alphas sum to zero, and gives alpha plus beta equal to C. Substitution leaves a concave quadratic maximization with box constraints. The positive-semidefinite kernel matrix ensures concavity, not strict concavity or unique dual coefficients.

KKT combines stationarity, primal feasibility, dual feasibility and complementary slackness; under this convex problem's usual feasibility conditions, these characterize optimality. Strictly beyond the correct margin implies alpha zero. Positive alpha identifies a support vector, not necessarily a margin-equality point: it can be inside or misclassified. Only a free multiplier, strictly between zero and C, guarantees margin equality.

This distinction matters when recovering the intercept. For free support vectors, b equals the label minus the inner product with w; averaging these expressions reduces numerical noise. When there are no free support vectors, a solver can use bounds implied by the KKT inequalities rather than blindly averaging every support vector. Predictions depend on support vectors through their weighted inner products with a query. Sparsity is dataset-dependent: difficult or heavily overlapping data can leave a large fraction of observations as support vectors.

Boyd and Vandenberghe [6.3] provide the convex-duality and KKT framework used to derive the SVM dual. Cortes and Vapnik [6.2] connect that solution to support vectors, making the active constraints interpretable rather than merely computational. Platt's sequential minimal optimization [6.8] connects the SVM dual to a practical solver by repeatedly solving small quadratic subproblems analytically. A numerical optimization method complements, rather than replaces, the feasibility and KKT checks developed here.

Notation for equation (6.6)

n, d	Training count and coordinates per input vector.
α, α_i	Dual margin-multiplier vector and nonnegative observation-i multiplier.
i, j	Training indices; every unbounded sum runs from one through n.
x_i, x_j	Training inputs (d entries each) in the dual inner product.
y_i, y_j	Indexed class labels, each minus one or plus one.
$x_i^\top x_j$	Euclidean inner product of preprocessed training inputs; linear Gram entry.
$\max, \text{s. t.}$	Maximize the concave dual subject to box bounds and signed-sum equality.

$$\begin{aligned} \max_{\alpha} \quad & \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^\top x_j \\ \text{subject to} \quad & 0 \le \alpha_i \le C, \quad \sum_i \alpha_i y_i = 0. \end{aligned} \tag{6.6}$$

The dual uses inner products between training observations. Stationarity also yields w equal to the sum of $\alpha_i \ y_i \ x_i$. The box upper bound comes from nonnegative slack and its C-weighted penalty.

The dual coefficient vector has n components. A nonzero optimal multiplier identifies a support vector; it need not lie exactly on a margin plane.

Notation for equations (6.7), (6.8)

i	Training index; both complementary-slackness equalities hold for every observation.
α_i	Multiplier of observation i's canonical-margin constraint.
w, b	Optimal linear-score coefficient vector and scalar intercept.
ξ_i	Nonnegative slack allowing canonical-margin violation.
$C - \alpha_i$	Nonnegative multiplier of observation-i slack nonnegativity.

$$\alpha_i \Big[y_i (w^\top x_i + b) - 1 + \xi_i \Big] = 0 \tag{6.7}$$

$$(C - \alpha_i) \xi_i = 0 \tag{6.8}$$

Complementary slackness links inactive constraints with zero multipliers. If zero is strictly less than α_i and α_i is strictly less than C, slack is zero and the margin constraint is active.

These are the complementary-slackness conditions, not the entire KKT system. Stationarity and primal and dual feasibility are also required.

◇ Derivation 6.2: Deriving the soft-margin dual and its box constraints

Notation for this result

n	Positive number of training observations; every unbounded observation sum ranges from one through n.
$\mathcal{L}$	Primal Lagrangian before minimizing over coefficients, intercept, and slack.
w, b, ξ_i, C	Coefficient vector, intercept, ith slack, and positive summed-slack penalty.
x_i, y_i, i	Training input, minus-one-or-plus-one label, and index over n observations.
α_i, β_i	Nonnegative multipliers for the ith margin and slack-nonnegativity constraints.
g, α	Dual objective and vector of n margin multipliers.
j	Second training index, independently ranging over all n observations.

Assumptions. There are finitely many real feature vectors and both classes are present. The penalty is strictly positive. Optimization uses the convex soft-margin primal with affine constraints. Sufficiently large positive slack gives strict feasibility, so strong duality applies.

The dual is a concave quadratic maximization with a signed-sum equality and one interval constraint per multiplier. It need not be strictly concave.

Read the complete supporting proof . *Advanced*

The feature-vector expansion follows from stationarity rather than an assumption about sparsity. Many observations can have nonzero multipliers; a solver may legitimately return a dense support-vector set. Strong duality makes the derived optimum equal to the primal optimum under the stated conditions.

◇ Proposition 6.3: What each support-vector multiplier actually implies

Notation for this result

i, x_i, y_i	Observation index, input, and minus-one-or-plus-one label.
x, w, b	Query input, coefficient vector of matching dimension, and scalar intercept of the optimal linear classifier.

$f(x) = w^\top x + b$	Optimal linear decision score, before a sign rule converts it to a class label.
α_i, ξ_i, C	Margin multiplier, nonnegative slack, and positive upper box bound.

Assumptions. Primal feasibility, dual feasibility, and all stationarity identities hold for an optimal soft-margin solution. The penalty is positive and the label encoding is minus one and plus one.

A multiplier strictly inside its box forces zero slack and an active margin constraint. At the upper bound the point may lie exactly on the correct margin, inside it, on the decision boundary, or be misclassified.

□ Proof *Advanced*

$$\alpha_i[y_i f(x_i) - 1 + \xi_i] = 0 \tag{6.T13}$$

$$(C - \alpha_i)\xi_i = 0 \tag{6.T14}$$

For a strictly interior multiplier, both complementary-slackness products constrain the solution: one forces zero slack and the other forces unit signed score. At a zero multiplier, zero slack is still required, but the margin constraint may be either tight or loose. A degenerate margin point can therefore have a zero multiplier. Being on the margin does not by itself imply being a support vector with positive multiplier.

$$0 < \alpha_i < C \implies \xi_i = 0, \quad y_i f(x_i) = 1 \tag{6.T15}$$

At the upper bound, the first product is active but the second imposes no additional slack restriction. Combining this with feasibility yields the following complete case implications.

$$\alpha_i = 0 \Rightarrow y_i f(x_i) \geq 1 \tag{6.T16}$$

$$\alpha_i = C \Rightarrow y_i f(x_i) = 1 - \xi_i \leq 1 \tag{6.T17}$$

A point strictly beyond the correct margin must therefore have zero multiplier. A free support vector yields a direct intercept estimate. If no such point exists, recover the intercept from the KKT bounds, not by averaging all support vectors indiscriminately.

$$b = y_i - w^\top x_i \qquad \text{for } 0 < \alpha_i < C \tag{6.T18}$$

□

Complementary slackness supplies these products, but feasibility and stationarity are still needed for KKT. In hard-margin fitting, a nonzero multiplier forces the margin equality used to recover the intercept. In soft-margin fitting, an upper-bound multiplier can coexist with positive slack. Therefore applying the equality to every support vector can give an incorrect intercept.

▶ Example: Recovering a linear decision function

For inputs -1 and 1 with labels -1 and 1, take C = 1. Dual multipliers 0.5 and 0.5 give signed coefficients -0.5 and 0.5. Multiplying by the inputs gives w = 1; either free support point then gives intercept zero and signed margin one. Reconstruct the scores from a linear SVC and compare them with decision_function. Recover multipliers as absolute signed coefficients and check their bounds numerically. Only multipliers strictly between zero and C justify the margin-equality intercept calculation; upper-bound support points need not have margin one.

Interpretation and limits

In soft-margin SVM, nonzero alpha does not imply an observation lies exactly on the margin. The equality is guaranteed for free support vectors, not all support vectors.

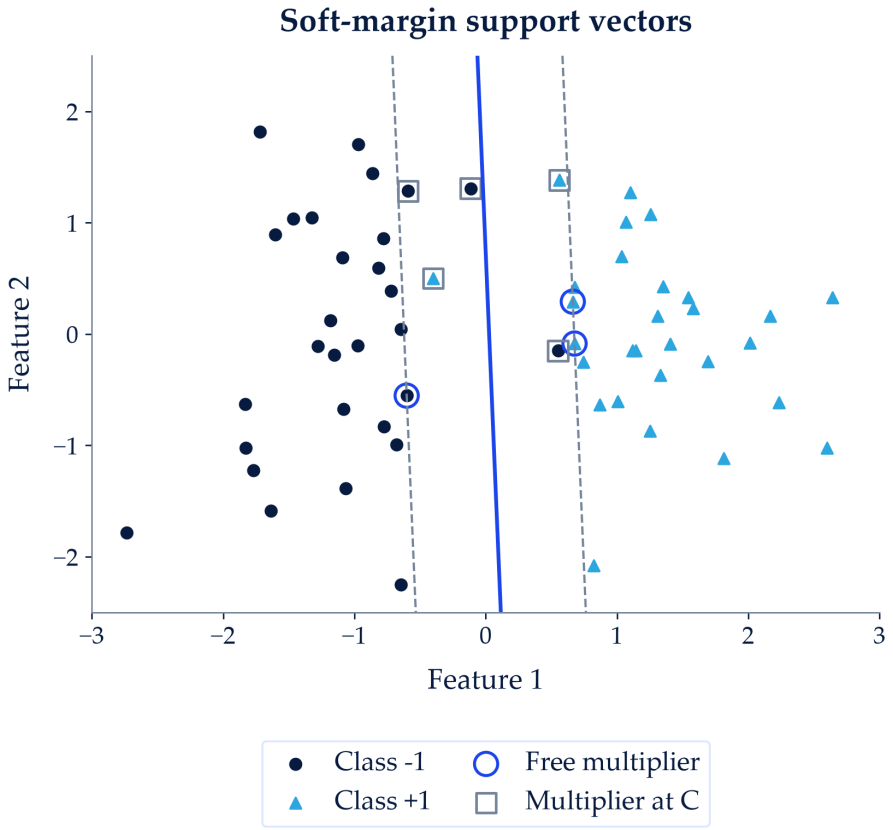


Figure 6.1. A fitted linear soft-margin SVM on two declared classes. Outlined circles identify multipliers strictly between zero and C; outlined squares identify multipliers at C. The former lie on unit margin, while the latter may violate it. The common upper bound C is 0.8.

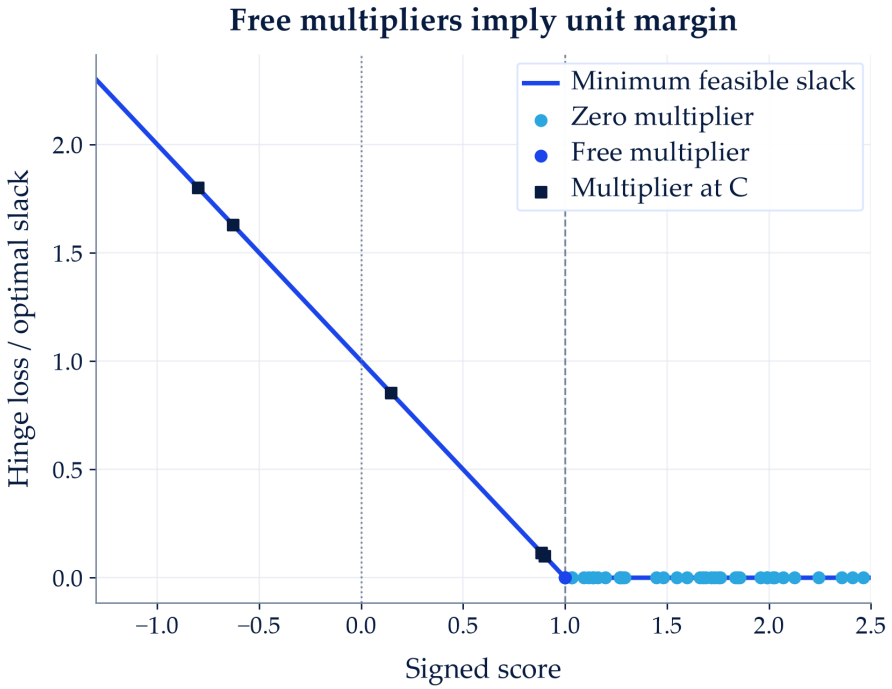


Figure 6.2. Horizontal position is signed score; height is optimal slack, max(0, 1 minus signed score). Free-multiplier circles sit at unit margin, verified numerically. Squares mark multipliers at the common upper bound C = 0.8, and cyan circles mark zero multipliers. Their distinct KKT roles must not be inferred from classification correctness alone.

Example: Preserve The Dual Constraint

Procedure references. The dual is specialized from the support-vector formulation [6.2]. The feasibility-preserving paired-update mechanism motivates SMO [6.8]; this exact two-point example is not a full SMO implementation.

A solver call can hide the mechanism of the dual. Consider exactly two scalar training inputs, $x_1 = -1$ and $x_2 = 1$, with labels $y_1 = -1$ and $y_2 = 1$. Use the linear kernel, no preprocessing, an unpenalized intercept, and the chapter's sum-of-slacks objective with $C > 0$. The equality constraint forces both dual multipliers to have the same value a. Thus moving one multiplier alone is infeasible; a paired update preserves the constraint.

$a = \alpha_1 = \alpha_2$ Common dual multiplier; the equality follows from minus alpha-one plus alpha-two equal to zero.

$D(a)$ Dual objective restricted to this feasible line; it is maximized, not minimized.

$$D(a) = 2a - 2a^2, \qquad 0 \le a \le C \tag{6.C1}$$

The input-label products are both one, so the recovered coefficient is $w = 2a$. The derivative of the dual objective is $2 - 4a$, with negative second derivative. Its unconstrained maximizer is one-half. Starting at zero, the exact feasible paired update therefore clips that value to the box; no numerical optimizer is needed for this two-observation problem.

a_* An optimal common dual multiplier.

$$a_* = \min\{C, 1/2\} \tag{6.C2}$$

w_* The corresponding optimal scalar primal coefficient.

$$w_* = 2a_* \tag{6.C3}$$

Choose $b = 0$ for the intercept. Each signed margin is then $2a_*$ and each minimal slack is $\xi = \max\{0, 1 - 2a_*\}$. The primal value is $P = w_*^2/2 + 2C\xi$. Direct substitution gives the same primal and dual value in every row below, establishing optimality by weak duality as well as the one-variable maximization. Fractions are exact.

Table 6.C1. An Exact Constrained Dual Update

Penalty C	Each multiplier	Each margin	Each slack	Primal = dual
1/4	1/4	1/2	1/2	3/8
1/2	1/2	1	0	1/2
1	1/2	1	0	1/2

Follow the same pair as C changes. At C=0.25 both multipliers reach the bound. At b=0 both labels are correct. Every b in [-0.5, 0.5] is optimal, with a zero score at either endpoint. At
one-half, the multipliers still hit the bound but the margins equal one. At one, they are free and their margin equalities determine intercept zero. Thus an upper-bound multiplier alone distinguishes neither a margin point from an interior point nor a correct prediction from a mistake.

Scope. This is a complete analytic solve of one feasible pair, not a general SMO implementation. Larger problems also need pair selection, feasible intervals when the other multipliers are fixed, degenerate-curvature handling, intercept recovery, and numerical stopping checks. Algorithm 6.1 is a solver-interface contract: its DualQP routine remains a required numerical component. This trace explains what that interface must preserve, rather than replacing it.

Recommended

6.4 Kernels and nonlinear support-vector classification

Notation for this section

C	Positive summed-slack penalty selected jointly with the kernel's distance scale.

A kernel lets the SVM use feature-space inner products without constructing the features explicitly. Replace inner products in the dual with kernel values to obtain nonlinear boundaries in the original coordinates. A similarity is not automatically a valid kernel: the usual convex formulation requires positive-semidefinite Gram matrices. Linear, polynomial and Gaussian radial basis function kernels encode different assumptions about which input relationships should count as similar.

The RBF kernel decays exponentially with squared distance, so interpret gamma only after fixing the feature units and scaling. Small gamma gives broad influence; large gamma concentrates it near individual observations. Tune gamma jointly with C on a logarithmic grid inside the fitted preprocessing pipeline. Large values can fit noise and overly restrictive values can underfit. Compare a linear baseline: the flexible boundary costs more computation but does not guarantee better prediction.

An SVC decision score is not a calibrated probability; Chapter 9 supplies a separate calibration procedure and its data boundaries. Kernel fitting can require substantial pairwise memory and computation, while prediction work grows with the retained support vectors. Benchmark both at deployment scale. A linear model or explicit kernel approximation may offer a better resource tradeoff than the exact kernel model.

Aronszajn's theory of reproducing kernels [6.4] supplies the functional-analytic basis for valid kernel constructions. In support-vector learning [6.2], a positive-semidefinite kernel allows inner products in a feature space to be evaluated without explicitly constructing its coordinates. Rahimi and Recht [6.9] approximate shift-invariant kernels with explicit randomized features so that linear learning methods can be used. The resulting approximation trades

representation size and computation against kernel-approximation error; it is not the exact kernel SVM.

Notation for equations (6.9), (6.10)

x, z	Input vectors in the same feature/preprocessing coordinates.
$k(x, z)$	Positive-semidefinite kernel: inner product of mapped inputs.
k_{RBF}	Gaussian RBF kernel, exponentially decaying in squared input distance.
$\exp$	Natural exponential; dimensionless argument when feature units and gamma agree.
$\|x - z\|^2$	Squared Euclidean distance in the representation supplied to the kernel.
$\langle \cdot, \cdot \rangle$	Kernel-feature-space inner product, also in infinite dimension.

$$k(x, z) = \langle \phi(x), \phi(z) \rangle \tag{6.9}$$

$$k_{\mathrm{RBF}}(x, z) = \exp(-\gamma \|x - z\|^2) \tag{6.10}$$

A valid kernel supplies an inner product in a feature space. For the RBF kernel gamma is positive and sets the distance scale over which similarity decays. Input standardization changes the distance and therefore the effective kernel.

Angle brackets denote the feature-space inner product; finite feature vectors can equivalently use a transpose product.

Notation for equation (6.11)

$f(x)$	Real query decision score before conversion to a label.
x, x_i	Query and observation-i training input in the fitted representation.
S	Training indices with nonzero fitted dual coefficients: support vectors.
α_i	Fitted nonnegative support-i coefficient, bounded by the soft-margin penalty.
y_i	Support-i class label: minus one or plus one.
$k(x_i, x)$	Fitted kernel between support vector i and query x.
b	Fitted scalar intercept consistent with soft-margin optimality.

$$f(x) = \sum_{i \in S} \alpha_i y_i k(x_i, x) + b \tag{6.11}$$

S indexes support vectors. The classifier uses the sign of this decision score. Its magnitude is not a probability and is not generally a literal Euclidean distance in the original input space.

The sign of the score determines the predicted class. Its magnitude is not a calibrated probability.

◇ Proposition 6.4: Why a kernel must be positive semidefinite

Notation for this result

n, i, j	Positive finite-input count and two observation indices, each ranging independently from one through n.
x_i, x_j	Inputs from the finite n-observation collection.
k, K	Kernel and Gram matrix (n by n).
a, a_i	Arbitrary real column (n entries) and its ith coefficient.
$\langle \cdot, \cdot \rangle, \lVert \cdot \rVert$	Feature-space inner product and associated norm.
x, z, γ	Real input vectors and positive inverse squared-distance scale.
$m, m!$	Nonnegative integer series index and factorial.
e^t	Natural exponential of a scalar.

Assumptions. The kernel is symmetric and real-valued. Its Gram matrix must be positive semidefinite for every finite collection of inputs, not just one observed sample. The Gaussian-kernel scale is positive and inputs are finite-dimensional real vectors.

Inner-product kernels always generate positive-semidefinite Gram matrices. This property preserves concavity of the SVM dual. The Gaussian radial-basis kernel satisfies it.

□ Proof *Advanced*

$$K_{ij} = k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle \tag{6.T19}$$

$$a^\top K a = \left\| \sum_i a_i \phi(x_i) \right\|^2 \geq 0 \tag{6.T20}$$

There are two different questions here. Feature-space inner products must produce a positive-semidefinite Gram matrix, proving necessity. Conversely, a positive-semidefinite matrix for one sample can be factored into feature coordinates for those sampled points. That finite construction does not certify a kernel everywhere: validity requires compatible positive-semidefinite matrices for every finite input set. One passing sample check cannot establish the global claim.

$$e^{-\gamma\|x-z\|_2^2} = e^{-\gamma\|x\|_2^2} e^{-\gamma\|z\|_2^2} \cdot \sum_{m=0}^{\infty} \frac{(2\gamma)^m}{m!} (x^\top z)^m \tag{6.T21}$$

Check the exponential expansion term by term. Every integer power of a dot product is an inner product of tensor-product features, hence positive semidefinite. Multiplying by a scalar feature factor at each argument also preserves that property. Nonnegative finite sums are therefore valid kernels. Finally, for any fixed finite set of inputs, entrywise convergence of the series preserves the nonnegative quadratic-form inequality, extending the argument to the exponential limit.

□

Symmetry and visually plausible similarity values are not enough to define a valid SVM kernel. Positive semidefiniteness requires every finite Gram matrix to have nonnegative quadratic forms, preserving the convex optimization structure. This tests mathematical admissibility, not predictive usefulness: a valid kernel may still encode a poor notion of similarity for the task.

▶ Example: Jointly tune locality and regularization

Reserve a two-moons test set before fitting. Three C values (0.1, 1, 10) crossed with three gamma values (0.1, 1, 10) give nine RBF candidates; three-fold validation requires 27 candidate fits. Each fit learns its own standardizer from that fold's training rows. Compare a standardized linear baseline on the same folds, select on development results, and refit before one final test assessment. A high-gamma surface can isolate observations; a low-gamma surface is broader. Plot development geometry and report the actual settings and scores rather than assuming the more flexible surface wins.

Interpretation and limits

An RBF decision score is not a probability. Enabling or fitting calibration is a separate estimation step and must not consume final test labels.

Algorithm 6.1. Kernel SVM: dual fit and verified intercept

Algorithm references. SVM formulation: Cortes and Vapnik [6.2]; convex duality and KKT conditions: Boyd and Vandenberghe [6.3]. Platt [6.8] supplies an SMO solver strategy. This procedure specifies a solver interface and verification, rather than reproducing SMO.

Notation and assumptions

$X, y_i, n, \mathcal{T}$	Training inputs, labels in {-1,+1}, sample count, and training-fitted preprocessing.
k, κ, K	Declared positive-semidefinite kernel; kappa(x,z)=k(T(x),T(z)) includes saved preprocessing; K is its n-by-n training Gram matrix.
C, τ, ε, T	Positive penalty, endpoint tolerance below C/2, verification tolerance, and positive solver budget.
α_i, g_i, b	Dual multipliers, training scores without intercept, and scalar intercept.
I, L, U	Free indices; lower-bound indices (+1 near zero or -1 near C); upper-bound indices (-1 near zero or +1 near C).
l, u, S	Maximum lower bound, minimum upper bound, and retained support-index set.
$f(x), \mathrm{sign}_0$	New-input decision score and a declared sign rule at zero.
$\mathrm{DualQP}, \mathrm{FitPrep}$	Budgeted solver for the chapter's box/equality-constrained dual; training-only preprocessing fitter.

DualQP starts from zero multipliers and solves the displayed convex dual. Every failed Require returns failure status. Endpoint sets use $\alpha \le \tau$ and $\alpha \ge C - \tau$ after feasibility checking. Keep all multipliers until intercept and optimality verification finish. Define L and U by the label/endpoint combinations above, not by the sign of a score.

Algorithm 6.1

Input: Finite training inputs containing both label classes.

Kernel, solver and prediction tolerances.

$$C > 0, \quad 0 < \tau < C/2, \quad \varepsilon > 0$$

Positive integer solver budget T.

Output: Preprocessing, support inputs/labels, multipliers, kernel, intercept and status.

01 **Require** Valid data, both classes, kernel assumptions and controls.

02 Training rows only. $\mathcal{T} \leftarrow \mathrm{FitPrep}(X)$

03 $K_{ij} \leftarrow \kappa(x_i, x_j)$

Algorithm 6.1 (continued)

$\alpha \leftarrow \mathrm{DualQP}(K, y, C; \varepsilon, T)$

Require Solver success, finite multipliers and feasible box/equality constraints.

Using the full multiplier vector. $g_i \leftarrow \sum_{j=1}^{n} \alpha_j y_j K_{ij}$

$I \leftarrow \{i : \tau < \alpha_i < C - \tau\}$

If $I \neq \varnothing$

$b \leftarrow |I|^{-1} \sum_{i \in I} (y_i - g_i)$

Else

Require Both endpoint-bound sets are nonempty.

$L \neq \varnothing, \quad U \neq \varnothing$

$l \leftarrow \max_{i \in L} (y_i - g_i)$

$u \leftarrow \min_{i \in U} (y_i - g_i)$

Require $l \leq u + \varepsilon$

$b \leftarrow (l + u)/2$

End if

Require KKT conditions and primal-dual gap pass with recovered intercept.

Prune only after the score-preservation check. $S \leftarrow \{i : \alpha_i \neq 0\}$

Define Apply saved preprocessing at inference.

$f(x) = \sum_{i \in S} \alpha_i y_i \kappa(x_i, x) + b$

Return Fitted state, sign-zero rule and verified status.

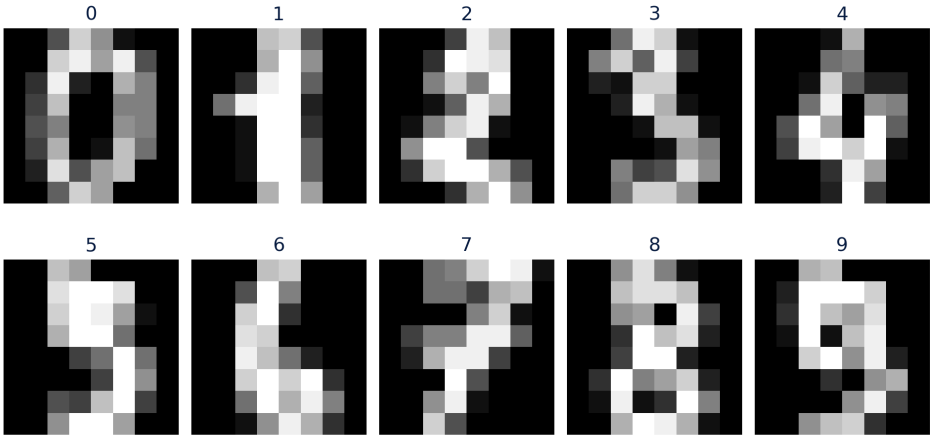
Why this works. Check feasibility, complementary slackness and a nonnegative primal-dual gap within numerical tolerance. If tiny coefficients are pruned, recheck scores under the declared prediction tolerance; otherwise retain them. Label-minus-score bounds have opposite endpoint roles for opposite labels. A decision score is not a probability.

From Binary Margins To A Multiclass SVC

Procedure references. SVC training and ties follow its official API [6.13] and user guide [6.14]. The bounded vote-plus-margin conversion is documented in the versioned implementation [6.15].

Digits — scikit-learn/UCI 8×8 copy

1,797 packaged images	**8 × 8** pixel grid	**10** digit labels



The first available example of each digit in the 1,797-image scikit-learn copy. The 8 × 8 blocks are actual pixels, not a downsampled MNIST illustration.

The scikit-learn Digits copy contains 1,797 handwritten-digit images, each represented by an 8×8 grid of values from 0 to 16. Labels run from zero to nine. These are reduced block counts from the UCI optical-digit collection, not the separate 28×28 MNIST benchmark. [6.10]

Here: load_digits contains the 1,797-image test portion of the larger 5,620-row UCI collection. Pixel values range from 0 to 16. New row splits do not establish performance on unseen writers.

Alpaydin and Kaynak / UCI, CC BY 4.0. Enlarged images from the scikit-learn copy. Source / terms.

The Digits case has ten classes, whereas the preceding dual has two. The missing step is a reduction: train binary problems and combine their outputs. Let $K \geq 3$ be the number of classes, listed in a fixed order $1, \dots, K$. One-versus-one trains one SVM on the observations of each class pair. The number M of fitted pairwise problems is therefore:

M Pairwise binary-problem count, not the number of distinct stored support inputs.

$$M = \frac{K(K-1)}{2} \tag{6.B1}$$

For a query x, write $f_{ij}(x)$ for the pairwise margin of classes $i < j$, oriented so positive favors class i and negative favors class j. This is the orientation of multiclass SVC's one-versus-one output. The formulas below assume nonzero pairwise margins; an exact zero requires the implementation's documented sign convention. Every pair casts one vote. Define $V_i(x)$ as class i's vote total.

i, j Class indices from one through K; the restricted sums include each opponent once.

$\mathbf{1}\{\cdot\}$ Indicator, equal to one when the stated condition holds and zero otherwise.

$$V_i(x) = \sum_{j<i} \mathbf{1}\{f_{ji}(x) < 0\} + \sum_{j>i} \mathbf{1}\{f_{ij}(x) > 0\} \tag{6.B2}$$

The default SVC prediction chooses a class with the largest vote count, taking the first tied class in its stored class order. Pairwise preferences can form a cycle, so a tie is not necessarily a numerical failure. The support inputs are shared across binary problems; a retained-vector count is not the sum of independent memory requirements for all pairs.

Training strategy and output shape are different controls. In scikit-learn SVC, decision_function_shape='ovo' exposes M pairwise scores; changing it to 'ovr' returns K combined class scores without training K one-versus-rest models. SVC(kernel='linear') still uses the pairwise construction. LinearSVC, or an explicit OneVsRestClassifier wrapper, is a different fitted procedure. For ten Digits classes, the SVC reduction has 45 binary problems even when its displayed decision array has only ten columns.

The combined SVC display uses votes plus a bounded correction from the signed margins. Define $c_i(x)$ as the total margin support for class i and $s_i(x)$ as the displayed class score. The following conversion describes the nonzero-margin case in the verified scikit-learn implementation.

$c_i(x)$ Signed margin sum favoring class i; it is not a probability or a common-unit geometric distance.

$$c_i(x) = \sum_{j>i} f_{ij}(x) - \sum_{j<i} f_{ji}(x) \tag{6.B3}$$

$s_i(x)$ Combined class decision score displayed by the OVR-shaped interface.

$$s_i(x) = V_i(x) + \frac{c_i(x)}{3(1 + |c_i(x)|)} \tag{6.B4}$$

The correction can break a voting tie but cannot overturn a one-vote lead. Each correction is strictly between minus one third and one third, so even opposite corrections differ by less than two thirds. A vote gap of one remains positive after adding them. The resulting scores are decision aids, not probabilities: they need not sum to one and are not calibrated.

Example: a three-class voting cycle. Name the ordered classes A, B and C, corresponding to indices 1, 2 and 3. At one illustrative query, suppose the pairwise scores are as follows. These declared scores demonstrate the decision rule; they are not a measured benchmark.

Table 6.B1. A Tie Can Survive Three Nonzero Margins

Pair	Margin	Vote
A versus B	0.2	A
A versus C	-0.4	C
B versus C	0.1	B

Each class receives one vote. The signed margin sums are -0.2, -0.1 and 0.3 for A, B and C. Equation (6.B4) gives displayed scores approximately 0.944, 0.970 and 1.077. Default vote prediction selects A by the declared class-order tie rule; selecting the largest displayed score selects C. SVC uses that latter tie-breaking behavior when break_ties=True with the OVR-shaped interface. Merely changing decision_function_shape does not enable it.

Exact zero pairwise margins need additional care: the displayed OVR conversion and the internal vote routine can use different zero-equality conventions. Use the fitted estimator's predict method with a saved tie configuration; do not silently substitute argmax of decision_function. The accompanying checks compare both interfaces on an actual fitted three-class SVC and separately verify the cycle above. Record the class order, reduction, tie setting, preprocessing and software version alongside a multiclass result. None of these decision rules supplies probability calibration.

Scikit-learn developers, SVC API: decision_function_shape and break_ties. Source

Scikit-learn developers, Support Vector Machines: multiclass training and the distinction from LinearSVC. Source

Recommended

6.5 Support-vector regression and practical model choice

Notation for this section

ϵ	Nonnegative regression-tube half-width in target units; epsilon in the prose denotes this parameter.
C	Positive coefficient multiplying summed epsilon-insensitive regression losses, under the stated norm-penalty convention.

Support-vector regression places an epsilon-insensitive tube around its prediction. Errors up to epsilon in absolute value have zero loss; errors beyond it incur a linear penalty for the excess. Fitting balances this loss against squared coefficient norm in feature space. Epsilon has target units. If the target is rescaled, adjust the relevant parameters deliberately and invert the transformation when reporting results.

Epsilon sets the tolerated residual size, C penalizes excess beyond that tube, and RBF gamma sets kernel locality. Tune them jointly on development data. Evaluate with the application's metric: epsilon-insensitive fitting need not be assessed by the same loss, and MAE, RMSE, and error slices reveal different behavior. The tolerance tube is not automatically a prediction interval.

The insensitive tube in support-vector regression [6.5] has target units; it is not a prediction interval. Weinberger and Saul [6.7], Section 3, use labels to pull target neighbors together and penalize differently labeled impostors under a positive-semidefinite metric. This is supervised fitting, not neutral preprocessing. Fit the metric inside each training fold, and keep target scaling, neighborhood size and model selection distinct.

A fair comparison includes simpler alternatives such as ridge, kNN regression, and tree-based models, using common splits and sensible preprocessing. Inspect residuals across target ranges and input regions, not just a single aggregate score. Check extrapolation behavior by plotting beyond the training domain: an RBF SVR tends toward its intercept as similarities to support vectors vanish. This may be useful regularization or an inappropriate deployment behavior. Save the complete fitted pipeline, feature order, parameter settings, and evaluation assumptions.

Smola and Schölkopf [6.5] develop support-vector regression and its epsilon-insensitive loss. The comparative treatment by Hastie, Tibshirani, and Friedman [6.6] motivates choosing classification or regression objectives according to the target, then tuning their parameters with the same evaluation discipline.

Notation for equations (6.12), (6.13)

y, f	Scalar observed target and prediction in the same target units.
ℓ_ϵ	Residual magnitude beyond the no-penalty half-width, or zero otherwise.
ϵ	Nonnegative no-penalty half-width in target units.
x_i, y_i	Training input vector and scalar regression target.
w, b	Feature-space coefficients and scalar intercept defining the optimized regressor.
$f(x)$	Predicted response: feature-space coefficient/input inner product plus intercept.
C	Positive coefficient on summed epsilon-insensitive training losses.
$\|w\|^2$	Squared norm in the chosen coefficient space, including the implied kernel space.

$$\ell_\epsilon(y, f) = \max(0, |y - f| - \epsilon) \tag{6.12}$$

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_i \ell_\epsilon(y_i, f(x_i)) \tag{6.13}$$

The objective uses summed epsilon-insensitive losses. Epsilon is the half-width of the no-penalty tube in target units, not a probability level. For kernel SVR the norm is in the implied feature space.

Here f without an argument denotes a scalar response, whereas f with an argument denotes the prediction function. The tube is not a prediction interval.

◇ Derivation 6.5: Support-vector regression needs two residual constraints

Notation for this result

w, b	Feature-space coefficients and scalar intercept.
C, ξ_i, ξ_i^*	Positive penalty and nonnegative upper/lower residual slacks; star labels a family, not optimality.
i, n	Training index 1 through n and observation count.
x_i, y_i, f_i	Training input, real target, and observation-i fitted response.
ϵ, ϕ	Nonnegative tube half-width and kernel feature map.
α_i, α_i^*	Nonnegative upper/lower residual multipliers; star marks the second family.
d_i	Signed difference of observation-i multipliers.

α, α^*, j	Multiplier vectors (n entries each) and second training index.
$k(x_i, x_j)$	Feature-space inner product provided by the regression kernel.

Assumptions. Targets are real-valued. The tube half-width is nonnegative and measured in target units; the penalty is positive. A positive-semidefinite kernel supplies the feature-space inner product. The intercept is unpenalized.

Epsilon-insensitive regression penalizes both sides of a residual tube. Its dual uses a difference of two multiplier vectors, rather than class labels multiplying one vector.

☐ Derivation *Advanced*

$$\min_{w,b,\xi,\xi^*} \frac{1}{2}\|w\|^2 + C\sum_i(\xi_i + \xi_i^*) \tag{6.T22}$$

$$y_i - f_i \le \epsilon + \xi_i, \qquad f_i - y_i \le \epsilon + \xi_i^* \tag{6.T23}$$

$$f_i = \langle w, \phi(x_i)\rangle + b \tag{6.T24}$$

Fix the prediction and minimize both nonnegative slacks: together they leave only absolute error beyond the tube half-width. Then assign one multiplier family to each residual inequality. The opposite inequality directions explain why their difference enters coefficient stationarity. Intercept stationarity makes those differences sum to zero; slack stationarity bounds each family by the penalty. These are distinct constraints, not interchangeable multiplier sums.

$$d_i = \alpha_i - \alpha_i^* \tag{6.T25}$$

$$w = \sum_i d_i \phi(x_i) \tag{6.T26}$$

$$\sum_i d_i = 0, \qquad 0 \le \alpha_i, \alpha_i^* \le C \tag{6.T27}$$

SVR needs separate multipliers for exceeding the tube above and below. Their difference determines the signed prediction coefficients. Eliminating the primal variables leaves a concave quadratic, a target-weighted linear term and a penalty for tube width. Unlike binary SVM, these signs arise from residual sides rather than supplied class labels; the classification dual cannot simply be reused unchanged.

$$\max_{\alpha,\alpha^*} \sum_i y_i d_i - \epsilon \sum_i (\alpha_i + \alpha_i^*) - \frac{1}{2} \sum_{i,j} d_i d_j k(x_i, x_j) \tag{6.T28}$$

□

A residual inside the tube incurs no training penalty; this is an optimization rule, not a probability statement about future targets. Choosing the tube width therefore supplies neither a confidence interval nor a guaranteed prediction-coverage fraction. Also keep the dual interpretations separate: signed SVR coefficients arise from regression constraints, not classification multipliers multiplied by binary labels.

▶ Example: Interpreting an epsilon tube

With epsilon 0.2 target units, absolute errors 0.1, 0.2, and 0.8 give tube losses 0, 0, and 0.6: subtract 0.2 and truncate at zero. Mean tube loss is 0.2, whereas MAE is 1.1/3, approximately 0.367 target units. Zero tube loss therefore does not mean zero error. For a noisy sine experiment, tune C, gamma, and epsilon on development data, then report held-out MAE or RMSE. Inspect extrapolation separately. Neither the chosen tolerance nor these three residuals establishes future coverage.

Interpretation and limits

The SVR epsilon tube defines a training loss tolerance, not a calibrated predictive interval. Its width has no automatic coverage guarantee.

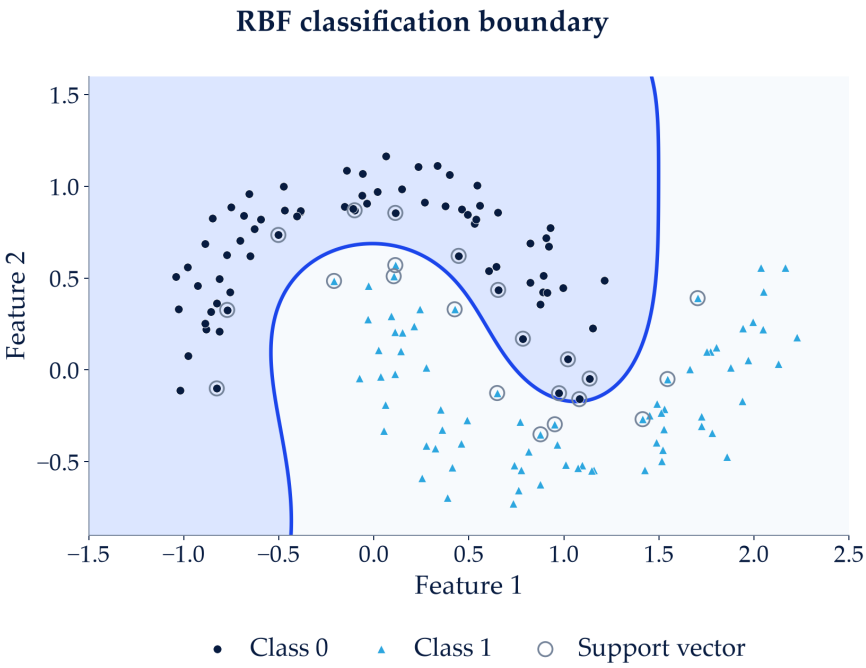


Figure 6.3. An RBF support-vector classifier fitted to two curved synthetic groups. The blue contour is decision score zero; outlined observations are support vectors. Kernel bandwidth and penalty determine the fitted geometry. The boundary is not a calibrated probability contour.

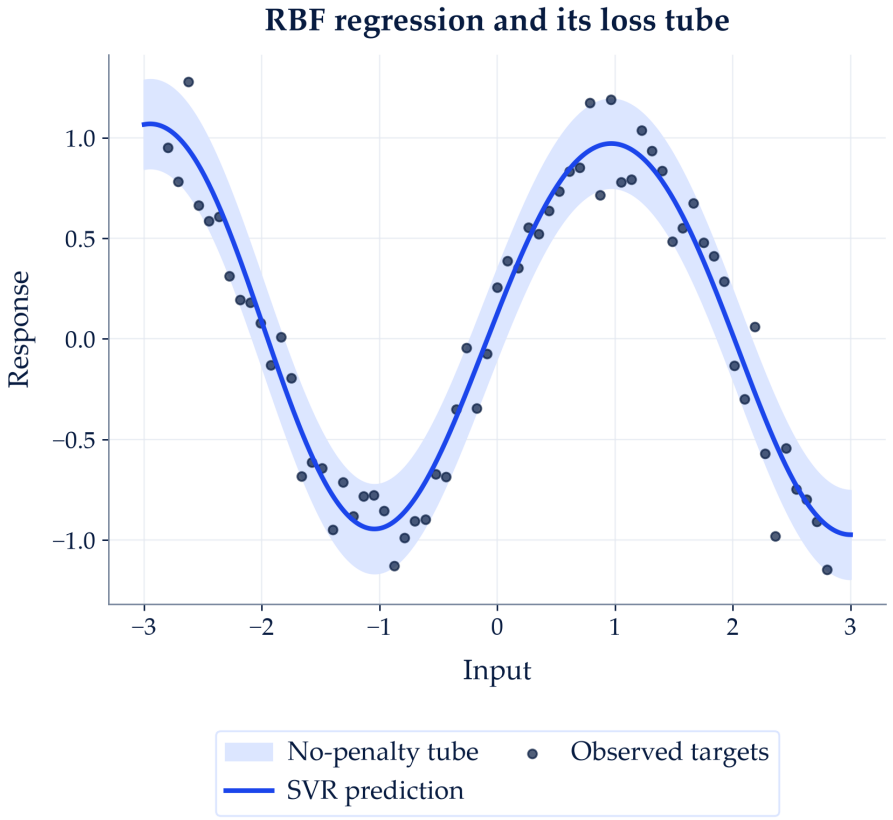


Figure 6.4. A fitted RBF support-vector regressor with epsilon-insensitive half-width 0.22 target units. The pale tube is the no-penalty region of the training loss around the prediction. It is not a prediction interval and carries no probabilistic coverage guarantee.

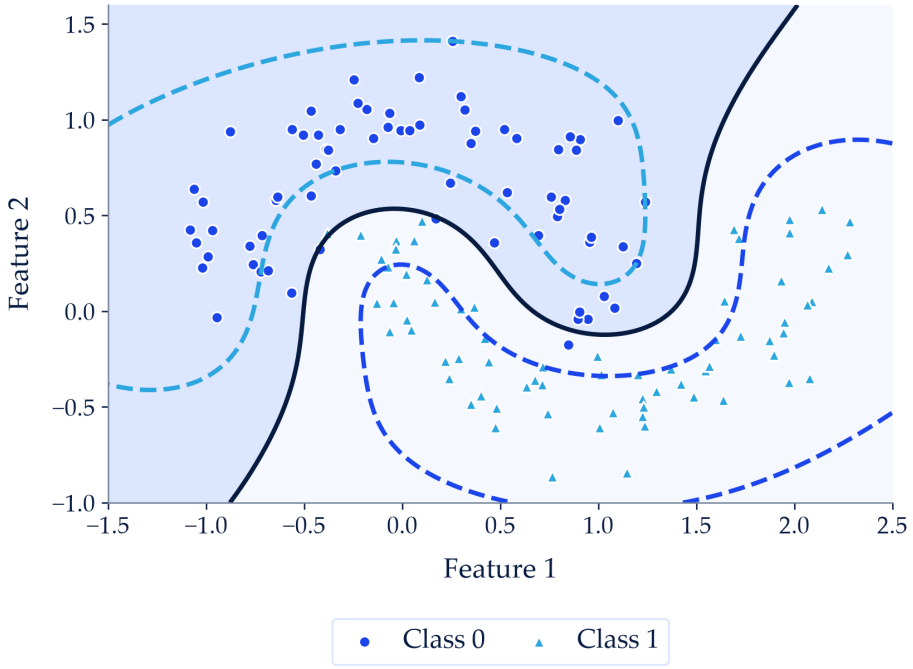


Figure 6.5. Actual RBF SVC on a synthetic two-moon dataset. The solid contour is decision score zero; dashed contours show scores -1 and +1. These are model margins, not calibrated probabilities or physical safety envelopes.

Table 6.1. Distance and margin methods

Method	How prediction is constructed	Critical design choice
KNN	Aggregate targets of nearby training cases.	Scaling, distance, neighbor count and inference cost.
Linear SVM	A regularized separating hyperplane based on margins.	Penalty C, class weighting and feature scale.
Kernel SVM	A linear margin in an implicit feature space.	Kernel validity, bandwidth and support-vector count.
SVR	A regularized predictor with an epsilon-insensitive residual region.	Target scale, tube width epsilon and the penalty convention.

Table 6.2. Chapter synthesis

Concept	Main conclusion	Engineering check
Geometry	A distance function defines what counts as similar.	Transformations must be fitted on the training fold.
Margins	Support vectors determine the fitted boundary.	Inspect slack violations and validate C.
Kernel	A positive-semidefinite Gram matrix supports the stated dual construction.	Do not substitute an arbitrary similarity without checking assumptions.
Regression tube	Residuals inside the tube incur no epsilon-insensitive loss.	Express epsilon in the target units.

Case Study: Digits: verify a margin, then compare local and kernel prediction

Problem and data

The Digits collection provides 1,797 small handwritten images with ten labels. Each image has sixty-four intensity coordinates, making geometry and computational cost concrete. Before comparing classifiers, reserve a stratified final test and declare that the evaluation concerns held-out benchmark rows. Writer identities are unavailable, so it does not establish performance on unseen writers. The task is to connect a verifiable binary margin calculation to a bounded multiclass comparison, not to report a universal image-recognition champion.

Experimental design

Using development zeros and ones only, standardize inputs and fit a binary linear SVM. Reconstruct its coefficient vector from signed dual coefficients and check free-support-vector margins separately from upper-bound coefficients. Then cross-validate four complete multiclass pipelines: five-neighbor classification, linear SVM and two declared RBF parameter pairs. Freeze the validation-selected primary procedure, refit the predeclared comparisons on development rows and evaluate once on the final test alongside a majority reference, retaining row predictions and reference counts.

Measured Reference Run

The stratified split reserved 360 of 1,797 images, leaving 1,437 development rows. Three-fold validation compared standardized five-neighbor classification, linear SVM with C=1, and two RBF pairs: (C=1, gamma=0.01) and (C=10, gamma=0.001). The last pair was the frozen primary selection. Every predeclared procedure was refitted on development rows before final scoring.

Table 6.3. Measured case-study evidence

Procedure	Test errors / 360	Accuracy	Retained vectors
Majority	324	0.1	0
Five neighbors	11	0.969	1437
Linear SVM	12	0.967	430
RBF (1, 0.01)	10	0.972	723
RBF (10, 0.001)	9	0.975	612

The selected RBF rule made the fewest observed errors. Separately, binary development score reconstruction verified linear-SVM algebra to machine precision, checking margin equality only for free support vectors. That agreement validates the calculation, not the RBF model's predictive advantage.

Failure Analysis And Limits

The selected model misclassified source rows 794, 129 and 1553, all true eights, as ones. These are inspectable failures, not proof of a single visual cause. Retained vectors mean stored examples for neighbors but support inputs for SVMs, not equivalent memory or operation costs. Missing writer identifiers prevent an unseen-writer claim.

Historical saved run with seed 2027. Preserve row order and software versions for distance ties. This 8-by-8 dataset is not MNIST; the four settings do not rank whole algorithm families.

▶ Example: inspecting the real data

Observed intensities 5 and 13 differ by 8 counts, or 0.5 after division by 16. A common positive scale rescales all Euclidean distances uniformly and preserves neighbor order. The case instead fits coordinate-wise standardization, which reweights pixels differently and can change neighbors. These transformations therefore encode different geometry.

Dataset source and documentation

Companion: Week 06, Real-Data Case Study notebook. Reproduce the declared run, then investigate a changed condition. No fixed benchmark score is promised.

MNIST handwritten digits

70,000 source images	**28 × 28** pixel grid	**10** digit labels



The first training example of each digit in MNIST. Each original image contains 28 × 28 grayscale pixels.

MNIST supplies 60,000 training and 10,000 test grayscale images of handwritten digits, each 28×28 pixels. The Chapter 12 case uses only threes and eights to study an explicit scalar-output network. That restricted experiment is not a ten-class MNIST benchmark and does not measure unrestricted handwriting recognition. [6.12]

Here: this source is distinct from the 8 × 8 Digits collection. The Chapter 12 case selects only threes and eights: 1,600 fit, 400 validation and 600 original-test images. The gallery shows the full ten-label vocabulary.

MNIST images © Yann LeCun and Corinna Cortes, CC BY-SA 3.0. This selected, enlarged montage is supplied under the same license. Source / terms.

From Error Counts To Paired Evidence

Procedure reference. Paired-index resampling and percentile intervals follow the SciPy bootstrap documentation [6.16]. Apply the pairing separately within the predeclared class strata, as specified here.

The Digits comparison above reports small differences in error counts. We now inspect their paired structure without changing any fitted rule. This is a retrospective teaching analysis of the already disclosed test, not a newly preregistered confirmation. The comparator is the fixed linear SVM with C=1, not whichever candidate looks worst on the test.

$i = 1, \dots, m$ Index of a test row; m=360.

$y_i, \hat{y}_i^A, \hat{y}_i^B$ True label and frozen predictions from selected RBF model A and linear comparator B.

d_i Paired zero-one loss difference. Negative values favor model A; the indicator is one when its condition holds.

$$d_i = \mathbf{1}\{\hat{y}_i^A \neq y_i\} - \mathbf{1}\{\hat{y}_i^B \neq y_i\} \tag{6.B20}$$

$\widehat{\Delta}$ Observed difference in misclassification rates; it estimates a conditional comparison of these fitted procedures, not all possible training runs.

$$\widehat{\Delta} = \frac{1}{m}\sum_{i=1}^{m} d_i \tag{6.B21}$$

Table 6.B2. Where The Frozen Classifiers Disagree

RBF model A	Linear model B	Rows
Correct	Correct	345
Correct	Wrong	6
Wrong	Correct	3
Wrong	Wrong	6

Model A makes 9 errors and B makes 12; the estimated difference is -0.008, or -0.833 percentage points. Cases on which both fail do not change the paired difference. Resampling the two prediction columns separately would destroy this information.

For an illustrative stratified paired bootstrap, draw test rows with replacement within each true class, retaining its original count and both model outcomes

together. Recompute the mean difference for each of 20,000 draws. With generator seed 20271006, the 2.5th and 97.5th percentiles are [-0.025, 0.008]. This interval contains zero; these observations do not establish a clear advantage in the conditional row-level comparison.

The resampling treats rows as independent within classes, holds the test class mixture fixed and does not retrain the models. Unknown writer grouping threatens that sampling assumption. The interval is approximate, discrete and conditional on the fitted models; it omits training-set variability and cannot certify performance on new writers. To plan a fresh confirmatory study, freeze the comparator and analysis before collecting an independent writer-identified test, then resample at the writer level when appropriate.

Implementation reference: SciPy bootstrap documentation describes paired resampling. Here it is applied separately within the pre-existing class strata. The executable companion records every test-row prediction and the bootstrap seed. Source

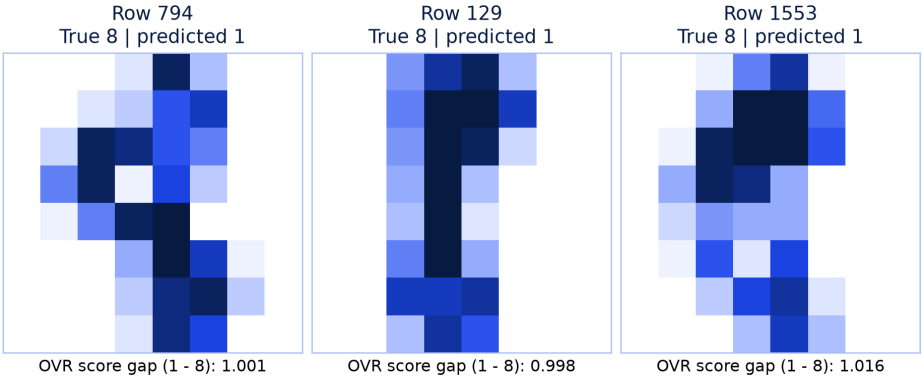


Figure 6.B1. The three Digits errors identified in the preceding case study, from four test eights predicted as ones. These are selected failure examples, not a representative sample. The score gap compares one-versus-rest presentation scores, not calibrated probabilities; its scale must not be interpreted as confidence.

One hypothesis is that downsampling has weakened the loops that distinguish an eight from a one. Another is that scaling and the fitted similarity geometry place these examples near the wrong class. The images alone do not distinguish the explanations. A new development-only study could collect higher-resolution views and compare fixed preprocessing rules on writer-separated folds. A nearest-training-image inspection may describe model geometry, but neither inspection authorizes further tuning on these three test errors.

A Controlled Learning Cycle

The Digits case compares complete pipelines on matched observations. Week 6 notebooks connect scaling, dual reconstruction, support-vector roles and SVR units to that comparison. The two-species Iris laboratory makes KNN votes and linear/RBF C-SVC mechanisms visible. Record a baseline and one change with metric, data roles and explanation; its binary scope below excludes multiclass voting and SVR, and decision scores remain distinct from probabilities.

Matched practice and simulator protocol · Packaged edition; launch instructions in the guide.

Laboratory 06 | Nearest Neighbors and Support Vector Machines

KNN votes and genuine linear/RBF C-SVC mechanisms.

Browser laboratory dataset: Iris. See its first-use portrait [6.11].

Two measured Iris species make SVM decision regions, score contours and support vectors inspectable; KNN reveals the actual neighbors behind a vote.

Open Laboratory 06 | Use the live view for readable controls.

Transfer: Ten-class Digits classification → Two Iris species, two petal measurements. No multiclass voting or SVR.

Chapter Capability

Build a scaled neighbor or kernel pipeline, tune it inside the training boundary, and explain support vectors and regularization in the fitted model.

Week 06: separate practice, solutions and assessment

References

[6.1] Cover, T. M.; Hart, P. E. (1967). Nearest Neighbor Pattern Classification. IEEE Transactions on Information Theory 13(1), 21-27. Source / DOI

[6.2] Cortes, C.; Vapnik, V. (1995). Support-vector networks. Machine Learning 20, 273-297. Source / DOI

[6.3] Boyd, S.; Vandenberghe, L. (2004). Convex Optimization. Cambridge University Press. Source / DOI

[6.4] Aronszajn, N. (1950). Theory of reproducing kernels. Transactions of the American Mathematical Society 68, 337-404. Source / DOI

[6.5] Smola, A. J.; Schölkopf, B. (2004). A tutorial on support vector regression. Statistics and Computing 14, 199-222. Source / DOI

[6.6] Hastie, T.; Tibshirani, R.; Friedman, J. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction. 2nd ed., Springer. Source / DOI

[6.7] Weinberger, K. Q.; Saul, L. K. (2009). Distance Metric Learning for Large Margin Nearest Neighbor Classification. Journal of Machine Learning Research 10, 207-244. Source / DOI

[6.8] Platt, J. C. (1998). Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines. Microsoft Research Technical Report MSR-TR-98-14. Source / DOI

[6.9] Rahimi, A.; Recht, B. (2007). Random Features for Large-Scale Kernel Machines. Advances in Neural Information Processing Systems 20. Source / DOI

[6.10] Alpaydin, E.; Kaynak, C. (1998). Optical Recognition of Handwritten Digits. UCI Machine Learning Repository. Source / DOI

[6.11] Fisher, R. A. (1936). Iris. UCI Machine Learning Repository. Source / DOI

[6.12] LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324. Source / DOI

[6.13] Scikit-learn developers. SVC: decision_function_shape and break_ties. Official API documentation; accessed 30 October 2026. Source / DOI

[6.14] Scikit-learn developers. Support Vector Machines, Section 1.4.1: Classification and multiclass classification. Official user guide; accessed 30 October 2026. Source / DOI

[6.15] Scikit-learn developers. _ovr_decision_function. sklearn/utils/multiclass.py, release 1.7.2; official source code; accessed 30 October 2026. Source / DOI

[6.16] SciPy developers. scipy.stats.bootstrap: paired resampling and percentile intervals. Official API documentation; accessed 30 October 2026. Source / DOI

Advanced

Deriving The Soft-Margin Dual And Its Box Constraints

Supporting proof for Section 6.3.

Notation For This Result

n	Positive number of training observations; every unbounded observation sum ranges from one through n.
$\mathcal{L}$	Primal Lagrangian before minimizing over coefficients, intercept, and slack.
w, b, ξ_i, C	Coefficient vector, intercept, ith slack, and positive summed-slack penalty.
x_i, y_i, i	Training input, minus-one-or-plus-one label, and index over n observations.
α_i, β_i	Nonnegative multipliers for the ith margin and slack-nonnegativity constraints.
g, α	Dual objective and vector of n margin multipliers.
j	Second training index, independently ranging over all n observations.

Assumptions. There are finitely many real feature vectors and both classes are present. The penalty is strictly positive. Optimization uses the convex soft-margin primal with affine constraints. Sufficiently large positive slack gives strict feasibility, so strong duality applies.

The dual is a concave quadratic maximization with a signed-sum equality and one interval constraint per multiplier. It need not be strictly concave.

☐ Derivation *Advanced*

Attach a nonnegative multiplier to each margin inequality and another to each slack nonnegativity inequality, writing constraints with a nonpositive left-hand side.

$$\begin{aligned}\mathcal{L} = \frac{1}{2}\|w\|_2^2 + C\sum_i \xi_i \\ + \sum_i \alpha_i(1 - \xi_i - y_i(w^\top x_i + b)) \\ - \sum_i \beta_i \xi_i.\end{aligned} \tag{6.T7}$$

A finite infimum in the unconstrained intercept and slack coordinates requires their linear coefficients to vanish. Differentiating the quadratic coefficient term gives the remaining stationarity identity.

$$w = \sum_i \alpha_i y_i x_i \tag{6.T8}$$

$$\sum_i \alpha_i y_i = 0 \tag{6.T9}$$

$$C - \alpha_i - \beta_i = 0 \tag{6.T10}$$

The last identity and multiplier nonnegativity are exactly the lower and upper bounds on each margin multiplier. Substitute stationarity into the Lagrangian: the coefficient-dependent terms become minus half the squared norm of the weighted input sum, while slack and intercept terms vanish.

$$g(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i^\top x_j \tag{6.T11}$$

$$\max_\alpha \; g(\alpha) \quad \text{subject to} \quad 0 \le \alpha_i \le C, \quad \sum_i \alpha_i y_i = 0 \tag{6.T12}$$

The quadratic matrix is a Gram matrix after multiplying each feature vector by its label, hence is positive semidefinite. Its negative is the dual Hessian, proving concavity. Linear dependence may leave flat directions, so distinct optimal multiplier vectors can represent the same coefficient vector. The hard-margin derivation drops the slack terms and removes the upper box bound.

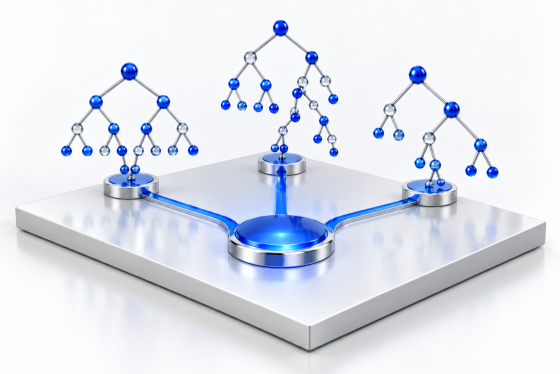
□

The feature-vector expansion follows from stationarity rather than an assumption about sparsity. Many observations can have nonzero multipliers; a solver may legitimately return a dense support-vector set. Strong duality makes the derived optimum equal to the primal optimum under the stated conditions.

Return to the chapter and its references

CHAPTER 07

Trees And Ensemble Learning



Reading path

Start with Essential. The other levels offer more depth.

Essential
7.1 7.2 7.3 7.4

Recommended
7.5

Advanced
Proofs 7.3 7.4 7.5

Worked example Lab task

Combining models is useful only when it improves the errors that remain. Distinguish averaging unstable learners from sequentially correcting a loss, and check the extra complexity against a simple baseline.

What You Will Be Able To Do

- Compute regression and classification split gains and explain greedy tree growth.
- Control tree complexity through leaf size, depth, and pruning selected on development data.
- Distinguish bagging, random forests, and gradient boosting by their training procedures.
- Implement probability voting and leakage-free out-of-fold stacking.

Before you begin: Supervised losses and validation; Variance, correlation, and averages; Basic probability and gradient descent.

Calculate one split and follow the pruning trace before studying additive fitting and out-of-fold stacking.

Algorithm scope: Mixed implementation scope: the growth/pruning fixture is explicit; Algorithm 7.1 supplies safeguarded boosting orchestration and delegates FitTree. Stacking likewise delegates base and meta-estimators.

Chapter notation

CART	Classification and regression trees: recursive fitted partitions with declared node losses and split rules.

$\mathrm{MSE}, \mathrm{SSE}$	Mean squared error and sum of squared errors, respectively; the normalization distinguishes them.

Decision trees partition input space and assign a simple prediction within each region, naturally representing interactions and threshold-like behavior. Their usefulness on tabular data does not make finite-sample partitions exact laws or a split near a known threshold causal evidence. Smooth models can also approximate sharp changes and deserve empirical comparison.

Because single trees can be unstable, we examine several combinations: bagging averages resampled fits, random forests randomize features, boosting improves a loss sequentially, and voting or stacking combines possibly different model families. Their data flows differ. In particular, stacking must train on predictions made without fitting the corresponding targets.

- *Essential*

7.1 Regression trees as recursive partitions

Notation for this section

j	Candidate split-feature index, ranging over the available input columns.
s	Scalar split threshold, in the selected feature's units.

A regression tree routes each input through axis-aligned questions, such as whether feature j is at most threshold s. Its leaf predicts the training-target mean under squared error. The piecewise-constant prediction captures interactions because earlier answers determine which later questions apply. Standardization is usually unnecessary: positive affine unit changes preserve candidate partitions, except for numerical ties and rounding.

To score a regression-tree split, first predict each child's target mean, then sum the two children's squared residuals. Subtract this from the parent's error to obtain the gain. Candidate thresholds lie between ordered distinct feature values. Choosing the largest immediate gain is greedy: it does not optimize the eventual tree, and a small data change can alter the entire later hierarchy.

A tree's flexibility does not guarantee good extrapolation. A new point beyond the observed range still reaches a leaf and receives a value based on targets already in that leaf. Ordinary regression trees therefore produce bounded piecewise-constant predictions rather than extending a learned linear trend. This behavior may be desirable for some tasks and wrong for others. Compare interpolation and extrapolation separately, inspect leaf sample sizes, and ensure that any preprocessing or categorical encoding used before the tree is also fitted within training folds.

Breiman, Friedman, Olshen, and Stone [7.1] developed CART as recursive partitioning for prediction. A regression tree therefore approximates the

response by locally constant values on data-dependent regions, rather than fitting one global linear relationship.

Notation for equations (7.1), (7.2)

x, x_i	Query and training-observation-i input vectors, respectively.
$f(x)$	Piecewise-constant regression-tree prediction in the query's leaf.
M, m	Terminal-leaf count and leaf index from one through M.
R_m	Input region routed to leaf m by the tree's splits.
n_m	Positive number of training inputs in leaf m.
c_m	Scalar leaf prediction: mean training target under unweighted squared loss.
y_i	Observed scalar regression target of training observation i.
$\mathbf{1}(x \in R_m)$	One if the query lies in leaf m; zero otherwise.
$i : x_i \in R_m$	Training indices with inputs in the specified leaf.

$$f(x) = \sum_{m=1}^{M} c_m \mathbf{1}(x \in R_m) \tag{7.1}$$

$$c_m = \frac{1}{n_m} \sum_{i:x_i \in R_m} y_i \tag{7.2}$$

The regions R_m are the tree leaves; n_m counts the training observations in leaf m. For ordinary unweighted squared-error fitting, c_m is that leaf's mean target. Leaf regions partition the input space, including locations beyond observed data support.

Using the explicit count n_m avoids confusing the number of training observations with the geometric size of a leaf region.

Notation for equation (7.3)

R	Parent input region containing the training observations considered for splitting.
R_L, R_R	Disjoint left- and right-child regions of the candidate split.
SSE	Regional sum of squared target residuals about the fitted regional mean.
Δ	Training squared-error reduction from replacing the parent prediction by the two child means.

$$\Delta = \mathrm{SSE}(R) - \mathrm{SSE}(R_L) - \mathrm{SSE}(R_R) \tag{7.3}$$

The gain Delta is the decrease in summed squared error for a parent region R split into left and right children. Dividing consistently by the parent sample count gives an equivalent mean-error comparison at that node.

Every SSE uses the observations in its own region. This formula assumes ordinary unweighted squared-error tree fitting.

◇ Proposition 7.1: Optimal leaf predictions and the exact gain from a split

Notation for this result

I, i, n_I	Nonempty leaf index set, observation index, and number of indices in that set.
$y_i, c, \bar{y}_I$	Observed target, candidate leaf prediction, and arithmetic mean target in the leaf.
Δ	Parent residual sum of squares minus the sum of the two child residual sums of squares.
n_L, n_R	Positive numbers of observations in the left and right children.
$\bar{y}_L, \bar{y}_R, \bar{y}$	Mean targets in the left child, right child, and parent, respectively.

Assumptions. Targets are real-valued and fitting uses unweighted squared error. The parent and both proposed child nodes contain at least one observation. The split partitions the parent's observation indices.

A leaf mean uniquely minimizes its squared-error loss. The reduction from replacing one parent mean by two child means equals a sample-size-weighted squared difference between the child means.

□ Proof *Recommended*

Compare any constant leaf prediction with the sample mean. Add and subtract that mean inside each residual before squaring. The cross term disappears because the centered targets sum to zero. The remaining difference is the sample count times the squared displacement from the mean: nonnegative, and zero only at the mean. This proves the leaf prediction rather than merely defining it.

$$\sum_{i \in I} (y_i - c)^2 = \sum_{i \in I} (y_i - \bar{y}_I)^2 + n_I (c - \bar{y}_I)^2 \tag{7.T1}$$

Within each child, write each target's displacement from the parent mean as its displacement from the child mean plus the child-to-parent mean difference. The cross term sums to zero because residuals around a child mean sum to zero. Adding the children therefore separates within-child error from the error explained by their different means.

$$\Delta = n_L(\bar{y}_L - \bar{y})^2 + n_R(\bar{y}_R - \bar{y})^2 \tag{7.T2}$$

$$\bar{y} = \frac{n_L\bar{y}_L + n_R\bar{y}_R}{n_L + n_R} \tag{7.T3}$$

$$\Delta = \frac{n_L n_R}{n_L + n_R}(\bar{y}_L - \bar{y}_R)^2 \tag{7.T4}$$

Substituting the parent weighted mean yields the final identity. Therefore, for fixed parent data, searching every admissible threshold for the largest gain is exactly the same as minimizing total within-child squared error. Neither calculation compares all possible future tree structures.

□

Read split gain as a combination of mean separation and supporting sample sizes. A single extreme target can make a child mean very different and yield a large training gain, without establishing a dependable group. Minimum leaf size and validation therefore remain important. Maximizing this immediate split gain solves only the current choice, not the global search over completed trees.

▶ Example: Calculate a split by hand

For ordered targets 1, 1, 5, 5, the parent mean is 3 and squared error is 4 + 4 + 4 + 4 = 16. Splitting after the second observation gives child means 1 and 5, zero child error, and gain 16. Splitting after the first leaves mean 11/3 in the other child, with squared error 32/3; its gain is only 16/3, approximately 5.333. The middle split therefore wins this local comparison. Its perfect training fit does not show that further splits on noisy data will generalize.

Interpretation and limits

A learned split is not automatically a causal threshold. Its location depends on data, available features, sampling variation, and the greedy search procedure.

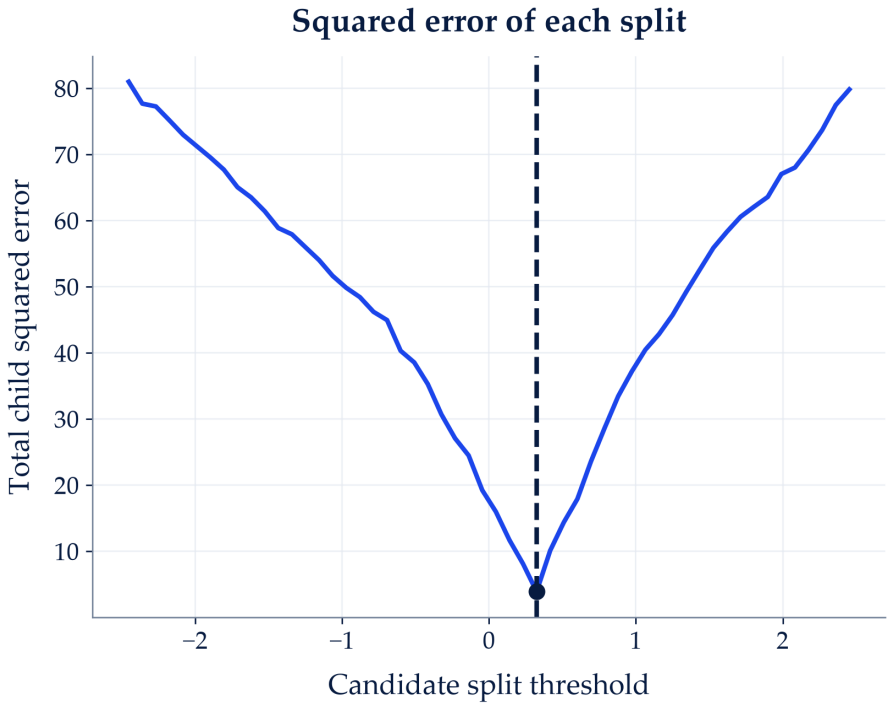


Figure 7.1. Every midpoint between ordered observations is evaluated using the sum of within-child squared errors and independently fitted child means. The minimum agrees numerically with a depth-one CART model. The selected threshold is a local split optimum, not a globally optimal tree.

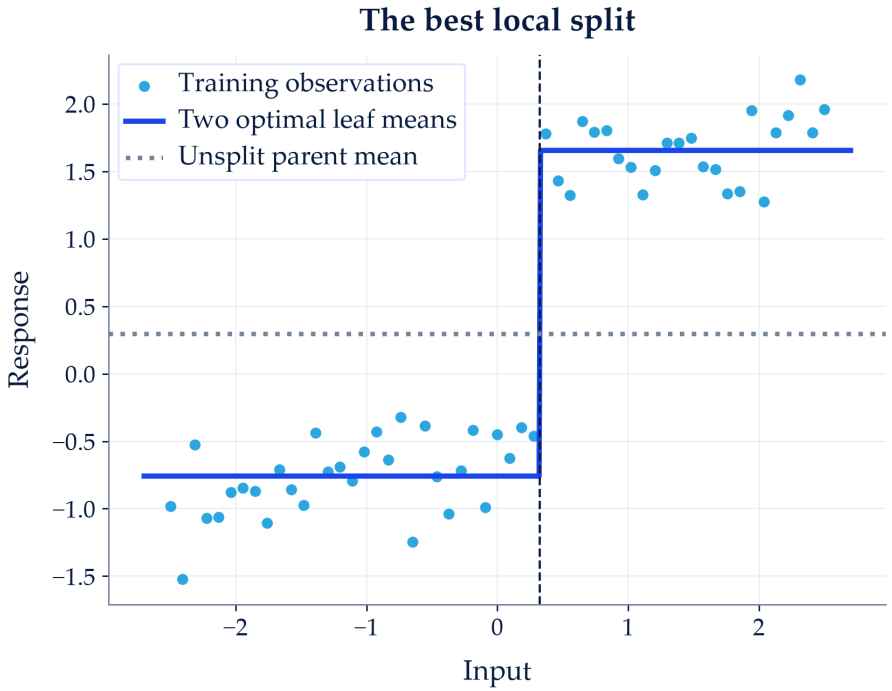


Figure 7.2. The fitted depth-one regression tree uses the optimal mean in each child. The dashed vertical line is the selected split; the dotted horizontal line is the unsplit parent mean. Its error reduction equals the count-weighted squared difference between child means, checked numerically.

Essential

7.2 Classification impurity and complexity control

A classification split should make its children purer without ignoring how many observations each contains. Gini impurity is one minus summed squared class proportions. Entropy is the negative sum of proportion times log proportion, with zero-probability terms treated as zero. Both vanish in a pure node. Score a split by parent impurity minus the sample-size-weighted child average; otherwise a tiny pure child can receive undue credit.

Depth, leaf size, and pruning control different aspects of a tree. Maximum depth limits path length; minimum samples per leaf limits small terminal groups; cost-complexity pruning trades fit against leaf count. Select them by development validation and report the resulting structure. Excessive depth can memorize noise, but overly shallow trees miss interactions and large leaves

blur meaningful boundaries. Neither strongest regularization nor best training fit is the selection rule.

Leaf class proportions provide convenient probability estimates, but small leaves can produce extreme values unsupported by much data. Calibration requires separate evaluation. Class weights change the fitting objective and may help represent asymmetric priorities, yet weighted leaf values may not match deployment prevalence. Impurity-based feature importances can favor features with many candidate split points and can divide importance unpredictably among correlated inputs. Use held-out inspection and domain checks before presenting tree importance as an effect size or causal explanation.

The CART framework [7.1] connects classification splits to impurity reduction and tree size to pruning. The greedy split criterion determines a local improvement; controlling the final tree's complexity remains a separate part of constructing the predictor.

Notation for equations (7.4), (7.5)

R	Training-node region used to estimate local class proportions.
c, p_c	Class index and within-node class fraction; fractions sum to one.
$G(R)$	Node Gini impurity; zero for a single-class node.
$H(R)$	Node class-label entropy, measuring its class mixture.
$\sum_c$	Sum over all task classes, including those absent from the node.
$\log$	Consistently chosen natural logarithm; zero-log-zero terms are zero.

$$G(R) = 1 - \sum_c p_c^2 \tag{7.4}$$

$$H(R) = -\sum_c p_c \log p_c \tag{7.5}$$

p_c is the class proportion within a node. Gini and entropy measure different forms of class mixing, not test accuracy. A consistent logarithm base only rescales entropy.

These are unweighted class proportions. With training sample weights, use their normalized weighted counterparts consistently.

Notation for equation (7.6)

R, R_L, R_R	Parent region and its disjoint left- and right-child regions.
n	Parent training count, equal to the two child counts summed.
n_L, n_R	Left- and right-child training counts, respectively.
I	Node impurity, e.g. Gini or entropy, used consistently for parent and children.

ΔI	Reduction in impurity, weighting each child by its fraction of parent observations.

$$\Delta I = I(R) - \frac{n_L}{n} I(R_L) - \frac{n_R}{n} I(R_R) \tag{7.6}$$

The parent contains n observations; its left and right children contain n_L and n_R, with n=n_L +n_R . Each child impurity is weighted by its share of the parent observations. This avoids confusing a parent count with the right-child count.

This notation reserves the subscript R for the right child; the parent count has no subscript. Weighted fitting replaces counts with total sample weights.

Notation for equation (7.7)

T	Candidate tree: branching structure and fitted terminal predictions.
$\mathrm{Cost}(T)$	Penalized training criterion balancing fitting loss and terminal-leaf count.
$\mathrm{TrainingLoss}(T)$	Specified training loss or impurity aggregate, normalized consistently with the pruning penalty.
a	Nonnegative penalty per terminal leaf, selected by development validation.
$\mathrm{Leaves}(T)$	Terminal-node set; bars count its nodes.

$$\mathrm{Cost}(T) = \mathrm{TrainingLoss}(T) + a\,|\,\mathrm{Leaves}(T)| \tag{7.7}$$

Cost-complexity pruning uses a nonnegative penalty a for tree size. The fitting-loss normalization and penalty convention are implementation-specific; select the pruning parameter using validation.

This is a generic cost-complexity form. Library implementations can differ in their definition and normalization of training loss.

◇ Derivation 7.2: Gini and entropy are optimized leaf prediction losses

Notation for this result

I_R, n_R	Node observation-index set and its positive sample count; R labels the node here.
i, y_i	Node observation index and its observed class label.
K, k	Number of possible classes and class index from one through K.
p_k, q_k	Empirical fraction in class k and candidate predicted probability for that class.

$\mathbf{1}[\cdot]$	Indicator equal to one when the bracketed statement holds, otherwise zero.
$\log$	Natural logarithm, giving node entropy and log loss in natural-log units.

Assumptions. The node contains finitely many unweighted observations with class labels from a fixed finite set. A leaf predicts a probability vector with nonnegative entries summing to one. Zero-probability logarithm terms use the limiting convention.

The empirical class-frequency vector minimizes both multiclass squared probability error and log loss. Their optimal node losses are Gini impurity and entropy, respectively.

☐ Derivation *Recommended*

$$\frac{1}{n_R}\sum_{i\in I_R}\sum_{k=1}^{K}(\mathbf{1}[y_i=k]-q_k)^2 = 1-\sum_{k=1}^{K}p_k^2 + \sum_{k=1}^{K}(q_k-p_k)^2 \tag{7.T5}$$

The class-indicator vector has one active coordinate per observation. Expanding its squared prediction error separates Gini impurity from a nonnegative squared distance between predicted and empirical class frequencies. That distance vanishes only when the vectors agree. This establishes Gini as the optimized indicator-vector loss. Squaring arbitrary numeric class labels instead would impose an ordering that those class indicators do not assume.

$$-\sum_{k=1}^{K}p_k\log q_k = -\sum_{k=1}^{K}p_k\log p_k + \sum_{k=1}^{K}p_k\log\frac{p_k}{q_k} \tag{7.T6}$$

The last term compares the predicted class distribution with the empirical one through relative entropy. The logarithm inequality, or Jensen's inequality, makes this term nonnegative, with equality at the empirical distribution. Predicting zero probability for a class observed in the node instead gives infinite log loss. Thus fitting node probabilities leaves entropy as the minimum achievable node loss.

A split redistributes observations; it does not give both children equal voting power in the loss. Sum each child's observation-wise losses, or equivalently weight its optimized average loss by its fraction of parent observations. An unweighted average gives a tiny child as much influence as a large one and therefore optimizes a different criterion.

□

This derivation explains both purity criteria and their weighting. It does not make small-leaf class frequencies well calibrated. With observation weights, empirical proportions and child fractions must be replaced consistently by normalized weights.

▶ Example: Why child weights matter

The parent has five observations from each class, so Gini impurity is 0.5. After splitting, a pure child holds two observations; the other holds class counts three and five. Its impurity is 1 minus 9/64 minus 25/64 = 15/32. Weight by child sizes: 0.2 times zero plus 0.8 times 15/32 = 0.375. The gain is 0.5 minus 0.375 = 0.125. An unweighted average instead gives 15/64 and the incorrect gain 17/64, approximately 0.266. Keep the fractions until the final display to avoid rounding the intermediate impurity.

Interpretation and limits

A perfectly pure training leaf may contain only one observation. Purity is not proof of reliable probability estimates or strong generalization.

Example: Grow, Then Prune The Weakest Link

Source and scope. The growth and cost-complexity pruning criteria follow the CART framework [7.1]; the four-observation trace is a constructed calculation.

Use four ordered scalar inputs 0, 1, 2 and 3 with responses 0, 2, 3 and 3, respectively. Fit a regression tree using unnormalized sum of squared errors, abbreviated SSE, and the mean response in each leaf. The root mean is two and its SSE is six. Trying midpoint thresholds 0.5, 1.5 and 2.5 gives total child SSE values two-thirds, two and fourteen-thirds. Greedy growth selects 0.5. Splitting its right child at 1.5 then produces three pure leaves predicting 0, 2 and 3. The last leaf contains both remaining observations.

Growth and pruning answer different questions. Starting from this three-leaf tree, compare the loss increase per removed leaf when an internal node is collapsed. For an internal node t, let T_t be its current descendant subtree, let $R(t)$ be the SSE if all its observations share one mean, and let $R(T_t)$ be the sum

of its current leaf SSE values. Write L_t for the number of leaves in that subtree; it exceeds one.

$g(t)$ Penalty per leaf at which retaining this subtree and collapsing it have equal cost, using the same unnormalized SSE convention.

$$g(t) = \frac{R(t) - R(T_t)}{L_t - 1} \tag{7.C1}$$

First compare the current pruning ratios: two-thirds for the right child and three for the root. Collapse the right child, yielding a two-leaf stump predicting zero and eight-thirds with SSE two-thirds. The root now represents a different subtree, so its old ratio is obsolete. Recomputing gives sixteen-thirds, not three. Weakest-link pruning must update these ratios after a collapse rather than sort the original values once.

Table 7.C1. The Nested Pruning Path And Its Penalized Costs

Retained tree	Training SSE	Leaf count	Cost at penalty a
Fully grown	0	3	$3a$
Root split only	2/3	2	$2/3 + 2a$
Root mean only	6	1	$6 + a$

Here $a \geq 0$ is the penalty per terminal leaf in equation (7.7). The three-leaf tree minimizes cost below two-thirds; the stump does so between two-thirds and sixteen-thirds; the root-only tree does so above sixteen-thirds. At either boundary, adjacent trees tie; declare a smaller-tree tie rule if desired. If the loss is instead mean squared error over these four observations, divide both pruning breakpoints by four. Penalty values cannot be transferred between loss normalizations without this conversion.

Scope. This is a complete finite growth-and-pruning trace for the stated data. The general weakest-link procedure recomputes internal-node ratios after each pruning step and needs a consistent rule for tied or nested weakest nodes. It searches subtrees of the tree already grown, not all possible trees. The path uses training data; choosing which retained tree to deploy requires development validation. A smaller penalized training cost is not evidence of a smaller future prediction error.

Essential

7.3 Bagging and random forests

Bagging aims to reduce sample sensitivity by fitting bootstrap resamples and combining predictions. Each resample draws observations with replacement. Average regression outputs; for classification, combine probabilities or votes. This often helps unstable learners such as deep trees. However, independent bootstrap draws conditional on one dataset do not create independent population estimates: the original sample and systematic errors remain shared.

Breiman's bagging paper [7.2] explains why averaging unstable fitted rules can help. His random-forest paper [7.3], Sections 2-3 and the feature-selection experiments, connects ensemble behavior to both tree strength and dependence. Reducing dependence is not enough if the individual trees become uninformative. In the ensemble comparison, vary the candidate-feature count while holding the data split fixed, and inspect ensemble validation error rather than interpreting a deeper tree or more trees as automatic improvement.

The benefit of averaging depends on both individual variability and error correlation. Under a simplified equal-variance, equal-correlation model, an average retains a correlation-dependent variance floor even with many trees. Random forests reduce correlation by considering a random subset of features at each split, in addition to bootstrap sampling in common configurations. This randomization does not force correlation to zero, and using too few candidate features can weaken individual trees. Forest size improves averaging stability but does not remove all overfitting or bias.

An out-of-bag (OOB) observation is omitted from a tree's bootstrap fit. Its OOB prediction does not, however, independently evaluate upstream transformations that used that row, especially supervised feature selection. Ordinary forest OOB scoring does not refit such preprocessing per bootstrap. Aggregate OOB predictions only under a suitable independent-observation design; retain group-aware or temporal evaluation otherwise. OOB-driven tuning of leaf size, feature subsampling or other settings is development, not final testing. Reserve an untouched test set and record memory and prediction costs.

Breiman’s bagging [7.2] reduces instability by averaging models fitted to bootstrap samples. Random forests [7.3] add randomized feature selection to diversify the trees, which explains why the correlation between their errors matters as much as the number of trees.

Notation for equations (7.8), (7.9)

x	Fixed query where base estimators and their average are compared.
B, b	Base-estimator count and index from one through B.

$f_b(x)$	Base-estimator-b prediction at the fixed query.
$\overline{f}(x)$	Equal-weight mean of the B query predictions.
σ^2	Common base-prediction variance over the specified random training/fitting process.
ρ	Common correlation of distinct base predictions under the same random process.
Var	Variance over repeated training/fitting realizations at the fixed query.

$$\overline{f}(x) = \frac{1}{B}\sum_{b=1}^{B} f_b(x) \tag{7.8}$$

$$\mathrm{Var}(\overline{f}) = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2 \tag{7.9}$$

The variance identity assumes B estimators with equal variance sigma squared and equal pairwise correlation rho. It illustrates a correlation floor; real forests need not satisfy these simplifying assumptions exactly.

The identity assumes equal variances and equal pairwise correlations. The assumed correlation matrix must also be positive semidefinite.

◇ Proposition 7.3: Averaging leaves a covariance-dependent variance floor

Notation for this result

B, b, c	Number of base estimators and two estimator indices from one through B.
$f_b, \overline{f}$	Random prediction of base estimator b at the fixed query and their equally weighted mean.
$\mathrm{Var}, \mathrm{Cov}$	Variance and covariance over the stated random training and fitting process.
σ^2, ρ	Common base-prediction variance and common correlation between distinct base predictions.

Assumptions. Predictions are evaluated at one fixed query and vary across a specified random training and fitting process. Every base prediction has the same finite variance and every distinct pair has the same correlation. The resulting covariance matrix is positive semidefinite.

The variance of an equally weighted ensemble consists of a shared component plus an averaging component. More estimators remove only the latter under this model.

□ **Proof** *Advanced*

$$\bar{f} = \frac{1}{B}\sum_{b=1}^{B} f_b \tag{7.T7}$$

$$\mathrm{Var}(\bar{f}) = \frac{1}{B^2}\sum_{b=1}^{B}\sum_{c=1}^{B}\mathrm{Cov}(f_b, f_c) \tag{7.T8}$$

Expand the variance of the average as a sum of covariances, including each estimator with itself. The diagonal contributes one common variance per estimator. Every ordered off-diagonal pair contributes correlation times that variance. Averaging divides the entire sum by the squared estimator count: the diagonal contribution shrinks with count, while shared covariance need not disappear.

$$\mathrm{Var}(\bar{f}) = \frac{B\sigma^2 + B(B-1)\rho\sigma^2}{B^2} = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2 \tag{7.T9}$$

The correlation assumption itself restricts the ensemble size. For a fixed size greater than one, positive semidefiniteness bounds common correlation below by minus the reciprocal of one less than that size. As the ensemble grows this lower bound approaches zero. Thus a fixed negative correlation cannot support an arbitrarily large equal-correlation ensemble; the usual infinite-ensemble variance floor assumes a feasible fixed nonnegative correlation.

Conditioning on one observed dataset can make separate bootstrap draws independent. Removing that conditioning reveals a shared source of error: every fitted tree learned from the same original sample. Their population covariance therefore need not vanish. Random feature subsets may reduce that dependence while weakening individual trees or changing bias. These competing effects explain why randomization has no universally favorable sign.

□

Doubling forest size cannot remove shared bias or a covariance floor. Assess the tradeoff among tree strength, correlation, and computational cost. The simplified equal-correlation formula is an explanatory model, not an assertion that real forest trees have identical variances.

▶ Example: Averaging cannot erase shared error

Let each tree's variance be 4 and each pair's correlation be 0.25. The average's variance separates into a shared term 0.25 times 4 = 1 and a reducible term 0.75 times 4 divided by tree count. Ten trees therefore give 1.3; one hundred give 1.03; the limit is one, not zero. These are consequences of the simplified equal-variance, equal-correlation model, not estimates for a fitted forest. Feature randomization can change both correlation and individual-tree behavior, so verify performance and stability as the ensemble grows.

Interpretation and limits

Random-forest trees share data and can retain correlated errors. Feature subsampling reduces correlation in useful cases; it does not establish independence or immunity to overfitting.

Essential

7.4 Gradient boosting follows the loss gradient

Boosting improves the current predictor in stages. Begin with the constant minimizing training loss. Differentiate the loss with respect to current predictions, negate those derivatives, and fit a weak learner to the resulting pseudo-residuals. Add its predictions scaled by the learning rate. Under half-squared error these targets are ordinary target-minus-prediction residuals; other losses require different derivatives.

A weak tree usually cannot reproduce every pseudo-residual; its restricted function class approximates the descent direction. Some implementations further optimize leaf values or use second-order information. The loss still determines the pseudo-residuals, including for logistic boosting. Histogram gradient boosting accelerates split search by binning feature values; it remains gradient-boosted trees, not a random forest.

Boosting can reduce approximation error by combining simple learners, but bias and variance are not cleanly assigned to separate algorithm families. More stages, deeper trees, and larger learning rates can fit noise; too little capacity can underfit. Choose learning rate, tree complexity, and stage count together. Early stopping uses a validation partition or appropriate internal validation, never the final test set. If selecting the best stage on a development holdout, refit the chosen procedure appropriately before the final evaluation and document the selection rule.

Friedman [7.4] interprets boosting as stagewise descent in function space, enabling different differentiable losses. Freund and Schapire [7.5] provide the foundational AdaBoost construction, whose reweighting viewpoint is related to, but not identical with, residual fitting. XGBoost [7.7] and LightGBM [7.8]

add different computational techniques to tree boosting, including sparsity-aware split search, approximate split statistics, sampling and feature bundling. CatBoost [7.9] addresses a different issue through ordered boosting and categorical-feature processing designed to reduce target-leakage-induced prediction shift; these methods should not be treated as interchangeable names for the simple residual-fitting algorithm.

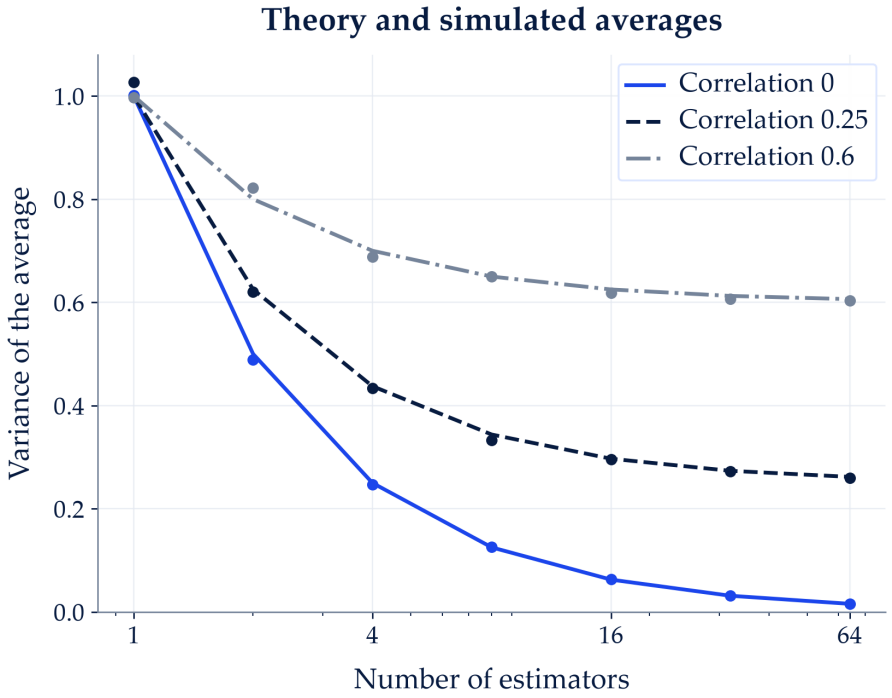


Figure 7.3. Horizontal position counts averaged estimators; height is their equal-weight average variance. Lines give the equal-variance, fixed-common-correlation identity; markers estimate it from 12,000 simulated draws per size. Follow the correlated curves toward their nonzero floors: more estimators cannot remove their shared component.

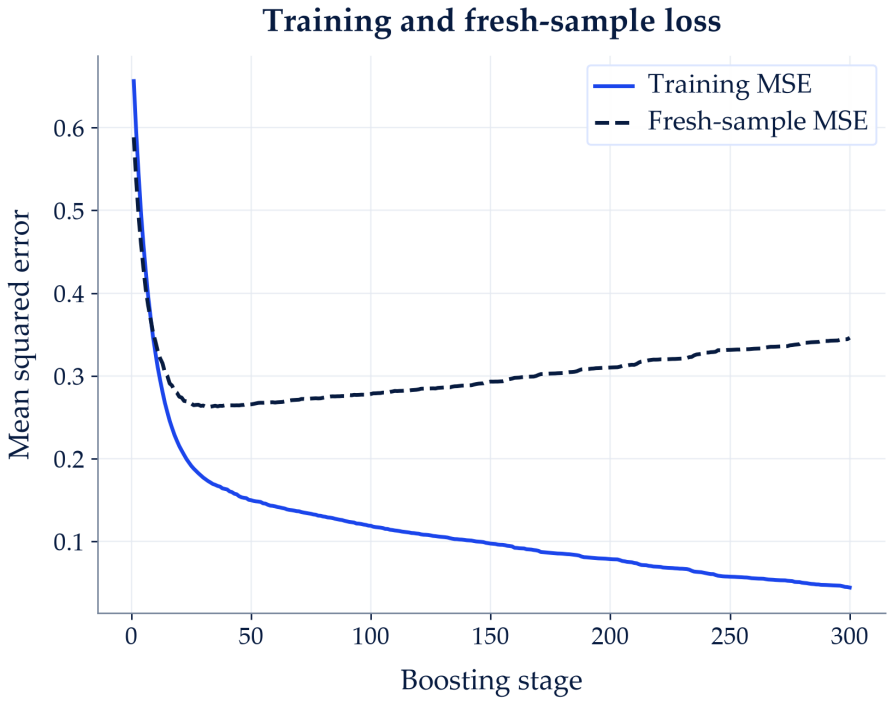


Figure 7.4. Actual squared-error gradient boosting on noisy synthetic regression. Training loss continues falling while a separate fresh-sample loss can flatten or worsen. This fresh sample diagnoses behavior and is not used to select a boosting stage. The curves do not claim monotonic improvement in generalization.

Notation for equations (7.10), (7.11)

i, m	Training-observation index and boosting-stage index; stages start at one.
x_i, y_i	Training-observation-i input vector and target.
$\ell(y_i, F(x_i))$	Differentiable observation loss as a function of its current scalar prediction.
F, F_{m-1}, F_m	Generic predictor and fitted additive models before and after stage m.
r_{im}	Negative prediction derivative of the observation loss at the preceding stage.
$h_m(x)$	Stage-m weak learner fitted to approximate negative training-loss gradients.

η	Positive stage learning-rate factor; smaller values shrink the additive update.
$\cdot\|_{F=F_{m-1}}$	Evaluate the derivative at the preceding model's predictions.

$$r_{im} = -\frac{\partial \ell(y_i, F(x_i))}{\partial F(x_i)}\bigg|_{F=F_{m-1}} \tag{7.10}$$

$$F_m(x) = F_{m-1}(x) + \eta h_m(x) \tag{7.11}$$

The tree h_m approximates the negative loss gradients r_{im} . The learning rate eta shrinks each update. Depending on the algorithm, leaf values or an additional step length are optimized after fitting the tree structure.

This is a scalar-output formulation. Algorithms may additionally optimize a step length or leaf values; that scaling can be incorporated into the stage learner.

Notation for equation (7.12)

y	Observed scalar target for one training case.
F	Current scalar prediction, not the whole prediction function.
$\ell(y, F)$	Half-squared error of target and prediction.
r	Negative prediction derivative of the loss; here the ordinary residual.

$$\ell(y, F) = \tfrac{1}{2}(y - F)^2 \quad \Longrightarrow \quad r = y - F \tag{7.12}$$

Ordinary residual fitting is the half-squared-error special case. With a mean loss, a common factor of one divided by n can be absorbed into the step convention.

For this loss the pseudo-residual is the ordinary residual. The equality does not hold unchanged for arbitrary classification or regression losses.

◇ Derivation 7.4: Why boosting fits negative gradients rather than arbitrary residuals

Notation for this result

n, i, x_i, y_i	Observation count, observation index, input vector, and observed target.
$F, \ell, \mathcal{J}$	In the objective and directional derivative, the current prediction function, differentiable observation loss, and summed empirical objective.
r_i	Negative derivative with respect to the current prediction for observation i.
h, t	Candidate weak learner and scalar step along its prediction direction.

y, F, r	In the two one-observation loss examples only: target, scalar score value, and negative derivative; F here is not a function.
$\sigma(F)$	Logistic sigmoid of the score; the second loss uses binary labels zero or one.

Assumptions. The empirical objective is differentiable with respect to the scalar prediction at every training observation. A candidate weak learner defines a direction in the finite vector of training predictions. A sufficiently small positive step is used when claiming local descent.

A weak learner decreases the objective to first order when its predictions have positive inner product with the negative loss gradient. Ordinary residuals are this gradient only for particular losses.

☐ Derivation *Advanced*

$$\mathcal{J}(F) = \sum_{i=1}^{n} \ell(y_i, F(x_i)) \tag{7.T10}$$

$$r_i = -\frac{\partial \ell(y_i, F(x_i))}{\partial F(x_i)} \tag{7.T11}$$

$$\frac{d}{dt}\mathcal{J}(F + th)\bigg|_{t=0} = -\sum_{i=1}^{n} r_i h(x_i) \tag{7.T12}$$

Differentiate the loss along the proposed additive update. If the weak learner's predictions have positive inner product with the negative gradient, the directional derivative is negative. Differentiability then guarantees improvement for sufficiently small positive steps. Squared-error fitting to pseudo-residuals seeks such a direction within the restricted tree class. Neither that approximation nor a favorable derivative guarantees improvement for an arbitrarily large update.

$$\ell(y, F) = \tfrac{1}{2}(y - F)^2 \Rightarrow r = y - F \tag{7.T13}$$

$$\ell(y, F) = \log(1 + e^F) - yF \Rightarrow r = y - \sigma(F) \tag{7.T14}$$

Differentiate with respect to the quantity the ensemble actually updates. Under squared error that quantity is the prediction, giving target-minus-prediction residuals. For binary log loss it is the log-odds score, but the negative derivative is target minus current probability, not target minus score. A line search or leaf fit can scale this direction; shrinkage reduces its contribution. Select the number of stages using development evidence.

Gradient boosting [7.4] fits loss-dependent negative gradients; ordinary residuals are the squared-error special case. Hollmann and colleagues [7.10] instead evaluate a pretrained TabPFN model that predicts using labeled examples as context. Their 2025 primary benchmark is restricted to at most 10,000 samples and 500 features, and includes datasets where CatBoost wins. Compare complete procedures under matched splits, metrics and budgets. Aggregate benchmark success establishes neither universal dominance nor the behavior of later model versions.

□

Gradient boosting is additive optimization under a specified loss, not a promise to eliminate bias without variance cost. The local descent argument concerns training loss only. Test performance may worsen even when every training update improves the objective.

▶ Example: One boosting update

Targets (1, 2, 6) minus predictions (2, 2, 4) give half-squared-error pseudo-residuals (-1, 0, 2). The weak tree approximates them by (-0.5, -0.5, 2), rather than fitting them exactly. Add 0.1 times this correction to obtain (1.95, 1.95, 4.2). Summed half-squared loss falls from 2.5 to approximately 2.073. This checks the additive update on training data, not future performance; validation determines when to stop. For logistic loss, differentiate with respect to the score instead of using target minus score.

Interpretation and limits

Boosting fits negative loss gradients; ordinary residuals apply to squared-error loss. Its stage count and early stopping must not be selected using the final test set.

Algorithm 7.1. Safeguarded gradient boosting

Source and adaptation. First-order stagewise boosting follows Friedman [7.4]. The finite step list, numerical checks and validation-prefix selection are explicit teaching safeguards; this is not a verbatim reproduction of the original algorithm.

Notation and assumptions

$D_{\mathrm{tr}}, D_{\mathrm{val}}, X, (x_i, y_i), n$	Disjoint training/validation data, training input matrix, training pair i, and training count.
$\ell(y, z), L(F), V(F)$	Differentiable case loss; summed training loss; fixed validation score, oriented lower-is-better.
$F_0, F_m, h_m, \mathcal{H}$	Initial constant, stage-m predictor, fitted regression tree, and declared tree family/rule.
$r_i, \eta, a_m, M, B, \delta$	Negative derivative, positive initial step, accepted step, stage/search budgets and nonnegative descent tolerance.
$\mathcal{A}, \mathrm{FirstDecrease}$	Ordered steps $\eta, \eta/2, \dots, \eta/2^{B-1}$; first finite strict training-loss decrease, or failure.
$\mathcal{R}, m_*, s$	Attempt log, selected prefix index, and termination status.

L(F) sums the training case losses. FirstDecrease scans the finite ordered step list; it logs every trial training loss and rejects nonfinite trials. Record validation scores only for accepted predictors. FitTree uses squared error. A nonfinite descent inner product fails. An unavailable initializer or nonfinite derivative returns a diagnostic, never a successful fit.

The double-bar list operator appends the right-hand entries to the left-hand list.

Algorithm 7.1

Input: Finite disjoint training and validation data; fixed differentiable loss.

Tree rule, validation metric and positive integer budgets.

$$\eta > 0, \quad M \geq 1, \quad B \geq 1, \quad \delta \geq 0$$

Output: Selected additive prefix, attempted-loss log, selected stage and status.

01 **Require** Valid data, fixed controls and untouched final test.

02 Finite constant minimizer. $F_0 \leftarrow \arg\min_{c \in \mathbb{R}} \sum_i \ell(y_i, c)$

03 **Require** Finite initializer, training loss and validation score.

04 $\mathcal{R} \leftarrow [(0, L(F_0), V(F_0))]$

05 $m_* \leftarrow 0$

Algorithm 7.1 (continued)

06 $s \leftarrow$ budget exhausted

07 **For** $m = 1, \ldots, M$

08 $r_i \leftarrow -\partial_z \ell(y_i, z)\big|_{z=F_{m-1}(x_i)}$

09 **Require** Finite pseudo-residuals.

10 Squared-error regression tree; require finite predictions on the training and validation inputs.
$h_m \leftarrow \mathrm{FitTree}(X, r; \mathcal{H})$

11 **Stop if** No verified descent direction. $\sum_i r_i h_m(x_i) \le \delta$

12 Log every attempted step.
$a_m \leftarrow \mathrm{FirstDecrease}(F_{m-1}, h_m, \mathcal{A})$

13 **Stop if** Search fails; do not add this tree.

14 $F_m \leftarrow F_{m-1} + a_m h_m$

15 Require finite training and validation scores before logging. On failure, set numerical-failure status and stop without accepting this stage.
$\mathcal{R} \leftarrow \mathcal{R} \,\|\, [(m, L(F_m), V(F_m))]$

16 **If better** Retain earlier stages on ties.
$V(F_m) < V(F_{m_*}) \;\Rightarrow\; m_* \leftarrow m$

17 **End for**

18 **Return** $(F_{m_*}, \mathcal{R}, m_*, s)$

Why this works. Set status to numerical failure, no descent, or search exhaustion on the corresponding early stop; return the saved finite prefix when one exists. This first-order variant uses neither Hessians nor leaf-specific line searches. Unsmoothed logistic loss needs both classes for a finite initializer. Training descent is not a generalization guarantee; final test data never select a stage.

Recommended

7.5 Voting and leakage-free stacking

Voting fixes a rule for combining models. Hard voting combines labels; soft voting averages class probabilities, possibly with nonnegative weights summing to one. Align class order first. Choose weights on development validation, never test results. Miscalibrated probabilities can give a model undue influence. Compare the ensemble with its individual models: complementary strengths must justify the extra complexity.

Build each development row's meta-features using base procedures fitted and tuned on other folds, including preprocessing. Otherwise an interpolating base model can leak that row's target into its meta-features. Fit the stacking meta-model on the completed out-of-fold matrix. Those predictions are its training inputs; they do not independently evaluate the whole stack.

At deployment, feed the fitted meta-model predictions from bases refitted on all development rows. These bases saw more data than those generating its out-of-fold training inputs, so prediction distributions can differ; this is not leakage, but still needs evaluation. Repeated tuning against the same out-of-fold matrix can overfit development. Use compatible group or temporal folds; some temporal rows have no valid earlier training history. Evaluate the complete stack once on an untouched test set, including calibration and inference costs.

Wolpert [7.6] introduced stacked generalization: a second learner combines predictions from base learners. Training that combiner on out-of-fold predictions preserves the distinction between learning which models to trust and merely exploiting their fitted training errors.

A tabular foundation-model comparison

Boosted trees remain useful baselines, but they are not the only serious candidates for tabular prediction. Hollmann and colleagues' 2025 Tabular Prior-data Fitted Network (TabPFN) study [7.10] evaluates a pretrained transformer on small-to-medium classification and regression tasks, including benchmarks restricted to at most 10,000 observations and 500 features. It reports strong aggregate results against tree baselines, while also showing datasets where CatBoost wins. This is evidence about that study's versions, datasets, metrics, and budgets, not a universal ordering or a forecast for 2027.

Unlike a network trained from scratch on one table, TabPFN learns across synthetic prediction tasks beforehand. At use time, labeled training examples provide context for predicting unseen labels. The synthetic-data generation process supplies a prior over tasks; it does not establish that a real application's causal structure matches that prior.

For a local comparison, keep boosted-tree and simple linear baselines, use identical admissible splits, and reserve final labels. Record the model and weight version, pretraining provenance, hardware, tuning budget, preprocessing, prediction latency, and memory. Separate reusable pretraining cost from per-dataset adaptation and inference cost. Freeze choices before final assessment; published aggregate success does not replace checks of calibration, shift, or resource suitability for the intended application.

Notation for equation (7.13)

x	Query supplied to each base classifier after its fitted preprocessing.
y, c	Class-valued outcome and the class label whose probability is combined.
B, b	Base-classifier count and index from one through B.
$p_b(y = c \mid x)$	Base-classifier-b probability for class c at the query.
a_b	Nonnegative classifier weight; weights sum to one.
$p_{vote}(y = c \mid x)$	Weighted mean of aligned base-classifier class probabilities.

$$p_{vote}(y = c \mid x) = \sum_{b=1}^{B} a_b p_b(y = c \mid x), \qquad a_b \geq 0$$
$$\sum_{b} a_b = 1 \tag{7.13}$$

Soft voting averages aligned class probabilities. Weights are fixed beforehand or selected on development data. An average of poorly calibrated probabilities need not be well calibrated.

Every classifier must refer to the same classes in the same order. Weight selection belongs to development data, not the final test set.

Notation for equations (7.14), (7.15)

n	Development-observation count and meta-feature row count.
Z, Z_{ib}	Out-of-fold meta-feature matrix and entry (i,b); scalar predictions give shape n by B.
i, b, B	Development-observation index, base-model index and base-model count.
x_i, x	Development-observation-i input and later query, respectively.
$fold(i)$	Observation-i validation fold in the predefined development split.
$f_b^{(-fold(i))}$	Base procedure b fitted without any observation in i's fold.
f_b^{full}	Base procedure b refitted on all development data after out-of-fold feature generation.

g	Meta-model fitted to out-of-fold predictions and development targets.
$\widehat{y}(x)$	Meta-model prediction from the refitted base-model predictions.

$$Z_{ib} = f_b^{(-\,\mathrm{fold}(i))}(x_i) \tag{7.14}$$

$$\widehat{y}(x) = g(f_1^{\mathrm{full}}(x), \ldots, f_B^{\mathrm{full}}(x)) \tag{7.15}$$

The meta-model g is trained on out-of-fold base predictions Z. For deployment, full denotes base models refitted on all development data, while the final test data remains untouched by either fitting stage.

For multiclass probability inputs, a base model contributes multiple columns rather than one. The displayed scalar-output notation suppresses that additional class dimension.

◇ Proposition 7.5: The exclusion condition behind out-of-fold stacking

Notation for this result

Z_{ib}	Entry in row i and base-model column b of the meta-feature matrix.
$x_i, v(i)$	Input of development observation i and its predefined validation-fold index.
$f_b^{(-v(i))}$	Base procedure b fitted using only observations outside the fold containing i.
i, b	Development-observation index and base-procedure index.

Assumptions. Observation pairs are independent draws and validation folds are fixed without consulting their target values. Every base fitting operation, including preprocessing and any nested tuning, excludes the fold whose predictions it generates.

Each out-of-fold meta-feature is produced by a base procedure that did not use the corresponding observation's target. This removes direct base-model target reuse, but does not make the fitted meta-model's training score an unbiased test estimate.

□ Proof *Advanced*

$$Z_{ib} = f_b^{(-v(i))}(x_i) \tag{7.T15}$$

Under independent sampling, a base procedure fitted outside a fold cannot see the held-out observation's target. Conditional on that observation's input, its meta-feature therefore cannot adapt to that particular target value. An

interpolating model fitted on all rows can instead reproduce the target directly as a meta-feature. The meta-model may learn that shortcut even though future inputs will not come with their targets; out-of-fold construction prevents this direct reuse.

Excluding a row from its base fit protects that row's meta-features, not the meta-model's own training score. The second stage deliberately fits those features to all development targets, so its residuals are in-sample and can be optimistic. After creating meta-features, refit base procedures on permitted development data, freeze the fitted meta-model, and evaluate the complete stack on untouched observations. Grouped or temporal data need corresponding exclusion rules.

□

Out-of-fold construction is a data-flow condition, not a guarantee that model combination improves performance. Target-aware tuning performed before constructing folds can defeat the condition even when the final prediction code appears to exclude each row.

▶ Example: Construct a three-model stack

With 600 development observations, three binary base classifiers, and five equal folds, each base fit uses 480 rows and predicts the other 120. Fifteen fits fill a 600-by-3 out-of-fold matrix with one positive-class probability per classifier; each row's target is excluded from its base fits. Fit one regularized logistic meta-model there, then refit three base procedures on all 600 rows: 19 fits, excluding tuning. Assess the complete stack on untouched data against an equal-weight probability vote. The meta-model's fit on its out-of-fold inputs is not that final assessment.

▶ Example: Follow one row through the combiner

For one new row, suppose the three frozen base models return positive-class probabilities (0.2, 0.6, 0.7). Equal-weight soft voting returns (0.2 + 0.6 + 0.7)/3 = 0.5. A hypothetical frozen logistic combiner with intercept -1 and coefficients (-1, 2, 1) instead gives logit -1 - 0.2 + 1.2 + 0.7 = 0.7, hence probability 1/(1 + exp(-0.7)), about 0.668. The coefficients illustrate inference; actual stacking learns them from out-of-fold development predictions. Neither calculation says which procedure generalizes better.

Interpretation and limits

Training a meta-model on base predictions from the same examples used to fit those bases causes leakage. Out-of-fold predictions are required, not optional decoration.

Dataset: Wisconsin Diagnostic Breast Cancer (WDBC). See its first-use portrait [7.11].

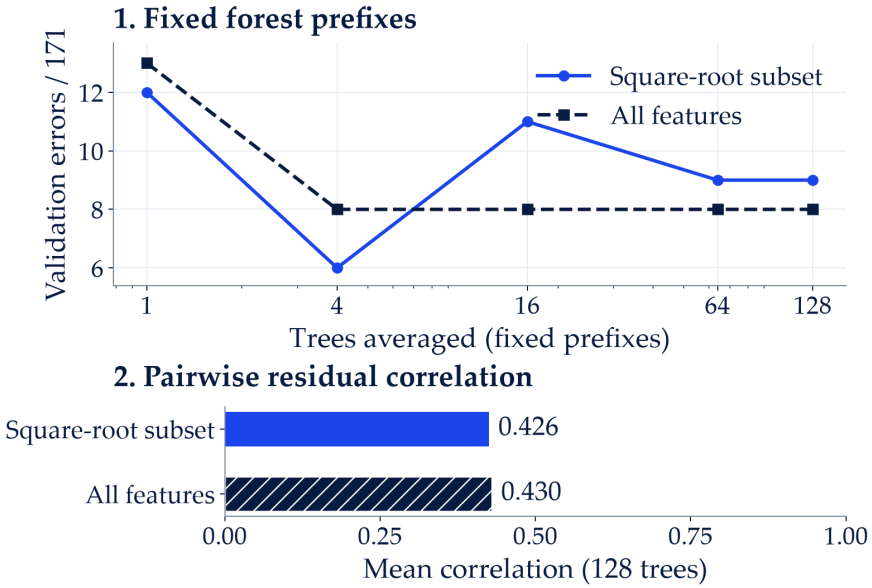


Figure 7.5. Measured forest prefixes on a fixed WDBC development split: 398 fitting and 171 validation rows. Two 128-tree forests use depth six and either square-root feature subsampling or all features. The upper panel evaluates fixed prefixes, not validation-selected tree counts; the lower panel measures pairwise tree residual correlations across those same validation rows. The population equal-correlation identity does not automatically apply to these empirical correlations. This is one descriptive benchmark experiment, not a clinical validation or uncertainty interval.

Table 7.1. Ensembles combine models in different ways

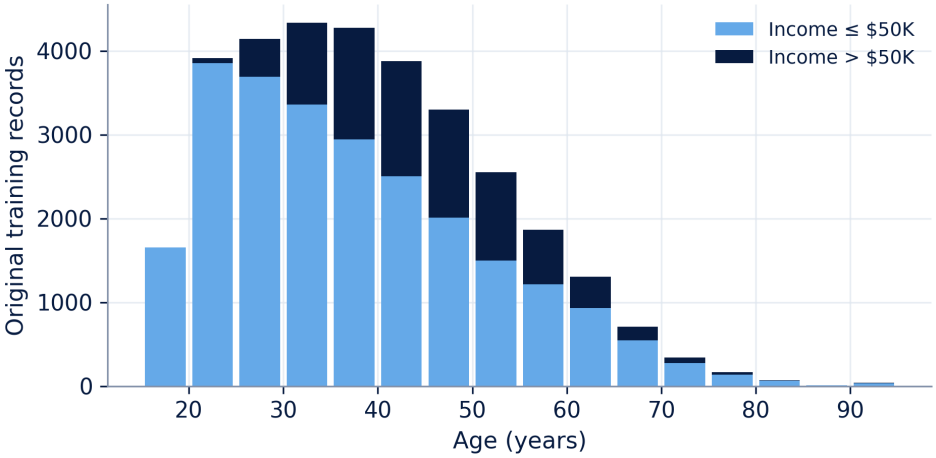
Method	Combination mechanism	Key limitation
Bagging	Fit on bootstrap samples and average or vote.	Highly correlated base learners limit variance reduction.
Random forest	Bagging plus randomized candidate features at splits.	Feature importance can be misleading with correlated predictors.
Gradient boosting	Sequentially fit corrections driven by a loss gradient.	Depth, learning rate and number of stages must be controlled together.
Stacking	Fit a meta-model to out-of-fold base predictions.	In-sample base predictions leak fitting information into the meta-model.

Table 7.2. Chapter synthesis

Concept	Main conclusion	Engineering check
Trees	Recursive partitions create locally constant predictions.	Constrain depth or leaf size and validate the whole fitting procedure.
Averaging	Variance reduction depends on correlation, not just model count.	Compare diversity and an individual-tree baseline.
Boosting	Each stage follows the chosen loss, not a universal residual formula.	Use loss-consistent pseudo-residuals.
Stacking	Training the combiner requires honest base predictions.	Generate features out of fold before fitting the meta-model.

Adult Census Income

48,842	14	2
source records	raw predictors	income labels



Age and income labels in the original 32,561-row training partition. Counts describe this historical sample, not contemporary population income rates.

Adult contains 48,842 historical census-income records with 14 numerical or categorical predictors. The task is whether annual income exceeds $50,000. Missing categorical entries and demographic proxies make preprocessing and interpretation important; excluding explicit sensitive fields does not establish fairness or suitability for decisions about people. [7.12]

Here: original partitions contain 32,561 training and 16,281 test records; the case uses a smaller 13,000-record sample. Missing categorical values require handling. Excluding race and sex from predictors does not establish fairness.

Becker and Kohavi / UCI, CC BY 4.0. Unweighted historical records. Source / terms.

Case Study: Compare tree ensembles on historical Adult census records

Problem and data

Adult offers a real mixed-type classification task with a historical income label. The question is whether an ensemble adds useful predictive performance over a constrained tree under one declared computational budget. This is a retrospective educational comparison, not a rule for allocating opportunities. Exclude the census weighting field and explicit race and sex predictors, document the resulting unweighted estimand, and retain the original test partition until the model and preprocessing choices are frozen.

Experimental design

Take fixed stratified subsets from the official partitions and reserve validation records within the original training data. Fit numerical imputation and categorical encoding on training records only. Give a depth-constrained tree, a random forest, and histogram gradient boosting the same transformed matrices. Select by validation macro F1 with a declared tie rule. Evaluate the selected model and a majority baseline once, then display the test confusion matrix and count-supported descriptive sex-slice errors without changing the decision rule.

Measured Reference Run

The original training partition supplied 7,500 fitting and 2,500 validation records; 3,000 original-test records were reserved. Fitting-only imputation and one-hot encoding excluded fnlwgt, race and sex. The fixed budget compared a depth-four tree, an 80-tree forest and 60 boosting iterations. Maximum validation macro F1 selected histogram boosting without refitting. Final scores are unweighted record metrics.

Table 7.3. Measured case-study evidence

Final procedure	Accuracy	Macro F1	Test count
Histogram boosting	0.862	0.793	3000
Training majority	0.764	0.433	3000

Boosting won validation macro F1: 0.798 versus the forest's 0.748 and constrained tree's 0.741. Macro F1 weights class-specific F1 equally; accuracy

weights records equally. Beating the majority reference on both final metrics therefore was not merely higher majority-class accuracy.

Failure Analysis And Limits

Aggregate performance concealed different descriptive errors: 0.079 among 989 records labeled Female and 0.167 among 2,011 labeled Male. Their positive counts also differed, 102 versus 607. Excluding the recorded fields did not make predictive behavior uniform. These slices neither identify causal discrimination nor establish fairness; correlated proxies and historical sampling remain. Excluding fnlwgt means the scores are not census-population estimates.

Historical saved run with original-partition boundaries and fixed subset seeds. Preserve fitted preprocessing and excluded-field choices; this is not a social decision system.

▶ Example: inspecting the real data

The first fitting record has workclass Private, activating its fitted one-hot indicator. The shared transformed matrix has 7,500 rows and 95 columns: 95 is an encoded dimension, not the raw predictor count. The final confusion-matrix cells must sum to 3,000, the number of selected original-test records.

Dataset source and documentation

Companion: Week 07, Real-Data Case Study notebook. Reproduce the declared run, then investigate a changed condition. No fixed benchmark score is promised.

A Controlled Learning Cycle

The Adult case evaluates the selected ensemble as a complete procedure. Week 7 notebooks connect split gains, forest diagnostics, staged fitting and leakage-free meta-features to that decision. The Wine laboratory isolates CART paths, bootstrap multiplicities and OOB eligibility. Record one changed setting with its metric, data roles and explanation; use the scope below to distinguish these mechanisms from pruning, forests, boosting and stacking.

Matched practice and simulator protocol · Packaged edition; launch instructions in the guide.

Laboratory 07 | Trees And Ensemble Learning

Learned CART paths, bootstrap multiplicities and OOB eligibility.

Browser laboratory dataset: Wine Recognition. See its first-use portrait [7.13].

Learned CART splits and bootstrap bagging reveal individual decision paths, repeated training rows and out-of-bag votes on measured Wine data.

Open Laboratory 07 | Use the live view for readable controls.

Transfer: Historical Adult income classification → Wine classification. No random forests, boosting, stacking or pruning control.

Chapter Capability

Compare a single tree, a bagged ensemble, and boosting using the same data boundaries and diagnose whether improvement survives held-out observations.

Week 07: separate practice, solutions and assessment

References

[7.1] Breiman, L.; Friedman, J. H.; Olshen, R. A.; Stone, C. J. (1984). Classification and Regression Trees. Chapman & Hall/CRC. Source / DOI

[7.2] Breiman, L. (1996). Bagging predictors. Machine Learning 24, 123-140. Source / DOI

[7.3] Breiman, L. (2001). Random Forests. Machine Learning 45, 5-32. Source / DOI

[7.4] Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. The Annals of Statistics 29(5), 1189-1232. Source / DOI

[7.5] Freund, Y.; Schapire, R. E. (1997). A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. Journal of Computer and System Sciences 55(1), 119-139. Source / DOI

[7.6] Wolpert, D. H. (1992). Stacked generalization. Neural Networks 5(2), 241-259. Source / DOI

[7.7] Chen, T.; Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Source / DOI

[7.8] Ke, G.; Meng, Q.; Finley, T.; Wang, T.; Chen, W.; Ma, W.; Ye, Q.; Liu, T.-Y. (2017). LightGBM: A Highly Efficient Gradient Boosting Decision Tree. Advances in Neural Information Processing Systems 30. Source / DOI

[7.9] Prokhorenkova, L.; Gusev, G.; Vorobev, A.; Dorogush, A. V.; Gulin, A. (2018). CatBoost: unbiased boosting with categorical features. Advances in Neural Information Processing Systems 31. Source / DOI

[7.10] Hollmann, N.; Müller, S.; Purucker, L.; Krishnakumar, A.; Körfer, M.; Hoo, S. B.; Schirrmeister, R. T.; Hutter, F. (2025). Accurate predictions on small data with a tabular foundation model. Nature 637, 319-326. Source / DOI

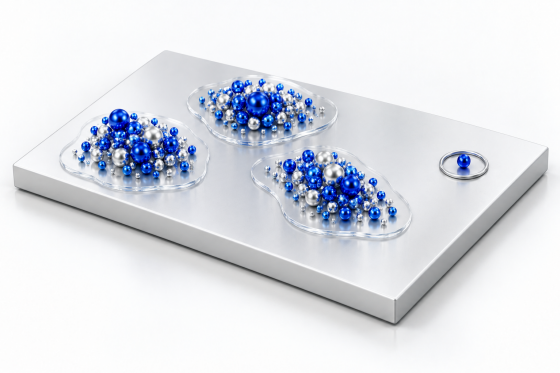
[7.11] Wolberg, W.; Mangasarian, O.; Street, N.; Street, W. (1993). Breast Cancer Wisconsin (Diagnostic). UCI Machine Learning Repository. Source / DOI

[7.12] Becker, B.; Kohavi, R. (1996). Adult. UCI Machine Learning Repository. Source / DOI

[7.13] Aeberhard, S.; Forina, M. (1992). Wine. UCI Machine Learning Repository. Source / DOI

CHAPTER 08

Clustering And Anomaly Detection



Reading path

Start with Essential. The other levels offer more depth.

Essential
8.1 8.2 8.4 8.5

Recommended
8.3

Advanced
Proofs 8.2 8.3

Worked example Lab task

A clustering method always reflects a geometry and a definition of a group. A visually attractive partition is a hypothesis about structure, not a discovered ground truth.

What You Will Be Able To Do

- Implement Lloyd's k-means updates and explain their objective and local-optimum behavior.
- Compare cluster structure using silhouette, stability, and task-specific evidence.
- Distinguish hierarchical linkage and DBSCAN density connectivity.
- Separate anomaly scores, distribution drift, and confirmed faults using a baseline-only monitoring experiment.

Before you begin: Distances and feature scaling; Means and squared-error objectives; Validation and representative sampling.

Follow one centroid update and one density neighborhood before interpreting a dendrogram or designing alerts.

Algorithm scope: Complete teaching implementations: Lloyd iterations and DBSCAN neighborhood expansion. Linkage and alert interpretation do not provide LOF or Isolation Forest implementations.

Shared chapter notation

ϵ Positive neighborhood radius for density-based spatial clustering of applications with noise; in chosen distance units and used by every query; epsilon in prose and pseudocode.

Chapter notation

DBSCAN Density-based spatial clustering of applications with noise: clustering through connected dense neighborhoods under a fixed metric.

Without target labels, clustering groups observations by a stated similarity rule and anomaly detection identifies departures from a reference. Neither determines meaning or consequence. A cluster may reflect a population, acquisition batch, measurement artifact or arbitrary division of a continuum; an anomaly may be a fault, a discovery or an ordinary but underrepresented observation.

K-means, hierarchical clustering and DBSCAN establish the geometries behind later anomaly and drift diagnostics. Controlled synthetic examples expose failures while separating mathematical objectives from substantive conclusions. Before using a geometric signal operationally, identify the reference data, the question its geometry answers and the additional evidence the decision requires.

- *Essential*

8.1 K-means and alternating optimization

Notation for this section

K Requested positive integer number of centroid groups; the count does not establish a natural number of semantic classes.

K-means alternates between two easier problems: assign observations to the nearest centroid, then replace each occupied centroid by its assigned mean. Lloyd's iteration repeats these steps under a declared stopping rule to reduce summed squared Euclidean distances to K centroids. Each exact step is non-increasing, but neither searches all assignments and centers jointly. A falling objective is therefore not a certificate of the global optimum.

Initialization matters because the objective has many local solutions. K-means++ spreads initial centers using a distance-based randomized rule, and multiple restarts compare solutions under the same objective. Document the random seed and restart count rather than relying on changing defaults. Empty clusters require a specified repair strategy in a manual implementation. Scaling matters because squared distances amplify large-coordinate differences; a noisy feature measured on a large scale can dominate the resulting partition even if it has little substantive value.

The partition induced by nearest centroids consists of convex Voronoi regions, making k-means most natural when compact groups can be represented by their means under the chosen metric. Unequal densities, elongated shapes, outliers, and curved structures can produce undesirable partitions. A centroid can also be an unrealistic combination of valid observations, especially for constrained or categorical data. Choosing K states a modeling resolution; it does not assert that nature contains exactly K classes or that every example must belong to a meaningful category.

Lloyd's quantization algorithm [8.1] provides the alternating assignment-and-centroid update underlying standard k-means. Each exact update cannot increase the within-cluster squared error, but that monotonicity does not establish convergence to the globally best partition. Arthur and Vassilvitskii [8.7] analyze k-means++ initialization, which samples subsequent centers according to squared distance from the centers already chosen. Its expected approximation guarantee is not a promise that a particular Lloyd run reaches the global optimum.

Notation for equations (8.1), (8.2)

J	Total within-cluster squared Euclidean distance for current assignments and centroids; inertia.
n, i	Fitted-observation count and index from one through n.
x_i	Observation-i vector in the feature coordinates supplied to clustering.
K, k	Requested cluster count and index from one through K.
c_i	Observation-i cluster index, chosen by nearest centroid.
μ_k, μ_{c_i}	Candidate-cluster-k centroid and observation-i assigned centroid.
$\arg\min$	Nearest-centroid index; ties require a consistent rule.
$\|\cdot\|^2$	Squared Euclidean norm in the chosen representation; scaling changes the objective.

$$J = \sum_{i=1}^{n} \|x_i - \mu_{c_i}\|^2 \tag{8.1}$$

$$c_i \in \arg\min_{k \in \{1,\dots,K\}} \|x_i - \mu_k\|^2 \tag{8.2}$$

c_i is the assigned cluster index, μ_k its centroid, and K the requested cluster count. The objective is a sum, often called inertia in implementations. Comparing it across different sample sizes requires attention to normalization.

Every centroid has the same number of coordinates as an input vector. These assignments minimize the objective only for fixed centroids.

Notation for equation (8.3)

k	Updated cluster index, from one through the requested cluster count.
μ_k	Updated centroid, one coordinate per input feature.
n_k	Positive count currently assigned to cluster k.
x_i	Observation-i vector in the clustering coordinates.
c_i	Observation-i assignment, fixed during this centroid update.
$i : c_i = k$	Indices currently assigned to the updated cluster.

$$\mu_k = \frac{1}{n_k} \sum_{i:c_i=k} x_i \tag{8.3}$$

For nonempty cluster k, its mean minimizes the sum of squared distances given fixed assignments. This update is specific to squared Euclidean geometry; other metrics generally require different representatives.

This update is defined only for a nonempty cluster and squared Euclidean loss. An empty cluster needs an explicit repair rule.

◇ Proposition 8.1: Lloyd's algorithm is monotone coordinate descent

Notation for this result

d	Positive number of real coordinates in each input and centroid vector.
n, i, x_i	Observation count, observation index, and real d-dimensional input vector.
c, c_i	Vector of cluster assignments and the index assigned to observation i.
μ, μ_k	Collection of K centroids and centroid k, each with d coordinates.
J, K, k	Distortion objective, requested cluster count, and cluster index from one through K.
t	Iteration index; superscripts indicate the current and next iterates.
$u, \bar{x}_k, n_k$	Candidate centroid, mean of currently assigned observations, and positive cluster count.

Assumptions. The dataset is finite and the objective uses squared Euclidean distance in fixed feature coordinates. Assignment and mean-update steps are exact. Occupied clusters are nonempty; an empty centroid may be retained because it contributes no loss before reassignment. Distance ties use a documented deterministic rule. Numerical stopping tolerances are separate from the exact mathematical argument.

An exact assignment step and an exact centroid step each cannot increase the distortion objective. This establishes monotone improvement, not a globally optimal partition.

Proof *Recommended*

$$J(c, \mu) = \sum_{i=1}^{n} \|x_i - \mu_{c_i}\|_2^2 \tag{8.T1}$$

At fixed centroids, each summand depends only on its own assignment. Choosing a nearest centroid minimizes that summand independently, so summing these choices cannot increase the objective.

$$c_i^{t+1} \in \arg\min_{1 \le k \le K} \|x_i - \mu_k^t\|_2^2 \tag{8.T2}$$

$$J(c^{t+1}, \mu^t) \le J(c^t, \mu^t) \tag{8.T3}$$

Freeze assignments before optimizing a centroid. Add and subtract that cluster's sample mean in every squared distance. Centered deviations sum to zero, canceling the cross term; the extra cost is cluster size times squared displacement from the mean. For a nonempty cluster it is minimized uniquely at the mean. This proves only the centroid update with assignments fixed, not global optimality of alternating updates.

$$\sum_{i:c_i=k} \|x_i - u\|_2^2 = \sum_{i:c_i=k} \|x_i - \bar{x}_k\|_2^2 + n_k \|u - \bar{x}_k\|_2^2 \tag{8.T4}$$

$$J(c^{t+1}, \mu^{t+1}) \le J(c^{t+1}, \mu^t) \le J(c^t, \mu^t) \tag{8.T5}$$

Distinguish convergence of objective values from termination of assignments. Non-increasing, nonnegative values have a limit. Finite termination additionally follows from finitely many assignments if clusters stay nonempty, every assignment change strictly decreases the objective, and ties use a fixed rule. Degenerate ties require care. A fixed point optimizes each block with the other held fixed; it need not beat a simultaneous change of assignments and centers.

□

Different initial centers can settle at different fixed points. Multiple restarts compare attained objectives without proving global optimality. Empty-cluster repairs that move occupied centroids need separate verification; the monotonicity proof covers only updates that do not increase the current objective.

◇ Proposition 8.2: Centroid assignment creates convex cells in the chosen metric

Notation for this result

K	Positive number of fitted centroid vectors.
x, μ_k, μ_j	Query input and two fixed centroids in the same d-dimensional coordinates.
k, j	In the nearest-center inequality only, candidate-centroid and comparison-centroid indices, each from one through K.
z, x_j, z_j	Second input and the jth coordinates of the two inputs.
d, j	Positive feature count; in the scaling identity only, j indexes coordinates from one through d, not centroids.
D, s_j	Diagonal scaling matrix and its positive scale for feature j.
diag	Matrix constructor placing its arguments on the diagonal and zeros elsewhere.

Assumptions. The fitted centroids are fixed and Euclidean distance is used for new assignments. Closed cells may overlap on ties; a fixed centroid-index priority supplies a consistent assignment rule. For the scaling extension, each feature scale is strictly positive and the same transform is applied to data and centroids.

Compare one centroid with each competitor. The squared-distance comparison cancels the input's squared norm and leaves a half-space. Intersecting these half-spaces gives that centroid's convex closed region; tied boundaries can be shared. Standardization replaces raw Euclidean distance by a diagonal quadratic metric. Unequal coordinate scales can change the partition. A common positive scale multiplies every squared distance by the same constant and preserves nearest-centroid assignments under the same tie rule.

□ Proof *Advanced*

$$\begin{aligned} & \|x-\mu_k\|_2^2 \le \|x-\mu_j\|_2^2 \\ \iff \quad & 2(\mu_j-\mu_k)^\top x \le \|\mu_j\|_2^2 - \|\mu_k\|_2^2. \end{aligned} \tag{8.T6}$$

Expand the two squared distances from a query to competing centroids. The query's squared norm appears on both sides and cancels, leaving a linear inequality. A centroid wins against all others only where all these inequalities hold. Each defines a half-space; their intersection is convex because every convex combination of two satisfying queries still satisfies every inequality.

$$D = \operatorname{diag}(s_1, \dots, s_d) \tag{8.T7}$$

$$\|D^{-1}(x - z)\|_2^2 = \sum_{j=1}^{d} \frac{(x_j - z_j)^2}{s_j^2} \tag{8.T8}$$

Subtracting a common training mean does not affect pairwise differences: the mean cancels. Dividing by feature scales does remain, reducing the contribution of coordinates with larger scales. If a measurement's units change, change its fitted scale by the same factor and standardized distances are preserved. Changing raw units alone changes the geometry and can change assignments.

□

Convex cells explain why a nearest-centroid partition can cut across curved or nested groups. Scaling specifies which deviations are comparable; it cannot establish semantic clusters or guarantee that noisy coordinates deserve equal influence.

▶ Example: One assignment and update

For observations 0, 1, 8, 9 and initial centers 0, 8, nearest-center assignments form groups (0, 1) and (8, 9). Squared error is 0 + 1 + 0 + 1 = 2. Updating to means 0.5 and 8.5 leaves four squared residuals of 0.25, so error becomes one; reassignment changes nothing. Add 100: the temporary group (8, 9, 100) has mean 39, far from its original pair. Later assignments may change again. This outlier sensitivity follows from squared loss, while different initializations can still lead to different local solutions.

Interpretation and limits

K-means imposes a requested cluster count rather than discovering a natural number of classes. Degenerate data can produce fewer occupied groups. A partition is not evidence that discrete natural classes exist.

Algorithm 8.1. Lloyd iterations with seeded restarts

Source and adaptation. Alternating nearest-centroid and mean updates follow Lloyd [8.1]. The declared restart, tie, empty-center and numerical policies specify this teaching implementation. For a k-means++ initializer, see Arthur and Vassilvitskii [8.7].

Notation and assumptions

$X = (x_i)_{i=1}^{n}, d, K$	Finite n-by-d input matrix and requested centroid count; $1 \le K \le n$.
$\mu = (\mu_k)_{k=1}^{K}, c, c'$	K d-entry centroids and current/candidate length-n assignment vectors.
$A(\mu), \mathcal{J}(c, \mu)$	Nearest-centroid assignment with lowest-index ties; $\sum_i \lVert x_i - \mu_{c_i} \rVert^2$.
$n_k(c), \mu', J, J'$	Assigned count for group k; candidate centroids and current/candidate objectives.
R, T, r, t, ε	Positive restart/iteration budgets, their indices, and declared roundoff tolerance.
$c_{\mathrm{old}}, s, \mathcal{B}, J_*$	Pre-update labels, run status, saved best state, and its objective.
$\mathrm{Init}, K_{\mathrm{occ}}$	Declared seeded centroid initializer; number of distinct returned labels.

A chooses the smallest centroid index among distance ties; $n_k(c)$ counts assigned indices. Init follows the declared seed schedule and must return K finite centroids with the input dimension. Require a finite initial objective. A failed check returns a numerical diagnostic and the best completed restart, if one was saved.
Compare restarts by strict objective improvement, preserving the first on ties. Failed finite-value or non-increase checks terminate with a numerical diagnostic rather than claiming convergence. All centroid updates below are simultaneous from c.

Algorithm 8.1

Input: Finite Euclidean observations and valid integer K.

Fixed seeds, positive integer restart/iteration budgets and roundoff tolerance.

Output: Best centroids, nearest-center labels, objective, occupied count and status.

Require Finite data, valid K, seeds and budgets.
$J_* \leftarrow +\infty$
For $r = 1, \dots, R$
    $\mu \leftarrow \mathrm{Init}(X, K, r)$
    $c \leftarrow A(\mu)$

Algorithm 8.1 (continued)

$J \leftarrow \mathcal{J}(c, \mu)$

$s \leftarrow$ budget exhausted

For $t = 1, \ldots, T$

  Save $c_{\text{old}} \leftarrow c$

  For every centroid; retain empty centers.

  $$\mu_k^{'} \leftarrow \begin{cases} n_k(c)^{-1} \sum_{i:c_i=k} x_i, & n_k(c) > 0 \\ \mu_k, & n_k(c) = 0 \end{cases}$$

  Reassign $c^{'} \leftarrow A(\mu^{'})$

  $J^{'} \leftarrow \mathcal{J}(c^{'}, \mu^{'})$

  Require Finite candidate and non-increasing objective.

  $J^{'} \leq J + \varepsilon$

  $(c, \mu, J) \leftarrow (c^{'}, \mu^{'}, J^{'})$

  If unchanged Set converged status.

  $c = c_{\text{old}} \;\Rightarrow\; s \leftarrow$ converged

  Break if Run status is converged.

End for

Save if better $J < J_* \;\Rightarrow\; (\mathcal{B}, J_*) \leftarrow ((\mu, c, J, s), J)$

End for

Return Saved state and distinct-label count. $(\mathcal{B}, K_{\text{occ}})$

Why this works. No empty-group division occurs. Retaining empty centers preserves descent but may yield fewer than K occupied groups. At convergence, occupied centers equal their assigned means. On budget exit the returned labels are nearest-center labels; centers need not equal the means of those newly reassigned groups. No restart certifies a global minimum. If no restart was saved, return only the numerical diagnostic.

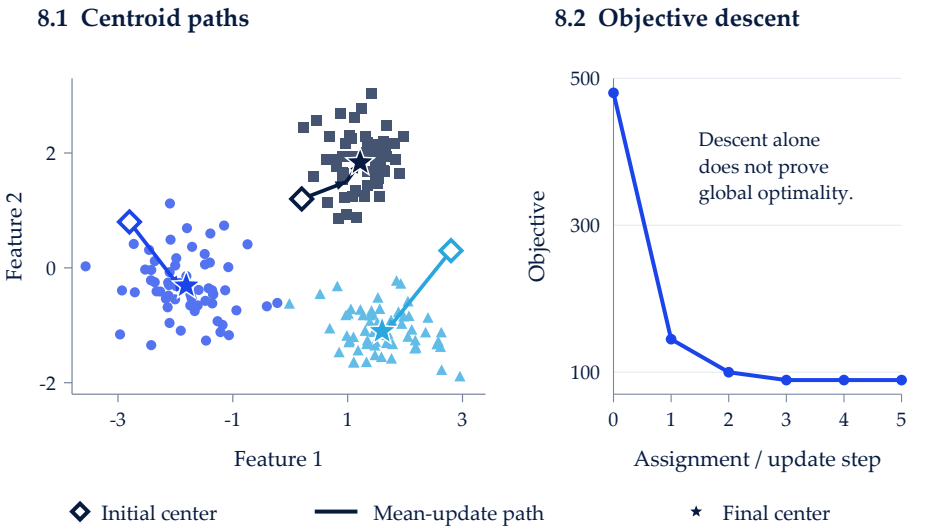


Figure 8.1. Explicit Lloyd iterations alternate nearest-centroid assignment and cluster-mean updates from one initialization. Paths show centroid movement; stars show final centers. The final grouping is one local result and can depend on initialization.

Figure 8.2. The squared-distance objective is recorded after every assignment and every mean update in the same Lloyd run. Non-increase is asserted numerically. Monotonic improvement of this objective does not prove convergence to a global optimum or establish application meaning for the clusters.

Essential

8.2 Evaluating cluster structure without inventing labels

Smaller inertia alone cannot choose K: the globally optimal k-means objective never increases with K and reaches zero with one center per distinct point. Separate local fits need not follow that trend. An elbow suggests diminishing returns, often ambiguously. Silhouette compares average within-cluster distance with the nearest alternative-cluster average. Both are internal geometric diagnostics, not proof that clusters represent the application's intended categories.

A stability comparison must refer to the same observations; cluster names themselves are arbitrary. Use a label-permutation-invariant measure such as adjusted Rand index. For resampled fits, compare shared observations or a

common probe set under an explicit assignment rule. Vary seeds, samples or defensible preprocessing. Stable acquisition artifacts and unstable sparse genuine groups are both possible, so agreement is evidence of reproducibility, not semantic truth.

When trusted labels exist for evaluation, external measures can compare the discovered grouping with those labels without pretending that clustering trained on them. In synthetic experiments, the generating labels help illustrate when an algorithm's assumptions match the design, but results remain properties of the constructed problem. For future assignment, k-means can map a new observation to a learned centroid. Evaluate whether those assignments remain useful on later data; a partition that summarizes one snapshot may not remain appropriate after the population or measurement process changes.

Rousseeuw [8.2], Section 2, compares average within-cluster dissimilarity with the smallest average dissimilarity to another cluster—not distance to the nearest other point. Singletons receive zero by convention. Declare the distance, scaling and treatment of DBSCAN noise [8.4] before comparing scores. Excluding difficult noise points can improve a reported score while reducing observation coverage; report both. This evaluates geometry under a chosen dissimilarity, not whether clusters are meaningful external categories.

Notation for equation (8.4)

i	Observation index for the silhouette calculation.
s_i	Silhouette, between minus one and one under standard conventions.
a_i	Mean distance to other members of the assigned cluster, excluding itself.
b_i	Smallest mean distance to all members of any one alternative cluster.
$\max(a_i, b_i)$	Larger of the two mean distances, normalizing their difference.

$$s_i = \frac{b_i - a_i}{\max(a_i, b_i)} \tag{8.4}$$

For a point in a nonsingleton cluster, a_i is its mean distance to other members of that cluster, and b_i is the smallest mean distance to another cluster. Evaluation requires at least two cluster labels and fewer labels than observations. Scikit-learn assigns singleton observations a silhouette of zero; a zero denominator also receives zero rather than an undefined ratio.

Use one consistent distance metric. Singleton and zero-denominator cases require the stated convention rather than direct division.

▶ Example: A useful score with a limited claim

For groups (0, 1) and (8, 9), observation 0 has within-group mean distance one and alternative-group mean distance 8.5. Its silhouette is (8.5 minus 1)/8.5 = 15/17, about 0.882. Observation 1 gives 13/15, about 0.867; symmetry makes the overall mean about 0.875. The large scores describe separation under this metric, not two natural categories. Compare counts, scaling and resampling stability, then investigate acquisition batches and task relevance. Maximizing an internal score cannot substitute for covering relevant conditions.

Interpretation and limits

A high silhouette or stable partition does not provide missing semantic labels. Internal geometry and downstream usefulness are separate evaluation dimensions.

Recommended

8.3 Hierarchical clustering and linkage choices

Hierarchical clustering records a sequence of groupings rather than one fixed resolution. Agglomeration starts with individual observations and repeatedly merges the closest groups. The dendrogram records those merges; cutting at a height or group count gives a flat partition. It supports exploration before choosing K, but depends on metric, linkage and sample. Its tree shape does not establish an evolutionary or causal hierarchy.

Ask what makes two clusters close before reading their dendrogram. Single linkage uses the nearest cross-cluster pair, allowing thin chains; complete linkage uses the farthest pair, favoring bounded diameters. Average linkage averages every cross-cluster distance. Ward asks a different question: which merge least increases within-cluster squared error? It requires Euclidean geometry. Choosing linkage changes the criterion being optimized, not merely the drawing.

A dendrogram's merge height has meaning only relative to the chosen linkage and metric. Compare multiple defensible choices, inspect influential bridge points or outliers, and avoid overinterpreting tiny branches. Many hierarchical methods require substantial pairwise computation and memory, limiting sample size. Standard agglomerative estimators also do not generally provide a direct predict method for arbitrary new observations. If deployment requires assigning future inputs, define an explicit assignment rule and validate it, or choose a model whose prediction interface matches the intended workflow.

Ward [8.3] formulated hierarchical grouping through an objective-function criterion. This gives one principled linkage rule, while also illustrating why a

dendrogram's sequence of merges depends on the distance and linkage chosen rather than revealing a unique hierarchy inherent in the data.

Notation for equations (8.5), (8.6)

A, B	Two nonempty clusters as sets of observed input vectors.
x, z	Observations from the first and second clusters, respectively.
$d(x, z)$	Chosen point distance in the fitted feature representation.
$d_{single}(A, B)$	Single linkage: minimum cross-cluster point distance.
$d_{complete}(A, B)$	Complete linkage: maximum cross-cluster point distance.
$\min, \max$	Extrema over every cross-cluster pair, not just the centroids.

$$d_{\text{single}}(A, B) = \min_{x \in A, z \in B} d(x, z) \tag{8.5}$$

$$d_{\text{complete}}(A, B) = \max_{x \in A, z \in B} d(x, z) \tag{8.6}$$

Single and complete linkage use opposite extremes of cross-group distances. Their resulting hierarchies can differ substantially even with the same input metric.

The distance function d here is not a feature count. Both linkage rules must use the same input metric for a controlled comparison.

Notation for equation (8.7)

A, B	Two nonempty clusters of observed feature vectors considered for merging.
$\lvert A\rvert, \lvert B\rvert$	First- and second-cluster observation counts.
μ_A, μ_B	First- and second-cluster arithmetic mean vectors.
$\Delta_{Ward}(A, B)$	Increase in total within-cluster squared error after merging.
$\lVert\mu_A - \mu_B\rVert^2$	Squared Euclidean centroid distance in the clustering representation.

$$\Delta_{Ward}(A, B) = \frac{|A||B|}{|A| + |B|} \lVert\mu_A - \mu_B\rVert^2 \tag{8.7}$$

Ward's rule asks how much within-group squared error would increase if two Euclidean groups shared one mean. The merge cost includes their sizes as well as their mean separation; it is not simply the nearest pair of observations. A software dendrogram may plot a transformed linkage distance, so read its convention before equating height with this raw cost.

This is the unweighted Euclidean merge-cost identity. A library's dendrogram height may apply a transformation to this raw increase.

◇ Derivation 8.3: Ward linkage is an exact increase in squared distortion

Notation for this result

A, B, n_A, n_B	Disjoint cluster index sets and their positive observation counts.
μ_A, μ_B, μ	Centroid vectors before merging and the centroid of the union.
Δ	Merged within-cluster squared error minus the sum of the separate within-cluster errors.
$\|\cdot\|_2$	Euclidean norm in the fitted feature coordinates.

Assumptions. Two disjoint, nonempty clusters contain unweighted real vectors in the same Euclidean coordinate system. Each cluster is represented by its arithmetic mean. The derivation uses raw within-cluster squared-error increase, not a software-transformed dendrogram height.

The cost of merging two clusters equals the squared distance between their means multiplied by a factor determined by their sample counts.

Read the complete supporting proof . *Advanced*

Ward linkage and nearest-centroid clustering share a squared-error geometry but optimize through different greedy operations. Using an arbitrary non-Euclidean dissimilarity breaks the derivation unless an appropriate Euclidean embedding and convention are established.

▶ Example: The bridge-point problem

Place observations at 0, 1, 2, 3, 4, 5. The endpoint clouds (0, 1) and (4, 5) are joined by bridge points 2 and 3. Single linkage needs only a chain of neighboring gaps, all of length one, to connect everything. Complete linkage compares the farthest members, so its final merge height is five. The different heights reflect different definitions of separation, not proof that the bridge is meaningful or contaminated. Investigate its provenance before removing it and report both metric and linkage.

Interpretation and limits

A dendrogram is an algorithm-dependent hierarchy, not automatic evidence of causal ancestry or a universally correct grouping resolution.

Essential

8.4 DBSCAN and density connectivity

In DBSCAN, only core points connect dense regions. Count indexed observations within epsilon, including the query itself. Meeting the minimum count makes a point core; a noncore neighbor of a core point is border; otherwise it is noise. Join connected core neighborhoods, then attach border points. Border points cannot bridge separate core components. Duplicate coordinates still count separately, so sampling frequency affects the density criterion.

This construction can identify nonconvex groups and does not require specifying their count. However, a single radius can struggle when meaningful groups have very different densities. Scaling, distance choice, sample size, and sampling frequency all alter neighborhood counts. Inspect sensitivity to epsilon and the minimum count rather than choosing settings to obtain a desired story. A nearest-neighbor distance plot can help propose radius candidates, but it does not certify an objective threshold between normal and abnormal observations.

DBSCAN's noise label denotes geometry, not a confirmed fault: a legitimate rare mode may be noise, while dense abnormal or drifting behavior may join a cluster. Standard DBSCAN clusters the supplied batch and has no general fitted predict interface. Refitting on a new batch changes the reference, potentially relabeling old and new observations. Diagnose against the intended reference before treating labels as normality judgments.

Ester and colleagues [8.4] introduced DBSCAN's density-based construction and explicit treatment of noise. Connectivity through core points distinguishes its clusters from convex k-means regions, while the neighborhood radius and minimum count define the density scale being investigated. Ankerst and colleagues [8.8] introduce OPTICS as an ordering that exposes density-based clustering structure across neighborhood scales. It is a useful extension when one DBSCAN radius is inadequate, not evidence that parameter choice or substantive interpretation disappears.

Notation for equations (8.8), (8.9)

x	Fitted observation whose density and core status are assessed.
n, i, x_i	Fitted-observation count, index and indexed feature vector.
$d(x, x_i)$	Chosen distance from the assessed observation to indexed observation i.

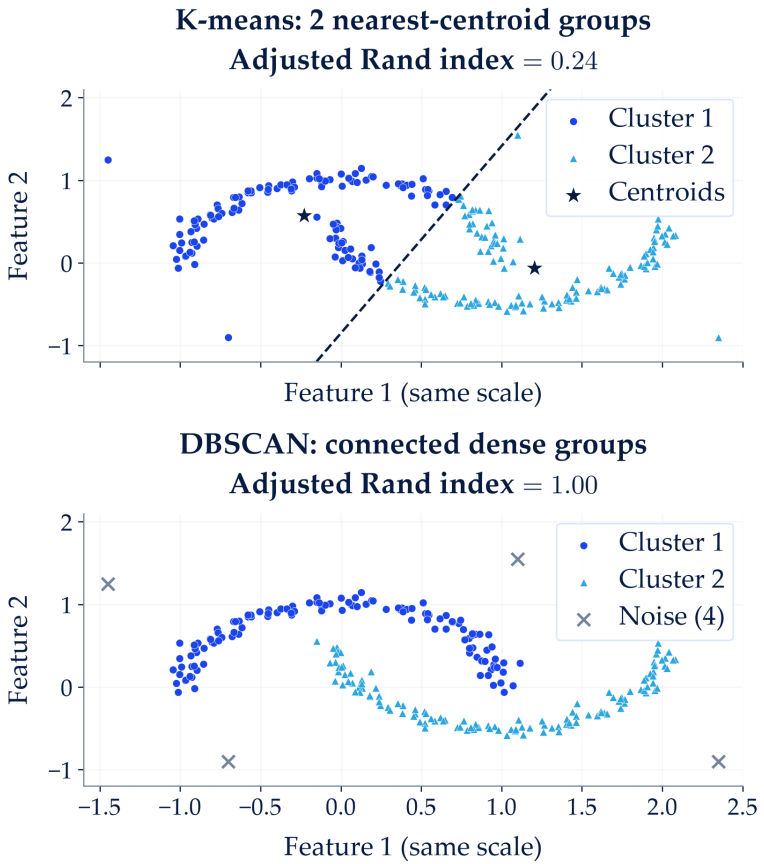


Figure 8.3. Compare assignments at identical coordinates and axis limits: 250 synthetic noisy moons (seed 31), plus four distant observations. K-means (K=2, 10 initializations) imposes convex nearest-centroid regions cutting across the curves; stars mark centers. DBSCAN (radius 0.18, minimum 5 neighbors including self) follows dense curves, marking 4 noise points. Adjusted Rand index on only the original 250 known moon labels is 0.24 for K-means and 1.00 for DBSCAN. These teaching labels are ordinarily unavailable; matching them does not establish application meaning.

$N_\epsilon(x)$	Fitted indices at distance at most epsilon from the assessed point.
$\lvert N_\epsilon(x)\rvert$	Neighborhood count, including itself and separately counted duplicates.

m	Positive integer minimum neighborhood count; min_samples in unweighted scikit-learn.
$\iff$	If and only if: the count condition exactly defines core status.

$$N_\epsilon(x) = \{i \in \{1, \dots, n\} : d(x, x_i) \le \epsilon\} \tag{8.8}$$

$$x \text{ core} \iff |N_\epsilon(x)| \ge m \tag{8.9}$$

Count indexed observations inside the epsilon neighborhood, not distinct coordinate values. Coincident observations count separately, and a fitted query x counts its own index. The point is core when this count reaches min_samples, denoted m. Epsilon sets the distance scale; the count condition determines whether the neighborhood is dense enough under that chosen scale.

This is the unweighted DBSCAN definition. Restricting neighborhoods to indexed observations prevents confusing a finite neighbor count with all points in a continuous ball.

◇ Proposition 8.4: Density connectivity is connectivity of core neighborhoods

Notation for this result

n	Positive count of indexed fitted observations, retaining repeated coordinates as distinct observations.
x_i, x_j, i, j	Indexed fitted observations and their indices from one through the sample count n.
d, ϵ, m	Symmetric distance metric, positive radius, and positive minimum-neighborhood count.
$N_\epsilon(i), V$	Neighbor index set of observation i and set of core observation indices.
$\lvert \cdot \rvert$	Number of indices in a finite set, including separate duplicate observations.
E	Undirected edges of the core-point graph; each edge joins distinct neighboring core indices.

Assumptions. The fitted observations form an indexed finite collection, so repeated coordinates remain distinct observations. Neighborhoods use a fixed symmetric metric and radius, include the point itself, and compare counts with a positive integer minimum.

DBSCAN core clusters are connected components of the graph joining neighboring core observations. Noncore border points can attach to those components but cannot propagate a path that merges two core components.

Proof *Recommended*

$$N_\epsilon(i) = \{j \in \{1, \dots, n\} : d(x_i, x_j) \leq \epsilon\} \tag{8.T12}$$

$$V = \{i : |N_\epsilon(i)| \geq m\} \tag{8.T13}$$

$$E = \{\{i, j\} : i, j \in V,\ i \neq j,\ d(x_i, x_j) \leq \epsilon\} \tag{8.T14}$$

A density-reachability path can continue only from a core observation. Each step leaves a core point and enters its neighborhood, so every intermediate point must also be core to permit another step. The final point may be noncore, because no further departure is needed. This is why core-to-core connectivity defines the components while border points can attach without linking components together.

A core-graph path is a valid density-reachability path because every departing point is core. Thus both constructions give the same connected core components. A neighboring noncore point can attach as a border point, but cannot continue the path into another component. If it touches two components, they remain separate; traversal order can decide which receives the border point. Ambiguous attachment is not a core-to-core bridge.

For a noncore observation, ask whether any core observation is its neighbor. If none is, it is noise even when other noncore observations are nearby; noise is not synonymous with isolation. If core neighbors belong to different components, its border assignment can be ambiguous. Core components are determined by the graph, but the geometric definitions need not uniquely assign every border point.

□

DBSCAN can follow curved dense groups because graph paths need not stay in a convex cell. It cannot infer whether a dense group is healthy or faulty, and a sparse valid operating mode may be labeled noise. A connected chain of border points alone must not merge separate dense cores.

▶ Example: A novel dense group is not noise

Use one-dimensional observations (0, 0.1, 0.2) and (10, 10.1, 10.2), radius 0.25, and minimum count three including self. Every observation sees all three members of its own group and none of the other: all six are core, producing two clusters and no noise. The second group can nevertheless be a new regime. Adding one legitimate observation at 5 gives noise because it has no core

neighbor. Thus neither a cluster label certifies normality nor a noise label certifies fault; investigate timestamps and provenance against a fixed reference.

Interpretation and limits

DBSCAN noise does not mean confirmed anomaly or fault, and a dense new regime may receive a normal cluster label. Clustering and reference-based monitoring answer different questions.

Algorithm 8.2. DBSCAN by indexed neighborhood expansion

Source and adaptation. Adapted from the DBSCAN neighborhood-expansion procedure of Ester and colleagues [8.4], Section 4. This version explicitly distinguishes visited and assigned states and counts repeated observations by index.

Notation and assumptions

$x_1, \dots, x_n, d, \epsilon, m$	Indexed observations, symmetric metric, positive radius and positive integer minimum count.
$N_p = \{i : d(x_p, x_i) \le \epsilon\}$	Cached neighbor-index set; includes p and counts repeated rows separately.
v_p, c_p, λ_p	Visited flag, core flag, and label of index p; labels 0 and -1 mean unassigned and provisional noise.
p, q, k	Outer-scan index, queue index and current positive cluster number.
Q, H	FIFO queue in fixed order and set of indices ever queued for this cluster.
$\mathrm{Visit}, \mathrm{Start}$	Cache a point's neighborhood/core status; initialize a queue and queued set.
$\mathrm{EnqueueNew}, \mathrm{Pop}$	Append unseen neighbor indices and update H; remove the next queue index.

Visit(p) caches N_p, sets c_p to whether $|N_p| \ge m$, and marks v_p true. Start(N) returns N in the fixed queue order and $H = N$. EnqueueNew appends exactly N minus H in that order, then adds those indices to H. Pop removes one index. Previously assigned cluster labels are never overwritten. The initial state uses all-false flag vectors and an all-zero label vector.

Algorithm 8.2

Input: Finite indexed observations and symmetric distance metric.
Fixed observation/queue order. $\epsilon > 0, \quad m \in \{1, 2, \ldots\}$
Output: Cluster/noise labels and core flags for every index.
$(\lambda_p, c_p)_{p=1}^{n}$

Require Finite inputs, valid radius/count and fixed traversal order.
$(v, c, \lambda, k) \leftarrow (\mathbf{0}, \mathbf{0}, \mathbf{0}, 0)$
For Every p in the fixed observation order.
    Continue if $v_p = \text{true}$
    Visit $\text{Visit}(p)$
    If $c_p = \text{false}$
        Label $\lambda_p \leftarrow -1$
        Continue Outer scan.
    End if
    $k \leftarrow k + 1$
    $\lambda_p \leftarrow k$
    $(Q, H) \leftarrow \text{Start}(N_p)$
    While $Q \neq \varnothing$
        $q \leftarrow \text{Pop}(Q)$
        Visit if needed $\neg v_q \Rightarrow \text{Visit}(q)$
        Expand core only $c_q \Rightarrow \text{EnqueueNew}(Q, H, N_q)$
        Assign if eligible $\lambda_q \in \{0, -1\} \Rightarrow \lambda_q \leftarrow k$
    End while
End for
Return (λ, c)

Why this works. Visited and labeled are different states: an earlier noise observation can become a border member without expanding the queue. Only core neighborhoods propagate connectivity. Each point is visited once and queued at most once per cluster, so the finite procedure terminates. Ambiguous border membership follows traversal order. Noise is a geometric label, not evidence of a fault.

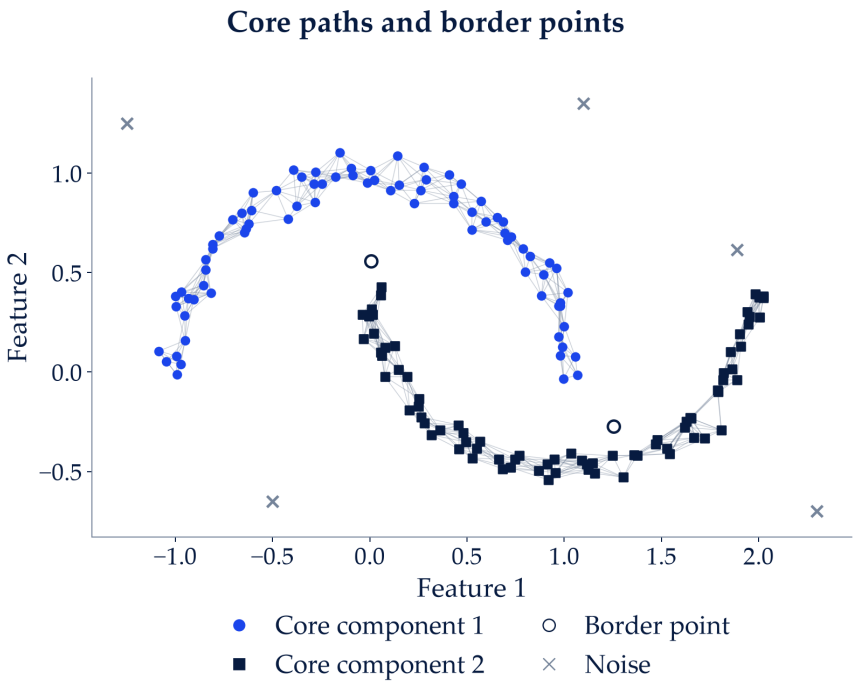


Figure 8.4. DBSCAN neighborhoods use radius 0.22 and minimum count six, including the observation itself. Edges connect core observations only. Hollow markers are border points; slate crosses are noise. Core connectivity can follow curved groups without identifying a meaningful physical category.

Essential

8.5 Anomaly scores, thresholds, and drift

First decide what the reference data represent. Novelty detection learns mainly from data believed to show acceptable behavior, then scores future cases. Outlier detection seeks unusual cases within data that may already contain contamination. Both can use distance, density, reconstruction or isolation-based scores. The distinction changes training, score interpretation and whether new observations should trigger refitting; unusualness is always relative to a reference or learned normality.

A score becomes an alert only after selecting a threshold. If the reference is reasonably representative, a held-out baseline subset can set an empirical upper score quantile corresponding to a desired reference alert fraction. That fraction is not a guaranteed future false-positive rate, especially under dependence or

drift. With verified event labels, evaluate precision, recall, and operational costs. Without them, inspect alert volume, stability, and expert-adjudicated samples while acknowledging that geometric rarity is not the same as confirmed harm.

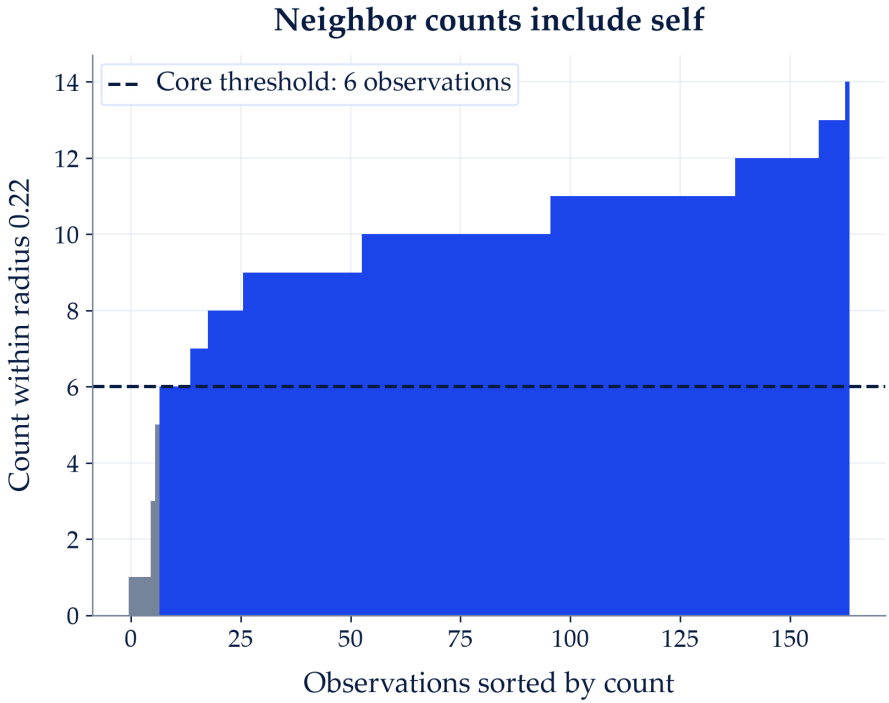


Figure 8.5. Direct counts within radius 0.22, sorted by size. A point is core exactly when at least six observations including itself lie in its neighborhood. The manually computed condition is checked against the fitted DBSCAN core indices. Low-count border and noise points are not both automatically outliers.

Drift concerns a distribution changing over time, not necessarily an isolated extreme observation. Input drift changes the input distribution; covariate shift additionally assumes the conditional outcome law stays fixed. Prevalence or that conditional law may also change. Compare missingness, units, ranges and scores with a saved reference, then use delayed labels to assess prediction quality. Define an investigation owner and an update rule before automatically incorporating alerts: refitting on all alerted data may normalize the change you meant to investigate.

Liu, Ting, and Zhou [8.5] introduce isolation-based anomaly scoring; Chandola, Banerjee, and Kumar [8.6] distinguish anomaly-detection settings. Breunig and colleagues [8.9], Definitions 5–7, define local outlier factor by comparing local reachability densities with those of neighboring observations. The neighborhood includes distance ties and can exceed the requested count. A high factor indicates relative local sparsity, not a calibrated probability of harm. Check neighborhood scale, context, costs and representative examples before converting either kind of score into an operational alert.

Notation for equations (8.10), (8.11)

$s(x)$	Anomaly score; larger means more unusual.
a	Nominal baseline alert fraction, strictly between zero and one.
$D_{baseline,val}$	Held-out reference data for threshold estimation, not anomaly-model fitting.
x_i	Reference-observation input used to estimate the score distribution.
$\mathrm{Quantile}_{1-a}$	Empirical score quantile at one minus the alert fraction, with declared interpolation.
τ	Estimated upper score threshold for alerts.
$alert(x)$	New-input binary alert: one when its score exceeds the threshold.
$\mathbf{1}[\cdot]$	One when the condition holds; zero otherwise.

$$\tau = \mathrm{Quantile}_{1-a}\{s(x_i) : x_i \in D_{\mathrm{baseline,val}}\} \tag{8.10}$$

$$\mathrm{alert}(x) = \mathbf{1}[s(x) > \tau] \tag{8.11}$$

The score s is oriented so larger means more unusual, and a is a chosen baseline alert fraction. The threshold is estimated from held-out reference data. It does not guarantee future error rates without additional assumptions.

The braces denote the empirical collection of scores, retaining repeated values. Finite samples, ties, dependence, and drift prevent a guaranteed future alert fraction.

▶ Example: A fixed-reference monitoring experiment

For a transparent threshold calculation, suppose a frozen detector gives held-out reference scores 1 through 20, with larger scores more unusual. Linear interpolation gives a ninety-fifth-percentile threshold of 19.05. With a strict greater-than rule, one reference score exceeds it. Illustrative fresh scores (5, 10, 15, 20) trigger one alert out of four; shifted scores (20, 21, 22, 23) trigger four. These tiny constructed batches demonstrate the calculation, not future false-positive or fault-detection rates. Do not refit the detector before

comparing them. Refitting DBSCAN on a combined batch asks a different, batch-clustering question.

Interpretation and limits

Automatically retraining on all incoming alerts can absorb the very change being investigated. Baseline updates require a documented review rule and trustworthy data.

Table 8.1. Clustering assumptions determine the answer

Method	Structure favored	Parameters and caveats
K-means	Compact groups around Euclidean centroids.	Specify K; assigns every point and can split curved clusters.
Hierarchical	A nested sequence under a chosen linkage.	The linkage and distance matter; the dendrogram is not a probability model.
DBSCAN	Density-connected components with sparse noise.	Specify epsilon and minimum neighborhood count; variable densities are difficult.
Anomaly detector	A score for unusual observations under a fitted baseline.	An unusual point is not necessarily a fault; thresholds require a purpose.

Table 8.2. Chapter synthesis

Concept	Main conclusion	Engineering check
Objective	A low clustering objective supports a particular geometry.	Plot original-space structure and compare alternative assumptions.
Validation	Without external labels, internal scores do not establish semantic truth.	Use stability, domain evidence and a stated downstream use.

Concept	Main conclusion	Engineering check
Density	DBSCAN expands through core points; border points do not bridge groups.	Document whether neighborhood counts include the point itself.
Anomalies	A score and a decision threshold are distinct.	Estimate false alerts against representative baseline data.

Case Study: Discover chemical groups without choosing them from cultivar labels

Problem and data

Dataset: Wine Recognition. See its first-use portrait [8.10].

The Wine measurements allow an unsupervised analysis whose external labels are available but deliberately hidden. Can a small set of chemical groups be selected from geometry alone, and does the answer depend on measurement scales? This case prevents cultivar information from entering splitting, scaling, or cluster selection. The final label comparison is reserved for interpretation, while a separate distance threshold demonstrates why unusual observations must not automatically be called faulty products.

Experimental design

Split row indices without stratification by cultivar and fit a standardizer on fitting rows only. Compare two, three, and four centroid groups using validation silhouette in the same standardized space. Freeze the selected model, compare repeat-seed stability, and fit a raw-scale control with the same chosen group count. Set a distance-alert threshold from validation scores. Reveal final cultivar labels only after these choices, then report adjusted Rand agreement, group counts, and the empirical alert fraction.

Measured Reference Run

A label-blind split used 106 fitting, 36 validation and 36 final wines. Fitting-only standardization defined the geometry. K-means compared two, three and four groups with ten initializations each; validation silhouette selected two. Cultivar labels remained sealed until final interpretation. A raw-scale control used the same selected group count. External agreement below is adjusted Rand index (ARI), not classification accuracy.

Table 8.3. Measured case-study evidence

Final representation	Groups	Count	Cultivar ARI
Standardized	2	36	0.43
Raw-scale control	2	36	0.102

Two groups maximized validation silhouette, 0.322; three additional seeds reproduced the partition. This checks stability under those starts, not cultivar recovery. Partial cultivar agreement and the changed raw-scale partition show that repeatable optimization can reflect the chosen geometry rather than scientific categories.

Failure Analysis And Limits

Optimization stability did not establish recovery of the three known cultivars: the label-free criterion preferred two groups. A frozen validation-distance threshold flagged six final wines, but these alerts are not confirmed defects. The two-component plot retained only 0.559 of fitting-set standardized variance, while clustering used all thirteen coordinates; visual separation cannot replace full-space evidence.

Historical saved run. Preserve the label-blind split, fitting scaler, fixed initializations and empirical alert threshold. The small collection does not establish future-production anomaly detection.

▶ Example: inspecting the real data

The first fitting wine has proline 1,105. Its fitted mean 737.632 and population standard deviation 324.897 give (1,105 minus 737.632)/324.897, about 1.131. Alcohol 13.05 becomes about 0.081 using unrounded fitted values. Raw magnitudes therefore do not alone determine either feature's distance contribution.

Dataset source and documentation

Companion: Week 08, Real-Data Case Study notebook. Reproduce the declared run, then investigate a changed condition. No fixed benchmark score is promised.

A Controlled Learning Cycle

The label-blind Wine case distinguishes geometric selection from cultivar agreement and confirmed faults. Week 8 notebooks audit centroid updates, ambiguous borders, dense novelty and selection without labels. The Iris laboratory exposes Lloyd, DBSCAN and linkage mechanisms. Record a baseline and one change with metric, data roles and explanation; retain the disclosed assignment extensions below and do not equate stable clusters with biological classes or anomalies with faults.

Matched practice and simulator protocol · Packaged edition; launch instructions in the guide.

Laboratory 08 | Clustering And Anomaly Detection

Lloyd updates, DBSCAN roles and fitted hierarchical linkage.

Browser laboratory dataset: Iris. See its first-use portrait [8.11].

Lloyd updates, DBSCAN neighborhoods and a fitted dendrogram expose different definitions of structure in unlabeled Iris petal geometry.

Open Laboratory 08 | Use the live view for readable controls.

Transfer: Thirteen-feature Wine clustering → Two-feature Iris petal geometry. No verified anomaly detector; held-out hierarchy/DBSCAN assignments use disclosed extensions.

Chapter Capability

Compare centroid, linkage, and density-based structure, explain rejected points, and assess stability without selecting settings from hidden labels.

Week 08: separate practice, solutions and assessment

References

[8.1] Lloyd, S. P. (1982). Least Squares Quantization in PCM. IEEE Transactions on Information Theory 28(2), 129-137. Source / DOI

[8.2] Rousseeuw, P. J. (1987). Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. Journal of Computational and Applied Mathematics 20, 53-65. Source / DOI

[8.3] Ward, J. H., Jr. (1963). Hierarchical Grouping to Optimize an Objective Function. Journal of the American Statistical Association 58(301), 236-244. Source / DOI

[8.4] Ester, M.; Kriegel, H.-P.; Sander, J.; Xu, X. (1996). A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, 226-231. Source / DOI

[8.5] Liu, F. T.; Ting, K. M.; Zhou, Z.-H. (2008). Isolation Forest. Proceedings of the IEEE International Conference on Data Mining. Source / DOI

[8.6] Chandola, V.; Banerjee, A.; Kumar, V. (2009). Anomaly Detection: A Survey. ACM Computing Surveys 41(3), Article 15. Source / DOI

[8.7] Arthur, D.; Vassilvitskii, S. (2007). k-means++: The Advantages of Careful Seeding. Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms. Source / DOI

[8.8] Ankerst, M.; Breunig, M. M.; Kriegel, H.-P.; Sander, J. (1999). OPTICS: Ordering Points To Identify the Clustering Structure. Proceedings of the ACM SIGMOD International Conference on Management of Data, 49-60. Source / DOI

[8.9] Breunig, M. M.; Kriegel, H.-P.; Ng, R. T.; Sander, J. (2000). LOF: Identifying Density-based Local Outliers. Proceedings of the ACM SIGMOD International Conference on Management of Data, 93-104. Source / DOI

[8.10] Aeberhard, S.; Forina, M. (1992). Wine. UCI Machine Learning Repository. Source / DOI

[8.11] Fisher, R. A. (1936). Iris. UCI Machine Learning Repository. Source / DOI

Advanced

Ward Linkage Is An Exact Increase In Squared Distortion

Supporting proof for Section 8.3.

Notation For This Result

A, B, n_A, n_B	Disjoint cluster index sets and their positive observation counts.
μ_A, μ_B, μ	Centroid vectors before merging and the centroid of the union.
Δ	Merged within-cluster squared error minus the sum of the separate within-cluster errors.
$\|\cdot\|_2$	Euclidean norm in the fitted feature coordinates.

Assumptions. Two disjoint, nonempty clusters contain unweighted real vectors in the same Euclidean coordinate system. Each cluster is represented by its arithmetic mean. The derivation uses raw within-cluster squared-error increase, not a software-transformed dendrogram height.

The cost of merging two clusters equals the squared distance between their means multiplied by a factor determined by their sample counts.

□ Derivation *Advanced*

$$\mu = \frac{n_A \mu_A + n_B \mu_B}{n_A + n_B} \tag{8.T9}$$

Within the first cluster, expand distance to the merged mean around its own mean. The cross term is zero because within-cluster deviations sum to zero. Repeat for the second cluster and add.

$$\Delta = n_A \|\mu_A - \mu\|_2^2 + n_B \|\mu_B - \mu\|_2^2 \tag{8.T10}$$

Subtracting the weighted mean expresses each displacement as the opposite cluster's fraction times the difference between original centroids. Factoring the common squared distance gives the merge cost.

$$\Delta = \frac{n_A n_B}{n_A + n_B} \|\mu_A - \mu_B\|_2^2 \tag{8.T11}$$

Selecting the pair with the smallest value is therefore the greedy merge with the smallest immediate increase in total squared distortion. This is not generally the closest pair of centroids because counts also matter. Nor does the sequence of greedy merges solve the best partition for every possible cluster count.

□

Ward linkage and nearest-centroid clustering share a squared-error geometry but optimize through different greedy operations. Using an arbitrary non-Euclidean dissimilarity breaks the derivation unless an appropriate Euclidean embedding and convention are established.

Return to the chapter and its references

CHAPTER 09

Probability Calibration And Model Diagnosis



Reading path

Start with Essential. The other levels offer more depth.

Essential
9.1 9.2 9.3

Recommended
9.4 9.5

Advanced
Proofs 9.2 9.4 9.5

Worked example Lab task

Two models with similar classification decisions can assign very different probabilities. Investigate the probabilities themselves and the mistakes hidden by an aggregate score.

What You Will Be Able To Do

- Distinguish probability calibration from the information retained by a predictor using Brier score and its population decomposition.

- Estimate and fit calibration maps using independent scores, with count-aware interpretation of reliability diagrams.

- Diagnose slice-specific errors and predictive dependence without confusing an association with a cause.

- Separate input, prevalence, and conditional shifts, and state precisely when a conformal coverage guarantee applies.

Before you begin: Classification probabilities and confusion matrices; Data splitting and pipelines; Tree ensembles and regularized classification.

Track which rows fit the model and calibrator, then interpret reliability counts before uncertainty guarantees.

Algorithm scope: Calibration and conformal orchestration: Algorithm 9.1 delegates its optimizer or monotone fit; Algorithm 9.2 explicitly constructs residual order-statistic intervals. Diagnostic associations are not causal identification.

Shared chapter notation

b	In reliability summaries, the index of a nonempty probability bin.
B_b	Evaluation-index set assigned to probability bin b under the declared rule.
n_b	Positive bin-b observation count; empty probability bins are omitted.
g	Label of a declared evaluation slice.
S_g	Nonempty observation-index set for evaluation slice g.
n_g	Observation count in evaluation slice g.

Chapter 3 separated classification scores, probabilities and decisions; Chapter 4 protected evaluation from selection. For a fitted predictor, we now examine what numerical confidence means and what evidence can explain failure. These questions matter when probabilities prioritize cases, quantify uncertainty or compare risks, rather than merely rank observations.

Proper scores define the reward; calibration compares predictions with outcome frequencies; slices expose aggregate blind spots; feature perturbations investigate dependence; and shift analysis separates warnings from demonstrated degradation. Conformal prediction then states the assumptions supporting a finite-sample guarantee.

Essential

9.1 Proper scores and probability information

A probability of 0.8 claims more than a high ranking: across comparable cases receiving that prediction, it describes an outcome frequency. Chapter 3's ranking and threshold tools cannot by themselves test this claim. This chapter evaluates the probability values using scores whose population optimum is truthful probability reporting, while retaining the earlier tools for ranking and decisions.

Brier score is the mean squared difference between a positive-class probability and a zero-or-one outcome. Under the binary convention below it lies between zero and one. Its conditional expectation separates into irreducible outcome uncertainty and squared error in the reported probability. The proof makes the minimum explicit. Log loss, derived in Chapter 3, is another strictly proper score, with a much stronger penalty for assigning near-zero probability to an outcome that occurs.

Calibration asks whether outcomes match reported probabilities; a proper score additionally rewards retaining useful risk information. A constant prevalence

forecast may be calibrated while assigning the same risk to very different cases. The next decomposition separates this lost resolution from calibration error and outcome uncertainty. Compare scores on the intended population and declare weighting, because changing prevalence also changes the constant reference forecast and its score.

Compare with a prevalence-only baseline and inspect the cases contributing the largest losses. State the binary or multiclass scoring convention and any clipping rule. Clipping makes logarithms finite but changes the evaluated probability claim; it is not merely an invisible computational detail. Interpret finite-sample score differences with uncertainty.

Gneiting and Raftery [9.1] connect proper scoring rules to the information conveyed by a probability forecast. This perspective motivates separating calibration from resolution: a constant forecast can match the overall event frequency without distinguishing low-risk from high-risk cases.

Notation for equation (9.1)

Brier	Empirical mean squared error of positive-class probabilities under the binary convention.
n, i	Evaluation sample count and index from one through n.
p_i	Positive-class probability predicted without fitting on this evaluation outcome.
y_i	Observed binary outcome, encoded as zero or one.

$$\mathrm{Brier} = \frac{1}{n}\sum_{i=1}^{n}(p_i - y_i)^2 \tag{9.1}$$

This binary convention scores only the positive-class probability. Summing errors over both binary class coordinates would double it.

◇ Proposition 9.1: Brier score targets the true conditional probability

Notation for this result

X, Y, x	Random input, binary outcome, and a fixed input value.
q	True conditional positive-class probability at that input.
$\Pr$	Probability under the evaluation population.
p	Candidate positive-class probability at the fixed input.
$\mathbb{E}$	Expectation under the evaluation population.

Assumptions. The outcome is binary and every expectation refers to a fixed evaluation population. At a given input, the candidate prediction is a probability between zero and one.

Expected squared probability error is uniquely minimized by the true conditional positive-class probability. Its excess over the optimum is exactly the squared probability error.

□ **Proof** *Recommended*

$$q = \Pr(Y = 1 \mid X = x) \tag{9.T1}$$

A binary outcome equals its own square. Consequently its first and second conditional moments both equal the positive-class probability. Expand the squared prediction residual and substitute these two moments. The resulting quadratic separates what is random in the outcome from the proposed probability, which is fixed while conditioning on the input.

$$\mathbb{E}[(p - Y)^2 \mid X = x] = p^2 - 2pq + q \tag{9.T2}$$

Complete the square to separate a prediction-dependent discrepancy from uncertainty that cannot be removed by choosing a different report at the same input.

$$p^2 - 2pq + q = (p - q)^2 + q(1 - q) \tag{9.T3}$$

The first term is nonnegative and is zero exactly at the true probability. Averaging this conditional identity over inputs proves strict properness for a measurable probability function, up to events of probability zero. Boundary probabilities are included without a logarithmic limit. The corresponding log-loss result and its Bernoulli likelihood derivation are established in Section 3.2, so they need not be repeated to use the two scores side by side.

□

A proper score discourages false confidence but is not a pure calibration statistic. The term depending on the true conditional probability can differ across populations and across the information available to a predictor.

▶ Example: Perfect calibration need not preserve useful information

Consider equally common groups with outcome rates 0.2 and 0.8. A constant 0.5 forecast is calibrated after pooling them and always has squared loss 0.25. Reporting the group-specific probabilities is calibrated too, but its expected loss in either group is 0.2 times 0.8 = 0.16. The improvement, 0.25 minus 0.16 = 0.09, comes from preserving risk differences, not repairing calibration. Thus two diagonal reliability curves can conceal different predictive usefulness. This is a population calculation, not a sample result.

Interpretation and limits

Do not call Brier score or log loss a pure calibration error. Their values also reflect the task's outcome uncertainty and the information retained by the predictor.

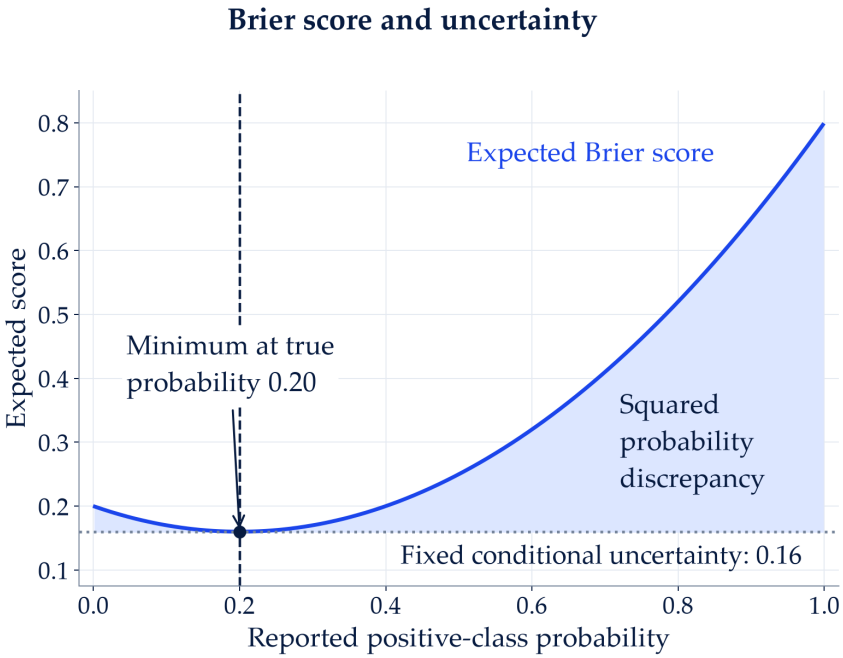


Figure 9.1. Exact expected Brier score for a Bernoulli outcome with probability 0.20. Risk separates into fixed conditional uncertainty and the shaded squared discrepancy between reported and true probability. The optimum is the true probability, independently of a later classification threshold.

Essential

9.2 Estimating and correcting calibration

Calibration compares what probabilities promise with what outcomes occur. Treat the fitted predictor's output on a fresh case as a random probability. Population calibration requires the conditional outcome frequency at each reported probability to equal that probability. A reliability diagram estimates a coarser, binned relationship: compare each nonempty bin's mean prediction with its positive fraction, and display the supporting count.

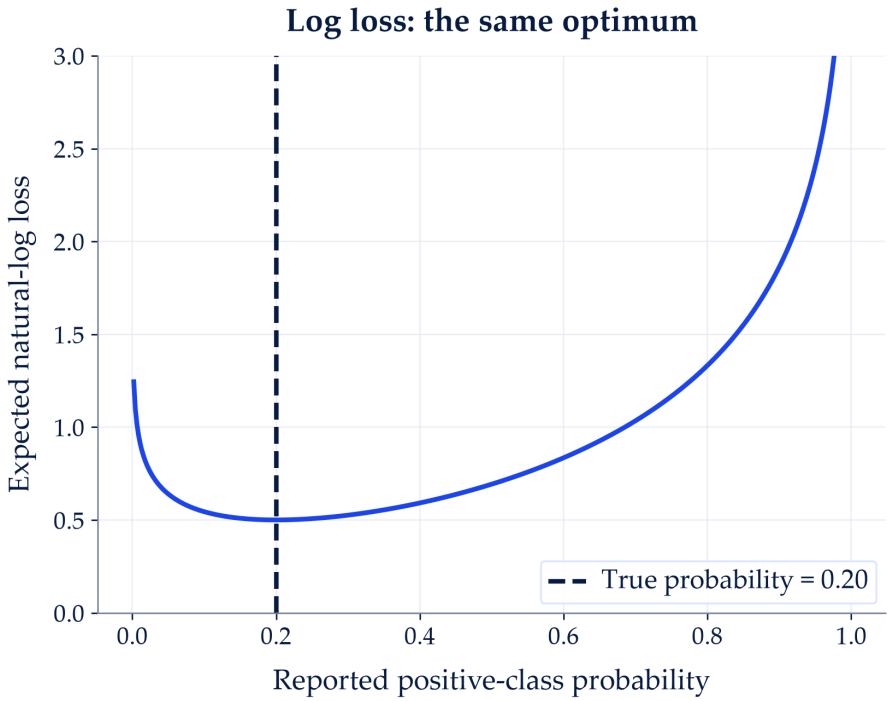


Figure 9.2. Exact expected natural-log loss under the same Bernoulli probability 0.20. Its unique interior optimum is the true probability, and confident errors are heavily penalized. This proper scoring objective assesses probability quality rather than only ranking.

Choose reliability bins before judging the final curve. Equal-width bins preserve fixed probability intervals but can leave sparse tails; equal-count bins trade that for changing interval widths. Show occupancy and score distributions. Expected calibration error depends on the bins. Opposite deviations can cancel within one bin, and choosing bins for visual agreement adds selection bias.

Recalibration learns a map from the already fitted model's score to a probability. A sigmoid map estimates a slope and intercept; isotonic regression fits a nondecreasing stepwise map. The former is compact but restrictive, while the latter can capture more shapes at the cost of greater estimation variance. To preserve strict ranking, use a positive sigmoid slope. Flat isotonic regions or zero slopes can tie scores. A fitted map may improve or worsen

new-data scores; the population projection theorem is not a guarantee for a finite estimated map.

Separate three roles: base fitting creates scores, calibration labels fit the map, and untouched evaluation assesses the frozen pair. For cross-fitting, each calibration score must exclude its own label from base fitting. Choose the mapping family on development evidence, not final evaluation, and assess the intended population; a previously accurate map need not transfer after that population changes.

Niculescu-Mizil and Caruana [9.2] compare the probability behavior of different learners and post-hoc corrections. Guo and colleagues [9.3] examine calibration in modern neural networks and temperature scaling; both works motivate fitting a calibration map on data separate from the final assessment. Kumar, Liang, and Ma [9.8] show how binning can conceal calibration error and develop a scaling-binning calibration procedure. Their analysis reinforces the need to distinguish the fitted map, the diagnostic binning rule and the finite data available to evaluate either.

Notation for equation (9.2)

Y	Fresh zero-or-one outcome from the evaluation population.
P	Random positive-class probability produced by the fixed fitted predictor on a fresh input.
$\mathbb{E}[Y \mid P]$	Conditional positive-outcome frequency among cases receiving the same predicted probability.

$$\mathbb{E}[Y \mid P] = P \tag{9.2}$$

Population calibration is a conditional-mean statement for the fixed predictor's random output on fresh cases.

This is a population equality holding almost surely, not an exact equality for each empirical bin.

Notation for equation (9.3)

i	Observation index restricted to the specified bin.
p_i	Positive-class probability for evaluation observation i.
$\overline{p}_b$	Mean predicted probability in bin b.

$$\overline{p}_b = \frac{1}{n_b} \sum_{i \in B_b} p_i \tag{9.3}$$

The horizontal coordinate of one nonempty reliability bin is its mean reported probability.

Notation for equation (9.4)

y_i	Observed zero-or-one outcome for observation i.
$\widehat{q}_b$	Observed positive fraction in the bin.

$$\widehat{q}_b = \frac{1}{n_b} \sum_{i \in B_b} y_i \tag{9.4}$$

The vertical coordinate is the observed positive fraction in that same bin; its uncertainty depends on the bin count.

Notation for equation (9.5)

x	Input to the frozen base predictor.
$s(x)$	Base scalar decision score or documented probability transformation.
a, b	Fitted calibration slope and intercept; b is not a bin index.
σ	Logistic sigmoid defined in Chapter 3.
$p_{\text{cal}}(x)$	Positive-class probability after the separately fitted calibration map.

$$p_{\text{cal}}(x) = \sigma(a\, s(x) + b) \tag{9.5}$$

The sigmoid from Chapter 3 maps a frozen base score to a recalibrated probability using a separately fitted slope and intercept.

◇ Theorem 9.2: Population recalibration is a conditional-mean projection

Notation for this result

X, Y	Fresh input and binary outcome from the evaluation population.
$\widehat{p}, P$	Fixed fitted probability function and the random probability it reports.
$m(P)$	Population conditional positive frequency given the reported probability.
$\mathbb{E}$	Expectation under the current evaluation population.
$\bar{q}, \mathrm{Var}$	Population positive-class prevalence and variance under the evaluation distribution.

Assumptions. A fitted predictor produces a random probability between zero and one on fresh evaluation observations. All expectations refer to the same evaluation population. The calibration function in the statement is the population conditional mean, not an estimated finite-sample curve.

Replacing a probability by the outcome frequency conditional on that probability weakly improves population Brier score. The improvement is exactly the squared calibration discrepancy.

□ **Proof** *Advanced*

$$P = \widehat{p}(X) \tag{9.T4}$$

$$m(P) = \mathbb{E}[\,Y \mid P\,] \tag{9.T5}$$

Group outcomes conceptually by the original reported probability and insert that group's true conditional mean into the residual. The residual around this mean has conditional expectation zero. The remaining prediction-to-mean discrepancy is fixed within the group, so its cross product vanishes on averaging. This uses a population conditional mean, not an estimated bin average.

$$\begin{aligned}\mathbb{E}[(\,Y - P)^2] = \mathbb{E}[(\,Y - m(P))^2] \\ +\mathbb{E}[(m(P) - P)^2].\end{aligned} \tag{9.T6}$$

The decomposition compares the original forecast with ideal recalibration. Its first term is the recalibrated Brier risk; the additional squared difference is nonnegative. Why is the new forecast calibrated even if several old scores merge? Apply the tower property: average the old score-conditional outcome frequencies over cases sharing the new value. Every such frequency equals that new value, so their conditional average does too.

$$\mathbb{E}[\,Y \mid m(P)] = m(P) \tag{9.T7}$$

At a fixed reported probability, binary outcome variance is the conditional outcome frequency times its complement. Now use total variance to compare these within-probability variances with overall outcome uncertainty. The variation in conditional frequencies becomes the resolution term; the discrepancy between those frequencies and the reported probabilities becomes calibration error. Substitution gives the stated Brier decomposition.

$$\begin{aligned}\mathbb{E}[(\,Y - P)^2] = \bar{q}(1 - \bar{q}) - \mathrm{Var}(m(P)) \\ +\mathbb{E}[(m(P) - P)^2].\end{aligned} \tag{9.T8}$$

This identity explains why a constant prevalence predictor can be perfectly calibrated while discarding useful discrimination: its resolution term is zero. It also distinguishes the population conditional function from a binned reliability diagram, whose bin averages approximate a coarser relationship and fluctuate with sample size.

□

Estimated sigmoid or isotonic calibration can worsen fresh-data performance through estimation error or shift; the theorem does not promise otherwise. Fit the map without reusing base-model targets, evaluate it on separate data, and show bin counts instead of treating a visually smooth reliability curve as proof.

▶ Example: The same discrepancy with different amounts of evidence

A bin reports mean prediction 0.8 and observed positive fraction 0.6. The plug-in binomial standard error is the square root of 0.6 times 0.4 divided by the bin count: approximately 0.155 for ten independent cases and 0.015 for one thousand. The same discrepancy therefore has very different sampling precision. These rough diagnostics do not correct dependence or adaptive grouping. Separately, reporting 0.9 where the true rate is 0.8 adds (0.9 minus 0.8) squared = 0.01 to conditional Brier risk. An estimated correction still needs independent evaluation.

Interpretation and limits

Training-set reliability is not calibration evidence for fresh cases. A smooth fitted curve and a low bin-dependent summary can both conceal sparse or unrepresentative data.

Algorithm 9.1. Calibrate a frozen predictor

Source and adaptation. The separated sigmoid/isotonic calibration workflow follows Niculescu-Mizil and Caruana [9.2], Section 2. Fixed bins, solver checks and score-domain rules are specified here; the optimizer or monotone solver remains delegated.

Notation and assumptions

D_b, D_c, D_e	Disjoint index sets identifying base-training, calibration and untouched evaluation pairs; binary outcomes use 0/1.
$\mathcal{T}, s, \mathcal{G}, g$	Fitted preprocessing, scalar base score, fixed calibration family and fitted probability map.
u_i, p_i, q_i, y_i	Calibration score, raw probability if available, calibrated probability and binary outcome.
$\mathrm{FitBase}, \mathrm{FitCal}, \mathrm{Scores}$	Training-only base fit; constrained calibration fit; declared proper-score evaluator.
$\mathcal{S}, \Delta$	Proper-score vectors (raw/calibrated subscripts), and calibrated-minus-raw differences.

$I_j, B_j, n_j, \bar{q}_j, \bar{y}_j$ Fixed probability bin, its evaluation indices/count, mean probability and observed positive fraction.

$\mathcal{E}$ Saved table of nonempty-bin counts, means and frequencies.

For isotonic calibration, fix the prediction rule outside the calibration-score range before evaluation, for example clipping to the nearest endpoint. Require a positive sigmoid slope for strict ranking. A zero slope or an isotonic plateau can create ties. FitCal uses the fixed budget/tolerance. Sigmoid fitting
needs both classes or a declared regularized/smoothed alternative. Scores applies fixed endpoint/clipping rules and rejects invalid probabilities. The vector p contains frozen-base probabilities on D_e when available. Initialize the bin table E empty. Fixed bins partition [0,1] with declared boundary inclusion. Failed Require returns a diagnostic.

The double-bar list operator appends the right-hand entries to the left-hand list.

Algorithm 9.1

Input: Disjoint data roles respecting declared group/time/independence boundaries.

Fixed base procedure, calibration family, solver controls and bin edges.

Output: Frozen base/calibrator, solver status, proper scores and counted reliability bins.

Require Valid data roles, binary labels and adequate calibration outcomes.

Fit and freeze Base-training data only. $(\mathcal{T}, s) \leftarrow \mathrm{FitBase}(D_b)$

Calibration inputs only. $u_i \leftarrow s(\mathcal{T}(x_i)), \quad i \in D_c$

Fit and freeze Using the fixed constrained family. $g \leftarrow \mathrm{FitCal}((u_i, y_i)_{i \in D_c}; \mathcal{G})$

Require Solver success and finite probabilities in[0,1].

Untouched evaluation inputs. $q_i \leftarrow g(s(\mathcal{T}(x_i))), \quad i \in D_e$

Proper scores under the declared convention. $\mathcal{S}_{\mathrm{cal}} \leftarrow \mathrm{Scores}(y, q)$

If A raw-probability output exists.

    Same cases and scoring convention. $\mathcal{S}_{\mathrm{raw}} \leftarrow \mathrm{Scores}(y, p)$

    Compare $\Delta \leftarrow \mathcal{S}_{\mathrm{cal}} - \mathcal{S}_{\mathrm{raw}}$

End if

For Each fixed bin j.

    $B_j \leftarrow \{i \in D_e : q_i \in I_j\}$

Algorithm 9.1 (continued)

Continue if Empty bin. $|B_j| = 0$

$n_j \leftarrow |B_j|$

Average $\bar{q}_j \leftarrow n_j^{-1} \sum_{i \in B_j} q_i$

Average $\bar{y}_j \leftarrow n_j^{-1} \sum_{i \in B_j} y_i$

$\mathcal{E} \leftarrow \mathcal{E} \parallel [(n_j, \bar{q}_j, \bar{y}_j)]$

End for

Return Frozen pair, status, scores, available differences and bin table.

Why this works. Compute raw probabilities on D_e from the frozen base if it supplies them; do not treat arbitrary decision scores as probabilities. Flag a missing or undefined score comparison. Select families and decision thresholds on separate development evidence, never D_e. A zero sigmoid slope can create ties. Recalibration can worsen fresh-data scores; show every nonempty bin count.

Temperature Scaling: What Changes And What Does Not

Source and scope. Temperature scaling follows Guo and colleagues [9.3]. The finite fitting bounds and ranking counterexample state the conventions and limits used in this worked procedure.

Temperature scaling recalibrates a frozen classifier without fitting new features. It preserves the largest-logit class decision with the same tie rule, not every possible threshold decision. Let C be the number of classes, $z_j(x)$ the finite logit for class j at input x, and T a shared positive temperature. Class indices j and k range from 1 to C. The calibrated probabilities are defined by the following softmax; exp is the exponential function.

$$p_T(j \mid x) = \frac{\exp(z_j(x)/T)}{\sum_{k=1}^{C} \exp(z_k(x)/T)} \tag{9.C1}$$

Let m be a positive integer. A calibration sample contains m observations (x_i, y_i), with i indexing observations and y_i the observed class index. It is not used to train the base classifier. Define J as its mean negative log likelihood, using natural logarithms. Before fitting, choose finite positive bounds T-min and T-max, with T-min smaller than T-max. These bounds make the search

compact; continuity guarantees that a minimizer exists, though it need not be unique.

$$J(T) = -\frac{1}{m}\sum_{i=1}^{m}\log p_T(y_i \mid x_i) \tag{9.C2}$$

$T_{\min}, T_{\max}, \widehat{T}$ Predeclared positive lower/upper bounds and a chosen minimizing temperature, respectively.

$$\widehat{T} \in \operatorname*{argmin}_{T\in[T_{\min},T_{\max}]} J(T) \tag{9.C3}$$

Evaluate the objective with shifted log-sum-exp, not by taking logarithms of rounded probabilities. A search may optimize log-temperature instead, with matching bounds. Freeze the fitted temperature before independent evaluation. A lower calibration-set loss is not a guarantee of lower population loss, and comparing calibration methods requires a separate selection protocol.

Proof of the decision invariance. Dividing all logits by the same positive number preserves every pairwise ordering and every tie within an example. Exponentiation and the common positive denominator preserve that ordering again. Consequently the largest-logit class, with the same tie rule, is unchanged. For T above one probabilities become less concentrated; as T tends to infinity they approach 1/C. When there is a unique largest logit, its probability tends to one as T approaches zero from above.

For binary classification, let d(x) be the positive-class logit minus the negative-class logit, and let sigma be the logistic sigmoid. The positive-class probability is sigma(d(x)/T), a strictly increasing function of d(x). Thus a single positive temperature preserves binary ranking and ROC-AUC, apart from finite-precision ties. Probability losses still change, and decisions at a fixed probability threshold other than 0.5 can change too.

Temperature preserves the winning class within each example, but not every class's probability ranking across examples. For class 1, logits (2, 1.5, -10) and (2, 0, 0) give probabilities about 0.622 and 0.787 at T one; at T four they give 0.518 and 0.452. The order reverses because each example has different competing logits in its denominator. Both still predict class 1. Consequently multiclass one-versus-rest AUC need not be invariant.

Method boundary. Positive-slope binary sigmoid calibration can additionally fit an intercept; temperature scaling cannot. Isotonic calibration can introduce ties through flat intervals. None of these methods proves calibration under a new population. In the companion laboratory, identify the fitting partition and

compare probability loss and bin counts, not only unchanged classification accuracy.

Guo, Pleiss, Sun, and Weinberger (2017), On Calibration of Modern Neural Networks, ICML: post-hoc temperature scaling of fixed logits. Source

Essential

9.3 Error slices and conditional reliability

Notation for this section

A, B	Two equally common input-group labels in the example; not matrices or estimated coefficients.

A good average can hide a bad subgroup. Define an error slice with an interpretable membership rule, such as source, time period or input-quality range. Within it, recompute the same declared loss and calibration quantities. Report observation and positive counts alongside the prediction distribution. These support the interpretation; an aggregate over the full population mixture can conceal the slice's failure.

The two-group construction in Section 9.1 also diagnoses slices: a constant 0.5 forecast matches the overall frequency but not either group's rate. The theorem below expresses this cancellation through conditional expectations and quantifies the information lost by discarding the group distinction.

Decision consequences follow from the fixed policy already developed in Section 3.4. At a preselected threshold, compare counts and costs across slices without searching for a new threshold on the final test set. A calibrated score supports an optimal constant-cost decision among policies that use that score; it need not retain every decision-relevant feature. An observed subgroup discrepancy is a reason to investigate labels, measurement quality, representation, and sampling, not an automatic instruction to use group membership as a decision feature.

Label each slice check as predeclared or exploratory. Searching many small slices selects some extreme errors by chance, so confirm discovered hypotheses on fresh or reserved data. Report counts, positives, and the declared loss; account for repeated or clustered observations rather than assuming independent cases. The aim is a reproducible failure hypothesis, not a ranking of tiny groups by their worst observed score.

The probabilistic viewpoint of Gneiting and Raftery [9.1] and the coverage distinctions emphasized by Angelopoulos and Bates [9.6] help explain why an aggregate guarantee is not a subgroup guarantee. The conditional decomposition below makes the missing information explicit when reliability is examined within a slice. Hebert-Johnson and colleagues [9.9] formalize

multicalibration over a specified family of computationally identifiable subpopulations, including overlapping groups. This strengthens the evaluation target beyond overall calibration without guaranteeing reliability on every conceivable slice.

Notation for equation (9.6)

i	Index of an observation in the displayed slice.
y_i, p_i	Observed binary outcome and predicted positive-class probability.
ℓ	Fixed per-observation probability loss, such as squared probability error or binary log loss.
$\widehat{R}_g$	Empirical mean loss within the declared slice.

$$\widehat{R}_g = \frac{1}{n_g} \sum_{i \in S_g} \ell(y_i, p_i) \tag{9.6}$$

Evaluate the already chosen probability loss on a declared nonempty slice, rather than introducing a different metric for each group.

Notation for equation (9.7)

$\hat{r}_g$	Empirical mean outcome-minus-probability residual in the slice.

$$\hat{r}_g = \frac{1}{n_g} \sum_{i \in S_g} (y_i - p_i) \tag{9.7}$$

A positive mean calibration residual indicates average underprediction in the slice. Zero mean residual alone does not establish full conditional calibration.

A zero mean residual permits cancellation across probability levels and therefore does not prove calibration within the slice.

◇ Theorem 9.3: Why global calibration can hide slice-specific error

Notation for this result

Y, P	Fresh binary outcome and the fixed predictor's random reported probability.
$m(P)$	Conditional positive frequency given only the reported probability.
$\mathbb{E}$	Expectation under the evaluation population.
G	Discrete slice label of the same evaluation case.
$m_G(P)$	Conditional positive frequency when both score and slice label are known.

Assumptions. The outcome is binary. A fixed fitted predictor reports a probability on a fresh evaluation case. A discrete slice label is available for diagnostic conditioning. All conditional expectations refer to the same population; no causal interpretation or permission to use the label operationally is assumed.

The conditional outcome frequency within a probability-and-slice group can differ from the global frequency at that probability. Using the finer conditional mean weakly improves population Brier risk; the improvement equals a nonnegative conditional-mean discrepancy.

□ **Proof** *Recommended*

$$m(P) = \mathbb{E}[Y \mid P] \tag{9.T9}$$

$$m_G(P) = \mathbb{E}[Y \mid P, G] \tag{9.T10}$$

At a fixed score, first compute the outcome frequency separately within each slice. Then average those frequencies using the slice proportions among observations with that score. The tower property gives the coarser score-only frequency. The weights are conditional slice proportions, not necessarily equal weights or the slices' proportions in the whole population.

$$\mathbb{E}[m_G(P) \mid P] = m(P) \tag{9.T11}$$

Global calibration permits slice errors to cancel: only their score-conditional average must equal the reported probability. To compare risks, subtract the finer conditional mean based on score and slice, then add it back inside the residual. The new cross term vanishes because outcome minus this finer mean has zero conditional expectation. It is therefore orthogonal to every square-integrable function of score and slice, including the forecast difference.

$$\begin{aligned}\mathbb{E}[(Y - m(P))^2] = \mathbb{E}[(Y - m_G(P))^2] \\ +\mathbb{E}[(m_G(P) - m(P))^2].\end{aligned} \tag{9.T12}$$

The final term measures information gained by keeping the slice distinction and is nonnegative. With equally mixed slices, a constant probability one-half and outcome frequencies one-fifth and four-fifths, it equals nine hundredths. This is the improvement available from the true conditional means, not a promised improvement from their estimates. Small samples, adaptive slice selection and population change can prevent a fitted correction from realizing it.

This also sharpens the decision result in Section 3.4. A globally calibrated score can support a minimum-cost rule among policies restricted to that score, yet a finer input representation may retain additional risk information. The mathematical possibility of an improvement is not evidence that a particular group label is appropriate or lawful to use as a decision variable.

□

Global reliability can result from cancellation of opposite conditional errors. Treat slice analysis as evidence for a reproducible hypothesis and validate any proposed correction independently.

▶ Example: A canceled residual is still a failure pattern

Reuse the equal-mixture population from Section 9.1.
Outcome-minus-prediction residuals are 0.2 minus 0.5 = -0.3 in A and 0.8 minus 0.5 = 0.3 in B. Their equal-weight average is zero, hiding both errors. If A instead represents three-quarters of cases, the mean becomes -0.15 despite unchanged group rates. Thus a global residual reflects both conditional errors and mixture weights. These are population calculations, not observations about a real group; finite-sample slice estimates require counts, uncertainty, and independent confirmation.

Interpretation and limits

Opposite subgroup errors can cancel globally. Conversely, a large error in a small slice may be a selection artifact rather than a reproducible mechanism.

Recommended

9.4 Interpreting predictive dependence

Permutation importance asks whether a fitted predictor relies on a feature's information. Keep the predictor fixed, shuffle that feature across reference observations, and measure the change in the chosen loss. A positive increase means the perturbation harmed prediction. Repeat shuffles to measure perturbation variability. This does not measure uncertainty from refitting on new training samples.

Dependence creates two different traps. Another near-duplicate input can preserve shared signal after one feature is shuffled, understating reliance on that signal. Independent shuffling can also create impossible combinations and overstate loss. Grouped permutation uses the same row shuffle for chosen columns, preserving their internal dependence while breaking associations outside the group. Conditional perturbation instead needs a defensible conditional model. Neither procedure estimates a physical intervention's causal effect merely by perturbing a predictor's inputs.

For partial dependence, set one coordinate to a common value in every reference case, run the fitted model, then average its responses. Individual conditional expectation keeps those case-specific curves instead of averaging immediately. Opposing responses can disappear in the average, much like opposing calibration errors across slices. Both displays may query unsupported feature combinations, so inspect support and name the output plotted, such as one class probability.

The result of an inspection should suggest a testable hypothesis. A suspected shortcut might predict larger errors after a particular acquisition change; a redundant feature might explain why single-column importance is small. Check the hypothesis on relevant held-out data and against alternative explanations. Explanation plots describe the model under a chosen perturbation, whereas observed residuals describe its agreement with outcomes; neither replaces the other.

Friedman [9.4] develops partial dependence for inspecting a fitted predictor. Strobl and colleagues [9.5], Sections 2.4–2.5, show how ordinary permutation disrupts predictor dependence and propose conditional permutation to preserve it. These importance measures answer different questions: state which intervention on the inputs is being measured. Neither importance rankings nor calibrated probabilities establish a causal effect, subgroup reliability or robustness to arbitrary distribution shift without additional assumptions.

Notation for equation (9.8)

X, y	Held-out n-by-d inputs and matching length-n targets.
j	Selected feature index from one through d.
$X^{\mathrm{perm}(j)}$	Reference matrix with column j shuffled across rows.
f	Prediction procedure fixed before and after perturbation.
L	Declared aggregate loss; lower is better.
I_j	Loss increase from the displayed permutation of feature j.
n, d	Reference observation count and feature count.

$$I_j = L(y, f(X^{\mathrm{perm}(j)})) - L(y, f(X)) \tag{9.8}$$

This sign convention uses a loss for which lower is better. One permutation produces one random estimate of loss increase.

Notation for equation (9.9)

$\mathrm{PD}_j(z)$	Partial dependence of a fixed scalar model output at candidate feature value z.
j, z	Varied feature coordinate and proposed replacement value.

n, i	Reference sample count and row index from one through n.
$x_{i,-j}$	Reference row without coordinate j; remaining coordinates keep their original order.
f	Fixed scalar predictor; classification specifies one class output.

$$\mathrm{PD}_j(z) = \frac{1}{n}\sum_{i=1}^{n} f(z, x_{i,-j}) \tag{9.9}$$

Replace the selected coordinate in each reference row before averaging predictions; the remaining feature order is unchanged.

Insert z at coordinate j before prediction; unsupported combinations can result when features are dependent.

▶ Example: Testing reliance on a shared signal

In a two-row regression illustration, both sensor columns and the target take values (-1, 1). A frozen predictor averages the two sensors and initially has zero MSE. Reverse one column: both predictions become zero and MSE becomes one. Reverse both columns with the same row permutation: predictions become (1, -1) and MSE becomes four. Joint perturbation preserves sensor agreement but breaks association with the target; single-column perturbation creates combinations absent from the original rows. The contrast diagnoses dependence under a specified perturbation, not causal effects. Repeat relevant diagnostics on the source that motivated the investigation.

Interpretation and limits

A feature's permutation importance depends on the fitted function, evaluation population, loss, and perturbation. A low value does not prove irrelevance, and a high value does not prove causality.

Interactive companion | Local evidence and predictive dependence

Local attribution allocates a prediction to input features under a chosen explanation procedure. Compare the available explanations for the same prediction and inspect which contributions support or oppose it. Contrast this local question with the dataset-level permutation experiment developed here. Agreement between explanation methods is not proof that a feature has a causal effect.

Open Explainability Lab

Sign-in may be required. Suggested comparisons are in the separate teaching companion.

Recommended

9.5 Distribution shift and the limits of uncertainty guarantees

Notation for this section

A, B	The earlier counterexample's input-group labels; the shift changes their mixture or outcome frequencies.

Ask which part of the distribution changed. Covariate shift changes inputs while preserving outcomes conditional on inputs. Label shift changes class prevalence while preserving inputs conditional on class. Concept shift changes the outcome relationship at a fixed input. Probability diagnostics are population-dependent, so compare reference and later distributions explicitly. These are distributional assumptions, not diagnoses that an unlabeled dashboard can establish by itself.

An input-distribution warning therefore does not prove that prediction quality has degraded. The altered region may be predicted equally well, or even more easily. Conversely, unchanged feature marginals can coexist with a changed outcome relationship and severe calibration failure. Begin an investigation with measurement integrity: units, feature order, missingness, and acquisition changes. Only then compare model scores, calibrated probabilities, and delayed-label residuals against the frozen reference.

Under pure covariate shift, a target-population risk can sometimes be expressed as a weighted reference risk. Require the same outcome law given inputs and absolute continuity of target inputs relative to reference inputs.
It does not create labeled evidence for genuinely unseen inputs.
Estimated weights can be unstable, so a large effective concentration on a few observations warns that the numerical correction is weakly supported.

Prediction sets make a coverage claim, not a calibrated-probability claim. Split conformal prediction fixes a score before calibration and uses a corrected empirical rank. With nine calibration residuals, 90 percent target coverage selects the ninth residual; 95 percent requires the infinite-threshold branch. The theorem explains these choices under exchangeability with the future pair. Coverage is marginal, not a guarantee for every input or subgroup, and arbitrary drift can break its premise.

Shimodaira [9.7] grounds importance weighting under covariate shift. Angelopoulos and Bates [9.6], Section 2.1 and the coverage proof, establish finite-sample marginal conformal coverage under exchangeability and the specified construction—not coverage for every input or subgroup. Tibshirani

and colleagues [9.10], Corollary 1, handle covariate shift with the correct input-density ratio and unchanged conditional outcomes. Estimated ratios introduce uncertainty and do not automatically preserve that guarantee. Report overall coverage, interval size and relevant subgroup diagnostics separately: easy cases can mask harder failures, and none of these summaries certifies validity under arbitrary shift.

Notation for equation (9.10)

t	Population/time-regime index; not a derivative or outcome.
P_t	Probability law of population/time regime t.
X, Y	Random input and its observed outcome.
$P_t(Y \mid X)$	Conditional outcome law given an input in regime t.
$P_t(X)$	Marginal input law in regime t.

$$P_t(X, Y) = P_t(Y \mid X)\, P_t(X) \tag{9.10}$$

Factor the joint distribution for one population into its conditional outcome law and input marginal; t labels a population or time regime.

This factorization is distributional notation; densities or probability masses may be used when appropriate.

Notation for equation (9.11)

x	Input value in the target population's support.
$p_{\text{new}}, p_{\text{ref}}$	New/reference input densities or masses relative to a common measure.
$w(x)$	Importance weight from the reference to the new input distribution.

$$w(x) = \frac{p_{\text{new}}(x)}{p_{\text{ref}}(x)} \tag{9.11}$$

Under covariate shift and support overlap, the density ratio reweights reference inputs toward the new population.

Require absolute continuity of the new input law with respect to the reference law. Use the ratio where the reference density is positive.

Notation for equation (9.12)

$\mathbb{E}_{\text{new}}, \mathbb{E}_{\text{ref}}$	Expectations under new and reference joint distributions.
X, Y	Input and target pair in the relevant population.
f	Fixed fitted predictor, unchanged during the comparison.
ℓ	Integrable per-observation prediction loss.
$w(X)$	Input-density ratio of new to reference population.

$$\mathbb{E}_{\text{new}}[\ell(Y, f(X))] = \mathbb{E}_{\text{ref}}[w(X)\ell(Y, f(X))] \tag{9.12}$$

For a fixed predictor and integrable loss, the weighted-risk identity follows only when the conditional outcome law is unchanged.

The identity assumes identical conditional outcome laws and support overlap; it is not justified for arbitrary concept shift.

◇ Proposition 9.4: Risk transfer under a covariate-shift assumption

Notation for this result

x	Input in the new population's support.
$p_{\text{new}}, p_{\text{ref}}$	New and reference input densities or probability masses.
w	Input importance-weight function, defined reference-almost everywhere.
$r(x)$	Common conditional expected loss at input x.
X, Y, f	Input, target, and fixed fitted prediction function.
$\ell, \mathbb{E}$	Per-observation loss and expectation over the conditional target law.
$\mathbb{E}_{\text{new}}$	Expectation under the new joint population.
$\int \cdot \, dx$	Integration over inputs; use a sum for discrete input masses.
$\mathbb{E}_{\text{ref}}$	Expectation under the reference joint population.

Assumptions. Reference and new populations share the same conditional outcome law given the input. The new input distribution is absolutely continuous with respect to the reference input distribution; their densities or masses use a common underlying measure. The predictor is fixed and the loss is integrable under the new population.

New-population risk equals reference-population risk weighted by the input-density ratio. The identity does not extend automatically to changed conditional outcome laws or unsupported inputs.

□ **Proof** *Advanced*

$$w(x) = \frac{p_{\text{new}}(x)}{p_{\text{ref}}(x)} \tag{9.T13}$$

First integrate the prediction loss over the conditional target distribution at each input. This produces one conditional risk function, because the conditional law is identical in the two populations.

$$r(x) = \mathbb{E}[\ell(Y, f(x)) \mid X = x] \tag{9.T14}$$

Average the common conditional risk over the new input distribution, then express that distribution through its density ratio relative to the reference. Absolute continuity ensures that no new-population mass lies outside reference support. Without it, a weighted reference average has no observations representing the missing region, however large the weights on observed regions become.

$$\mathbb{E}_{\text{new}}[\ell(Y, f(X))] = \int r(x) p_{\text{new}}(x)\, dx \tag{9.T15}$$

$$\int r(x) p_{\text{new}}(x)\, dx = \int r(x) w(x) p_{\text{ref}}(x)\, dx \tag{9.T16}$$

Conditioning back on the reference input turns the final integral into the reference expectation of weighted loss. This proves the identity, but not the reliability of estimated weights. A ratio estimated from finite input samples can be unstable, particularly when new-population mass falls in a scarcely observed reference region.

$$\mathbb{E}_{\text{new}}[\ell(Y, f(X))] = \mathbb{E}_{\text{ref}}[w(X)\ell(Y, f(X))] \tag{9.T17}$$

□

A weighted estimate is justified by a conditional-stability assumption, not by an input-shift test alone. Without that assumption or overlap, new labeled evidence is required.

◇ Theorem 9.5: A finite-sample prediction-set guarantee from exchangeable ranks

Notation for this result

m, i	Positive calibration sample count and calibration observation index.
X_i, Y_i, R_i	Calibration input, observed target, and resulting nonconformity score.

s	Fixed scoring function; larger values indicate poorer conformity.
α, k	Target miscoverage strictly between zero and one, and selected rank index.
$\lceil \cdot \rceil$	Ceiling to the smallest integer not below its argument.
$R_{(k)}, \widehat{q}$	The kth smallest calibration score and chosen score threshold.
$\mathcal{C}(x), x, y$	Prediction set at a query input, that input, and a candidate target value.
X_{m+1}, Y_{m+1}	Future input and outcome exchangeable with the calibration pairs.
$\Pr$	Probability over calibration data and the future pair, conditional on the frozen training information.
$\hat{f}$	Point-prediction function fitted independently of the calibration outcomes.

Assumptions. A model and nonconformity score are fixed using a proper training set before calibration outcomes are examined. Conditional on that training information, the calibration pairs and one future pair are exchangeable. Independent identically distributed pairs satisfy this condition; arbitrary time drift does not. The target miscoverage level is strictly between zero and one. The same fixed scoring rule is applied to every calibration and candidate pair.

The split-conformal set defined by the corrected order statistic contains the next outcome with probability at least the desired coverage level, averaged over calibration and the next pair. No correctness assumption on the fitted point predictor is required.

□ Proof *Advanced*

$$R_i = s(X_i, Y_i), \quad i = 1, \dots, m \tag{9.T18}$$

$$k = \lceil (m+1)(1-\alpha) \rceil \tag{9.T19}$$

$$\widehat{q} = \begin{cases} R_{(k)}, & k \le m, \\ +\infty, & k = m+1. \end{cases} \tag{9.T20}$$

$$\mathcal{C}(x) = \{ y : s(x, y) \le \widehat{q} \} \tag{9.T21}$$

For distinct scores, exchangeability makes the future score equally likely to occupy any of the calibration-count-plus-one ranks. The set accepts exactly the ranks up to the chosen index, so rounding that index upward gives the coverage lower bound. For ties, imagine independent continuous tie breakers. The resulting ranks are again uniform; removing tie breakers and accepting

equality can only enlarge the accepted event. Thus ties make this unrandomized construction conservative, not less covered.

$$\Pr\{\, Y_{m+1} \in \mathcal{C}(X_{m+1})\} \geq \frac{k}{m+1} \geq 1-\alpha \tag{9.T22}$$

An absolute-residual score gives a symmetric interval around a frozen point predictor. More adaptive scores can change interval shapes without changing the rank argument, provided they are fixed before calibration and applied identically.

$$s(x, y) = |y - \hat{f}(x)| \tag{9.T23}$$

$$\mathcal{C}(x) = [\hat{f}(x) - \widehat{q}, \hat{f}(x) + \widehat{q}] \tag{9.T24}$$

□

Split-conformal coverage averages over the calibration sample and next pair, not a fixed input, subgroup or realized calibration sample. Tuning the predictor or score with calibration outcomes breaks the exchangeable-rank argument. Use the required order statistic, not an interpolated percentile: too few calibration examples may require an infinite interval rather than justify a finite one.

▶ Example: Two shifts requiring different evidence

Keep the true group rates 0.2 and 0.8, but increase B's share from one-half to three-quarters. Prevalence becomes 0.25 times 0.2 plus 0.75 times 0.8 = 0.65; the correct group forecasts remain calibrated. Alternatively, keep equal shares but raise B's rate to 0.95. Input counts need not change, yet the old 0.8 forecast underpredicts B by 0.15. Input monitoring can reveal the first change; labeled residuals are needed for the second. Neither altered population inherits the old sample's ordinary conformal guarantee automatically.

Interpretation and limits

A drift alert, an empirical coverage percentage, and a theorem have different meanings. State the reference population, available labels, and assumptions before deciding whether recalibration or retraining is justified.

Algorithm 9.2. Split-conformal residual intervals

Source and adaptation. The corrected calibration rank and exchangeable coverage argument follow Angelopoulos and Bates [9.6], including the infinite-threshold branch when the selected rank exceeds the calibration count.

Notation and assumptions

$\hat{f}, x_i, y_i, m$	Frozen real-valued predictor, calibration pairs, and positive calibration count.
α, k	Miscoverage strictly between0 and1, and corrected integer rank.
$R_i, R_{(k)}, \widehat{q}$	Absolute residual, kth order statistic counting ties separately, and saved half-width.
$x, \mathcal{C}(x)$	Future input and its prediction interval.
$\lceil\cdot\rceil, +\infty$	Integer ceiling and infinite threshold, producing the whole real line.

The predictor and absolute-residual score are fixed before calibration outcomes are examined. Use reliable ceiling arithmetic at integer boundaries. The kth order statistic is not an interpolated percentile. Any nonfinite calibration prediction or residual stops with numerical-failure status.

Algorithm 9.2

Input: Frozen predictor fitted on proper training data.

Calibration pairs exchangeable with a future pair, conditional on training.

Fixed miscoverage and score convention. $m \geq 1, \quad 0 < \alpha < 1$

Output: Frozen predictor, saved threshold, miscoverage, count and score convention.

01 **Require** Valid count/miscoverage and proper training-calibration separation.

02 **For** $i = 1, \dots, m$

03 $R_i \leftarrow |y_i - \hat{f}(x_i)|$

04 **Require** Finite residual.

05 **End for**

06 Corrected rank. $k \leftarrow \lceil (m+1)(1-\alpha) \rceil$

Algorithm 9.2 (continued)

If $k \leq m$

  Count tied residuals separately. $\widehat{q} \leftarrow R_{(k)}$

Else

  $\widehat{q} \leftarrow +\infty$

End if

Define Frozen future-input interval. $\mathcal{C}(x) = [\hat{f}(x) - \widehat{q}, \hat{f}(x) + \widehat{q}]$

Return Predictor, threshold and saved calibration contract.

Why this works. Membership uses a weak inequality, so tied scores are included. If k=m+1, the interval is the whole real line. Never retune the model, score or miscoverage using these calibration outcomes. The guarantee is marginal over calibration and the next exchangeable pair, not conditional on every input or subgroup; arbitrary drift invalidates the premise.

Table 9.1. Calibration and diagnostic tools

Tool	What it measures or changes	Interpretation limit
Brier / log loss	Evaluate probability forecasts using proper scores.	Reflect discrimination and uncertainty as well as calibration.
Reliability diagram	Compare forecast means with observed frequencies in bins.	Sparse bins and arbitrary bin boundaries can obscure evidence.
Sigmoid / isotonic	Fit a mapping from held-out scores to probabilities.	Calibration labels must be excluded from base fitting.
Permutation / dependence	Probe a fitted model with specified feature perturbations.	Importance and dependence are not causal effects.

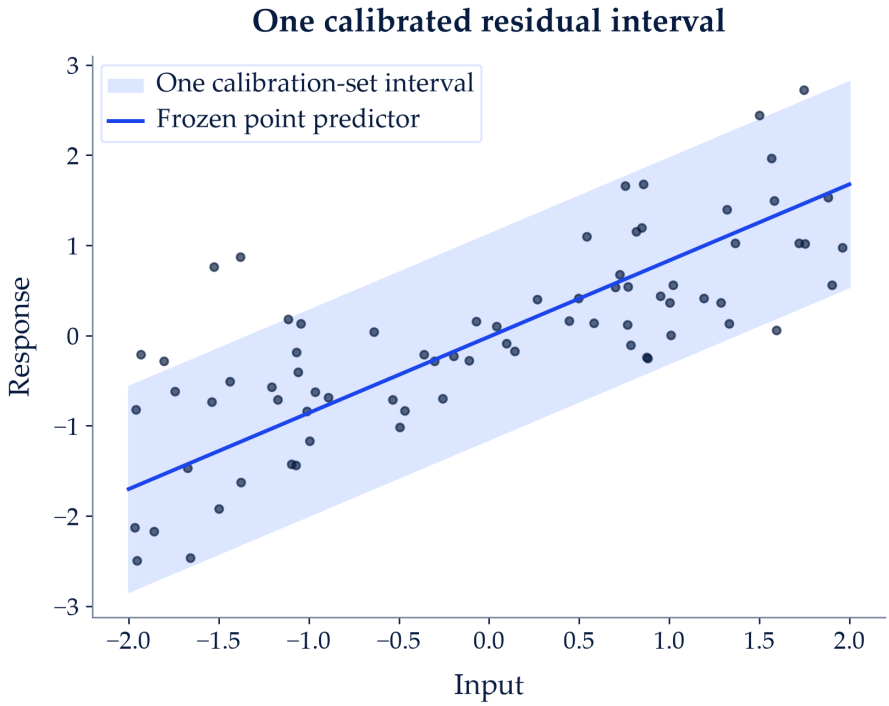


Figure 9.3. A point predictor is frozen before a separate set of 99 residuals calibrates its interval. For 90% target marginal coverage, the residual threshold is the 90th order statistic, not an interpolated percentile. This panel displays one calibrated interval and new noisy observations from a heteroscedastic population.

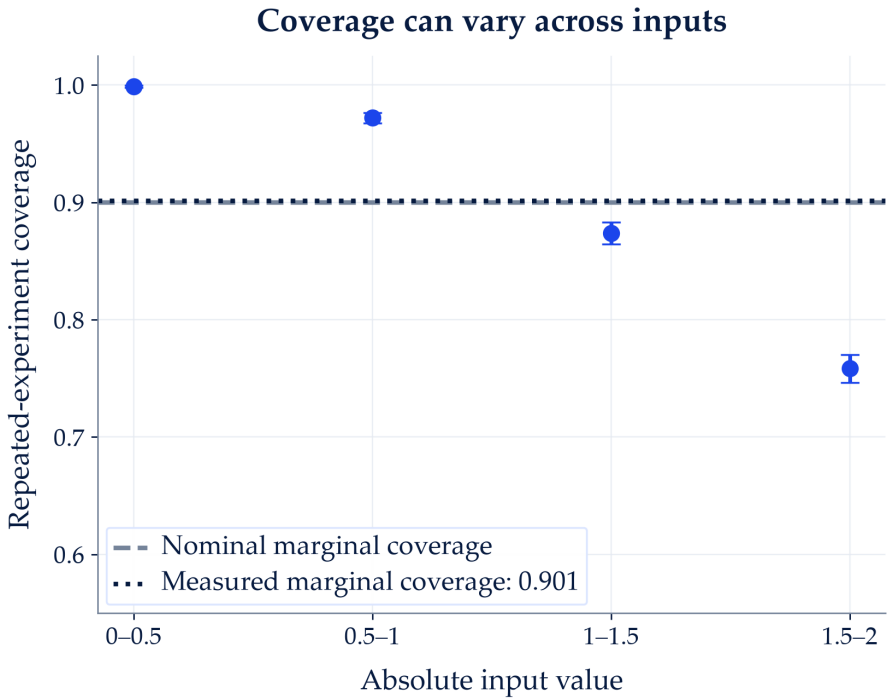


Figure 9.4. Group inputs by magnitude horizontally and read coverage vertically. Twenty thousand independent calibration-and-test experiments under one heteroscedastic population give about 90% marginal coverage, yet different group rates. Bars are approximate 95% Wilson intervals. Split conformal guarantees a marginal rank property, not equal coverage in every group.

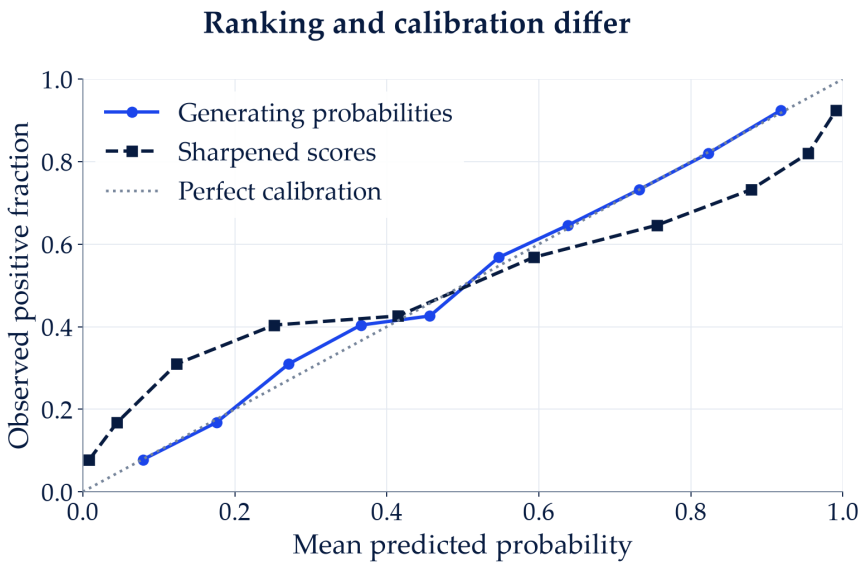


Figure 9.5. Synthetic outcomes are sampled from the generating probabilities. A monotonic sharpening preserves ranking but makes scores more extreme. The reliability plot reveals the resulting calibration distortion. Bin estimates remain noisy even when the generating probabilities are known.

Table 9.2. Chapter synthesis

Concept	Main conclusion	Engineering check
Calibration	Probability values should match event frequencies in the relevant population.	Show counts and evaluate new data after fitting a calibration map.
Error slices	Global scores can conceal subgroup failure.	Predeclare important slices and report their sample sizes.

Concept	Main conclusion	Engineering check
Interpretation	A diagnostic answers a specific perturbation question about a model or input distribution.	Check correlated features and unrealistic feature combinations.
Shift	A changed input distribution is a warning, not proof of degraded predictive risk.	Seek outcome evidence and distinguish which assumption changed.

Case Study: Audit probability claims on Wisconsin diagnostic measurements

Problem and data

A classifier can rank historical observations usefully while making poor numerical probability claims. This case asks whether a separately fitted sigmoid map improves a frozen forest's probability scores on independent development observations. The positive event is explicitly malignant, reversing the library's original coding. Four data roles prevent calibration and threshold choices from consuming final evidence. The experiment is methodological education only, with hypothetical costs and no authorization for clinical use or recommendations.

Experimental design

Reserve a final stratified subset first, then allocate the remaining records to base fitting, calibration, and policy validation. Fit the forest once and train a sigmoid map on clipped raw log odds from calibration rows. Compare raw and mapped probabilities on policy-validation Brier score, retaining the raw pipeline on ties. Select a threshold using a predeclared hypothetical cost grid. Evaluate the frozen choice once and display proper scores, confusion counts, and a reliability table with nonempty-bin support.

Measured Reference Run

Four stratified roles separated 227 forest-fitting, 114 sigmoid-calibration, 114 decision-validation and 114 final records. The forest used 100 trees. Decision validation selected raw or recalibrated probabilities by Brier score, then searched nine thresholds under illustrative false-negative/false-positive costs of five/one. It selected recalibration and threshold 0.1; the original forest and calibration fits remained frozen.

Table 9.3. Measured case-study evidence

Final probability source	Brier	Log loss	ROC AUC
Sigmoid recalibrated	0.026	0.12	0.987
Fitting prevalence	0.233	0.658	0.5

Decision-validation Brier score fell from 0.025 to 0.022, selecting recalibration. Its higher final Brier score, still far below the prevalence reference, shows why selection evidence is not a promised final score. Calibration fitting, pipeline selection and final evaluation each need their own data role.

Failure Analysis And Limits

The final reliability bin with mean predicted probability 0.708 contained only four records, all malignant; the middle bin contained only two. Such discrepancies and tiny denominators preclude a precise calibration guarantee. Neither apparently good aggregate scores nor the chosen threshold establishes clinical safety, subgroup calibration or transportability.

Historical saved run; retain all four disjoint roles, the positive-label recoding, fixed bins and fitting-only slice boundary. No clinical recommendation follows.

▶ Example: inspecting the real data

The first final record is benign, so its malignant indicator is 0. Its saved predicted probability is about 3.305e-4; the squared difference from 0 contributes about 1.093e-7 to the Brier total. Averaging all 114 individual contributions gives the reported Brier score. One tiny contribution establishes neither aggregate calibration nor clinical safety.

Dataset source and documentation

Companion: Week 09, Real-Data Case Study notebook. Reproduce the declared run, then investigate a changed condition. No fixed benchmark score is promised.

A Controlled Learning Cycle

The WDBC probability audit keeps fitting, calibration, policy selection and final evaluation distinct. Week 9 notebooks connect calibration boundaries, information loss and exchangeability to uncertainty claims. The two-feature laboratory isolates temperature/isotonic mappings and counted reliability bins. Record one controlled comparison on unseen rows with metric, data roles and explanation; the scope below prevents reading it as conformal coverage, causal explanation or clinical validation.

Matched practice and simulator protocol · Packaged edition; launch instructions in the guide.

Laboratory 09 | Probability Calibration And Model Diagnosis

Temperature/isotonic mappings, separate calibration rows and counted bins.

Temperature and isotonic calibration use a separate calibration partition, with counted probability bins and before-and-after predictions.

Open Laboratory 09 | Use the live view for readable controls.

Dataset: Wisconsin Diagnostic Breast Cancer (WDBC). See its first-use portrait [9.11].

Transfer: WDBC probability audit with distinct fitting roles → Two-feature WDBC classification. No conformal guarantee, causal explanation or clinical validation.

Chapter Capability

Keep fitting, calibration, selection, and final evaluation distinct, and explain both the evidence and the limits of a reliability claim.

Week 09: separate practice, solutions and assessment

References

[9.1] Gneiting, T.; Raftery, A. E. (2007). Strictly Proper Scoring Rules, Prediction, and Estimation. Journal of the American Statistical Association 102(477), 359-378. Source / DOI

[9.2] Niculescu-Mizil, A.; Caruana, R. (2005). Predicting Good Probabilities With Supervised Learning. Proceedings of the 22nd International Conference on Machine Learning. Source / DOI

[9.3] Guo, C.; Pleiss, G.; Sun, Y.; Weinberger, K. Q. (2017). On Calibration of Modern Neural Networks. Proceedings of Machine Learning Research 70, 1321-1330. Source / DOI

[9.4] Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. The Annals of Statistics 29(5), 1189-1232. Source / DOI

[9.5] Strobl, C.; Boulesteix, A.-L.; Kneib, T.; Augustin, T.; Zeileis, A. (2008). Conditional variable importance for random forests. BMC Bioinformatics 9, Article 307. Source / DOI

[9.6] Angelopoulos, A. N.; Bates, S. (2023). Conformal Prediction: A Gentle Introduction. Foundations and Trends in Machine Learning 16(4), 494-591. Source / DOI

[9.7] Shimodaira, H. (2000). Improving predictive inference under covariate shift by weighting the log-likelihood function. Journal of Statistical Planning and Inference 90(2), 227-244. Source / DOI

[9.8] Kumar, A.; Liang, P.; Ma, T. (2019). Verified Uncertainty Calibration. Advances in Neural Information Processing Systems 32. Source / DOI

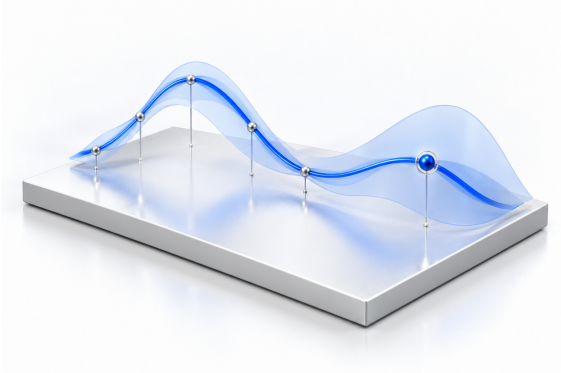
[9.9] Hebert-Johnson, U.; Kim, M.; Reingold, O.; Rothblum, G. (2018). Multicalibration: Calibration for the (Computationally-Identifiable) Masses. Proceedings of Machine Learning Research 80, 1939-1948. Source / DOI

[9.10] Tibshirani, R. J.; Barber, R. F.; Candès, E. J.; Ramdas, A. (2019). Conformal Prediction Under Covariate Shift. Advances in Neural Information Processing Systems 32. Source / DOI

[9.11] Wolberg, W.; Mangasarian, O.; Street, N.; Street, W. (1993). Breast Cancer Wisconsin (Diagnostic). UCI Machine Learning Repository. Source / DOI

CHAPTER 10

Gaussian Processes And Experiment Selection



Reading path

Start with Essential. The other levels offer more depth.

Essential
10.1 10.2 10.3 10.4 10.5

Recommended
Proofs 10.3

Advanced
Proofs 10.1 10.2 10.4

Worked example Lab task

When another measurement has a cost, selecting where to measure becomes part of learning. Compare uncertainty-driven information gathering with searching for a high objective value.

What You Will Be Able To Do

- Construct a Gaussian-process prior and compute a noisy regression posterior.
- Distinguish latent-function uncertainty from uncertainty in a future observation.
- Implement pool-based active learning and compare maximization-oriented UCB and expected improvement.

Before you begin: Multivariate normal distributions and covariance; Kernel functions and matrix operations; Training, validation, and untouched test partitions.

Compute posterior moments and separate latent from observation uncertainty before choosing an acquisition objective.

Algorithm scope: Inference and acquisition orchestration: exact GP conditioning uses Cholesky solves; Algorithms 10.1-10.2 specify finite-pool selection and request accounting while delegating GP fits and oracle responses.

Shared chapter notation

σ_n^2 Positive common observation-noise variance, in squared response units; subscript n denotes noise, not sample count.

ν	Positive dimensionless Matérn smoothness parameter, fixed here to three halves; nu in prose.
κ	Nonnegative dimensionless upper-confidence-bound exploration coefficient multiplying latent posterior standard deviation.
ξ	Nonnegative improvement offset in objective units, raising the fixed threshold for positive improvement.

Chapter notation

GP	Gaussian process: distribution over functions specified by a mean and valid covariance kernel.
RBF	Radial basis function; here the squared-exponential covariance kernel, not a separate predictor.

Point prediction estimates a response; measurement selection also considers how new evidence could change that estimate. A Gaussian process (GP) combines relationships between inputs with a probability distribution over function values. Its uncertainty is conditional on the model assumptions, not evidence that those assumptions describe the world.

The GP supports two distinct decisions: active learning requests labels to improve prediction, while Bayesian optimization requests evaluations to find a good objective value. Their acquisition rules and evaluation protocols differ. Synthetic notebook experiments expose that distinction without assuming either method always outperforms random sampling.

- *Essential*

10.1 A distribution over related function values

Notation for this section

n, d	Positive training-example count and input-feature count.
X	Real n-by-d training-input matrix, one example per row.
x_i, i	Length-d input column for training example i, indexed from one through n.
K	Symmetric n-by-n latent training-response covariance, before observation noise.
f	Unknown scalar latent response function on input space, in target-response units.
y_i, y	Measured scalar response for example i and length-n training-response column.

A Gaussian process is a prior over functions, not a claim that inputs are Gaussian. For any finite input list, it assigns a joint normal distribution to their function values. Evaluate the mean function to get expected responses and the covariance kernel to link deviations around them. The resulting consistent finite-dimensional distributions define the function prior.

For n training examples, arrange the input rows in X with shape n by d, while each mathematical input x_i is a d-dimensional column vector. Evaluating the kernel for every input pair produces an n by n matrix K. Valid kernels make this matrix positive semidefinite: every weighted combination of its coordinates has nonnegative variance. This requirement matters because an arbitrary similarity score need not define a legitimate Gaussian distribution.

Separate the latent response f from the measured target y. The standard regression model adds independent, zero-mean Gaussian observation noise with variance σ_n squared. Repeated measurements at the same input can then differ even though the latent function is unchanged. An explicit mean function is valuable when trends are known. A zero prior mean is a modeling choice, often combined with target centering learned from training data, not a universal property of Gaussian processes.

Rasmussen and Williams [10.1] develop Gaussian processes as distributions over functions specified by a mean and covariance. This construction distinguishes uncertainty about a latent response function from the additional variability of a future noisy observation.

Notation for equations (10.1), (10.2), (10.3)

f	Unknown scalar latent response function before independent measurement noise.
$\mathcal{GP}(m, k)$	Gaussian process: every finite collection of function values is jointly Gaussian.
m	Prior mean function in response units: expected latent response before observing training targets.
k	Positive-semidefinite latent covariance kernel, in squared response units.
x_i	Training-example-i input column with d coordinates.
y_i	Training-example-i scalar target, including measurement noise.
i, n, d	Training index from one through n, training count and input dimension.
ϵ_i	Independent zero-mean Gaussian observation noise for example i, independent of the latent process.

$\mathcal{N}(0, \sigma_n^2)$ Univariate normal with zero mean and observation-noise variance sigma-n squared.

$$f \sim \mathcal{GP}(m, k) \tag{10.1}$$

$$y_i = f(x_i) + \epsilon_i \tag{10.2}$$

$$\epsilon_i \sim \mathcal{N}(0, \sigma_n^2) \tag{10.3}$$

The latent function is f; m is its prior mean; k is its covariance function. Target y_i is the function at input x_i plus independent Gaussian noise ϵ_i. The positive quantity σ_n squared is observation-noise variance.

The Gaussian assumption concerns latent function values and the stated noise model, not the distribution of input features.

Notation for equations (10.4), (10.5)

X	n-by-d training inputs; each row is a transposed input vector.
$\mathbf{f}_X$	Length-n column of latent values at the training inputs.
$\mathbf{m}_X$	Length-n column of prior means at the same inputs.
K	Symmetric positive-semidefinite n-by-n prior latent training covariance.
K_{ij}	Training-input-i/j latent covariance, before observation noise.
$k(x_i, x_j)$	Kernel covariance between the two d-dimensional training inputs.
i, j, n, d	Training indices i,j run from one through n; d is the feature count.
$\mathcal{N}(\mathbf{m}_X, K)$	Multivariate normal with mean m-X and covariance K.

$$\mathbf{f}_X \sim \mathcal{N}(\mathbf{m}_X, K) \tag{10.4}$$

$$K_{ij} = k(x_i, x_j) \tag{10.5}$$

The vectors $\mathbf{f}_X$ and $\mathbf{m}_X$ contain function and mean values at the n training inputs. K is the n by n prior covariance matrix. This finite collection is sufficient for implementing GP regression.

▶ Example: Two inputs, one covariance assumption

For two standardized inputs, specify zero prior mean, unit latent variances and covariance 0.8. The latent difference has variance 1 plus 1 minus twice 0.8 = 0.4: shared variation cancels in a difference. Add independent observation noise of variance 0.09 to each. Observed variances become 1.09 but covariance stays 0.8, so difference variance rises to 0.58. Independent errors add rather than cancel.

These are chosen prior assumptions, not covariance estimates obtained from two observations.

Interpretation and limits

A normally distributed posterior does not mean the inputs or observed targets must have a normal marginal histogram. The assumptions concern conditional function values and the observation model.

Essential

10.2 RBF and Matérn kernels encode smoothness

The squared-exponential, or radial basis function, kernel specifies how similarity fades with squared input distance. Amplitude controls latent variation; length scale controls how far observations inform each other. Long scales favor slowly varying functions, while short scales permit rapid change. Both interpretations depend on feature units. Standardization changes the numerical length-scale meaning and must remain inside the training workflow.

RBF sample functions are extremely smooth. The Matérn family allows rougher functions through a smoothness parameter called nu. A common fixed choice is nu equal to three halves, whose kernel has a simple expression and permits less smooth behavior than the RBF. Selecting a Matérn kernel is not a claim that reality is rough; it is a deliberate change in the prior function class. Compare plausible choices using validation designed for the deployment setting.

Log marginal likelihood scores the observed targets under a proposed covariance model, balancing fit with the covariance's complexity. Maximizing it can leave local solutions or poorly identified combinations of amplitude, noise and length scale. Bounded searches, multiple starts and prior knowledge can help distinguish them. A short fitted length scale may describe genuine variation, unmodeled noise or an unsuitable representation; the fit alone cannot identify which explanation, much less establish a causal mechanism.

Stein [10.2] analyzes covariance assumptions in spatial interpolation, while Rasmussen and Williams [10.1] connect such kernels to Gaussian-process learning. Kernel smoothness is consequently a modeling assumption about related function values, not just a numerical tuning device.

Notation for equation (10.6)

k_{RBF}	RBF covariance kernel, also called squared exponential.
$x, x^{'}$	Two d-dimensional input columns; prime distinguishes inputs, not differentiation.
d	Feature count in the kernel's consistently scaled coordinates.

σ_f^2	Nonnegative signal variance in squared response units; latent prior variance at one input.
ℓ	Positive isotropic correlation length scale, in input-distance units.
$\|x - x'\|$	Euclidean input distance in the selected representation; its square enters the exponent.
exp	Natural exponential: converts the dimensionless negative distance term to positive correlation.

$$k_{\mathrm{RBF}}(x, x') = \sigma_f^2 \exp\left(-\frac{\|x - x'\|^2}{2\ell^2}\right) \tag{10.6}$$

Signal variance σ_f squared sets the amplitude, ell is the positive length scale, and the norm is Euclidean distance between d-dimensional inputs x and x prime. This isotropic form uses one length scale for all features.

This isotropic kernel uses one shared length scale. Changing feature units without changing that scale changes the prior.

Notation for equation (10.7)

$k_{3/2}$	Matérn covariance with smoothness fixed to three halves.
r	Nonnegative Euclidean input distance in the chosen representation.
σ_f^2	Nonnegative signal variance: latent prior variance at zero distance.
ℓ	Positive length scale, in the units of r.
exp	Natural exponential governing distance-dependent covariance decay.

$$k_{3/2}(r) = \sigma_f^2 \left(1 + \frac{\sqrt{3}r}{\ell}\right) \exp\left(-\frac{\sqrt{3}r}{\ell}\right) \tag{10.7}$$

This is a Matérn kernel with smoothness nu equal to three halves. Here r is the distance between inputs; ell and σ_f squared retain their length-scale and signal-variance meanings.

The subscript 3/2 specifies smoothness; it is not a training-example index.

◇ Derivation 10.1: The marginal likelihood balances residual fit and covariance volume

Notation for this result

n	Positive training-observation count in the marginal likelihood.
e, y, m_X	Length-n residual, observation and fixed prior-mean columns; e is not the exponential base.

A, R	Positive-definite n-by-n observation covariance and lower-triangular Cholesky factor with positive diagonal.
$\mathcal{L}$	Natural log marginal observation density conditional on inputs and model parameters.
$\det, \log, \pi$	Matrix determinant, natural logarithm, and the circle constant, respectively.
θ_j, ∂_j	Scalar covariance hyperparameter and partial differentiation with respect to it; j indexes fitted kernel/noise parameters.
A_j	n-by-n covariance derivative with respect to that parameter.
tr	Trace: the sum of diagonal entries of a square matrix.
α	Length-n coefficient column solving the observation-covariance system for the residual.

Assumptions. The observation covariance is positive definite and differentiable in each scalar kernel parameter. The mean is held fixed for the derivative below. Noise parameters are included in the covariance if they are being estimated.

Taking the logarithm of the multivariate Gaussian density yields a quadratic residual term, a log-determinant term, and a normalization constant. Its covariance-parameter gradient follows from differentiating an inverse and a determinant.

Read the complete supporting proof . *Advanced*

Marginal-likelihood optimization learns a covariance model from the observed data, but it does not validate the assumed model family or replace held-out predictive checks.

▶ Example: Reading a length scale numerically

Take an RBF kernel with signal variance 1 and length scale 2. At distance 2, the distance-to-length-scale ratio is 1, giving exponent negative one half and covariance approximately 0.607. At distance 4, the ratio is 2, giving exponent negative two and covariance approximately 0.135. Converting meters to centimeters must also change the length scale from 2 to 200; both ratios then remain unchanged. Converting only the inputs instead makes these locations nearly uncorrelated. Check the dimensionless ratio before interpreting a fitted length scale: units and preprocessing belong to the covariance specification.

Interpretation and limits

A fitted short length scale does not establish a sensitive physical or causal mechanism. Check scaling, noise assumptions, validation errors, and alternative kernels first.

Essential

10.3 Conditioning and interpreting posterior uncertainty

GP prediction updates a joint Gaussian by conditioning on observations. With zero prior mean, the posterior mean is a training-target weighted combination, where covariance and noise determine the weights. Latent variance is prior variance minus the part explained by observations. For a nonzero mean, subtract training means before conditioning and restore the query mean afterward; omitting these steps changes the model.

Factor the training covariance plus modeled noise using Cholesky decomposition. Solve the two triangular systems for the mean coefficients, then reuse the factor to compute covariance reductions; do not form an inverse. Numerical jitter can stabilize near-singularity but is not an estimate of observation noise. Dense factorization costs cubic time in n and quadratic memory, so dataset size and approximation choices matter even when the query calculation is simple.

First ask what the interval predicts. Latent variance concerns f; an independent future measurement adds observation-noise variance under the same model. These conditional Bayesian intervals need not achieve empirical coverage when the mean, kernel, noise or deployment distribution is wrong. Plug-in hyperparameters also omit estimation uncertainty. Check interval width and coverage on appropriate held-out groups.

Rasmussen and Williams [10.1] derive prediction by conditioning a joint Gaussian distribution. Uncertainty remains conditional on the kernel, noise model and fitted hyperparameters; misspecification or a new population can invalidate calibration. Titsias [10.7], equation (9), extends inference through a variational lower bound with a residual-covariance trace penalty. Inducing inputs are variational parameters, not extra observations. More inducing points can tighten the bound, but do not guarantee calibration under a wrong kernel. Distinguish approximation error, observation noise and model misspecification.

Notation for equations (10.8), (10.9)

x_*	Query input column for the latent posterior mean and variance.

μ_*	Scalar latent query posterior mean given training observations.
$s_f^2(x_*)$	Scalar latent query posterior variance, excluding future observation noise.
k_*	Length-n column of covariances between each training input and the query.
K	n-by-n latent training covariance before observation noise.
y	Length-n observed training-target column, including noise.
σ_n^2	Common independent training-noise variance; subscript denotes noise.
I	n-by-n identity, placing independent noise variance on the covariance diagonal.
$k(x_*, x_*)$	Prior latent query variance: kernel evaluated twice at the query.
n	Training-observation count used to condition the GP prior.

$$\mu_* = k_*^T (K + \sigma_n^2 I)^{-1} y \tag{10.8}$$

$$s_f^2(x_*) = k(x_*, x_*) - k_*^T (K + \sigma_n^2 I)^{-1} k_* \tag{10.9}$$

These formulas assume a zero prior mean. The n-vector k_* contains covariances between training inputs and query x_* ; y is the n-vector of targets; I is the identity matrix. μ_* and s_f squared are the latent posterior mean and variance.

These expressions assume a zero prior mean and an invertible noisy covariance matrix. Implement the inverse action with linear solves.

Notation for equation (10.10)

x_*	Query for a future independently noisy response.
$s_y^2(x_*)$	Future measured-response posterior variance, including noise.
$s_f^2(x_*)$	Latent-response posterior variance at the same query.
σ_n^2	Independent future-noise variance, equal to modeled training-noise variance.

$$s_y^2(x_*) = s_f^2(x_*) + \sigma_n^2 \tag{10.10}$$

Variance s_y squared concerns a new noisy observation; s_f squared concerns the latent function. The addition assumes future noise is independent with the same variance σ_n squared.

Adding variances requires future observation noise to be independent of the latent response conditional on the model.

◇ Theorem 10.2: Gaussian conditioning and the Schur complement

Notation for this result

y, m_X	Length-n training-observation and prior-mean columns; n counts observations.
f_*, m_*	Length-q latent-query and prior-mean columns; q counts queries.
A	Positive-definite n-by-n observation covariance: training kernel plus independent noise covariance.
C, B	n-by-q training-query and q-by-q latent-query covariances.
r	Length-q latent-query residual after affine prediction.
Cov	Covariance (one vector) or cross-covariance (two), under the joint Gaussian law.
S	q-by-q residual covariance: observation-block Schur complement.

Assumptions. The training observation vector and a finite query vector are jointly Gaussian. The observation covariance is positive definite. Kernel, mean, and noise parameters are fixed during conditioning; query values are latent function values rather than new noisy measurements.

The conditional query distribution is Gaussian. Its mean is an affine correction to the prior mean, and its covariance is the Schur complement of the observation covariance. This covariance is positive semidefinite and cannot exceed the prior query covariance in any direction.

□ Proof *Advanced*

The block vector contains latent query values and noisy training observations. Independent training noise contributes to the observation covariance block but not to its cross-covariance with latent query values. That cross-covariance comes only from the shared latent kernel. Keeping these roles separate prevents counting independent noise as information about the function.

$$\begin{bmatrix} y \\ f_* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m_X \\ m_* \end{bmatrix}, \begin{bmatrix} A & C \\ C^T & B \end{bmatrix} \right) \tag{10.T8}$$

Subtract the affine prediction based on the observations and examine the remaining query residual. This is a linear transformation of the jointly Gaussian vector, so it remains jointly Gaussian with the observations. The next covariance calculation can therefore establish independence from zero cross-covariance; for a general non-Gaussian distribution, zero covariance alone would not suffice.

$$r = f_* - m_* - C^T A^{-1}(y - m_X) \tag{10.T9}$$

$$\mathrm{Cov}(r, y) = C^T - C^T A^{-1} A = 0 \tag{10.T10}$$

The residual is uncorrelated with the observation block. Joint Gaussianity upgrades this statement to independence: with zero cross term, the joint Gaussian characteristic function factors. Conditioning on observations therefore leaves the residual distribution unchanged. Expanding its covariance then gives the next formula: the two cross terms subtract the correction twice, and the correction's own variance restores one copy. The net effect is one subtraction.

$$S = \mathrm{Cov}(r) = B - C^T A^{-1} C \tag{10.T11}$$

$$f_* \mid y \sim \mathcal{N}\big(m_* + C^T A^{-1}(y - m_X), S\big) \tag{10.T12}$$

The conditional covariance is a covariance of residuals, so it is positive semidefinite. The amount subtracted from the prior is also positive semidefinite, since it uses the inverse of a positive-definite observation covariance. Therefore conditioning cannot increase variance in any linear combination of query values, even when the residual covariance is singular. A new independent noisy measurement adds its own noise covariance afterward. Compute the formulas with linear solves, not an explicit inverse.

□

Posterior contraction is a statement inside a fixed Gaussian model, not a promise of calibrated real-world error bars. Re-estimating hyperparameters changes the model and can increase reported uncertainty.

▶ Example: One observation is enough to see the distinction

At one input, let the prior mean be zero and variance 1, and observe target 2 with noise variance 0.25. The observation variance is 1.25, giving posterior mean 2/1.25 = 1.6 and latent variance 1 minus 1/1.25 = 0.2. A future independent measurement adds noise 0.25, producing variance 0.45. Standard deviations are about 0.447 for the latent function and 0.671 for the measurement. The mean moves toward the noisy target without interpolating it; measurement uncertainty remains larger because a fresh error has not been observed.

Interpretation and limits

A GP does not provide calibrated real-world uncertainty automatically. Coverage depends on model assumptions and must be checked under the intended evaluation distribution.

Example: Two Observations, Triangular Solves, And One New Label

Source and scope. The Cholesky posterior calculation follows Rasmussen and Williams [10.1], Chapter 2, Algorithm 2.1. The numerical values are a constructed arithmetic example.

This arithmetic example is constructed to expose the calculation, not to report dataset performance. Use scalar inputs $x_1 = 0$ and $x_2 = 1$, observed targets y=(1,2) transposed into a column, a zero prior mean, and independent Gaussian observation-noise variance 0.25. The kernel below has unit marginal variance; log is the natural logarithm and x-prime is a second input.

$$k(x, x') = \exp\left[-(\log 2)(x - x')^2\right] \tag{10.C1}$$

Let K be the two-by-two kernel matrix at the observed inputs, I_2 the identity matrix, and A the observation covariance $K + 0.25 I_2$. Let L be its lower-triangular Cholesky factor with positive diagonal. Rounded displays below are never fed back into subsequent calculations.

$$A = \begin{pmatrix} 1.25 & 0.5 \\ 0.5 & 1.25 \end{pmatrix} \tag{10.C2}$$

$$L = \begin{pmatrix} \sqrt{1.25} & 0 \\ 0.5/\sqrt{1.25} & \sqrt{1.05} \end{pmatrix} \tag{10.C3}$$

Direct multiplication verifies A=L L-transpose. Define b by solving Lb=y, then alpha by solving L-transpose alpha=b. Forward substitution gives b approximately (0.894,1.561); backward substitution gives alpha=(4/21,32/21). No inverse matrix is formed.

Predict at x-star=0.5. Its covariance with each observed latent value is r=2 to the power minus one quarter, approximately 0.841. Define k-star=(r,r) transposed and solve Lv=k-star. This gives v approximately (0.752,0.492). The posterior latent mean mu-star and variance s-star-squared now follow.

k_*, α, μ_* Two-element query covariance column, solved coefficient column, and posterior latent mean at x-star.

$$\mu_* = k_*^T \alpha \approx 1.442 \tag{10.C4}$$

v, s_*^2 Solution of the lower-triangular query solve and posterior variance of the latent function, respectively.

$$s_*^2 = 1 - v^T v \approx 0.192 \tag{10.C5}$$

A fresh noisy observation at this input has variance approximately 0.442: the latent variance plus 0.25, not another subtraction of the noise. Suppose this eligible query is selected and its newly acquired label is 1.5. Keep the kernel and noise fixed. Let mu-new and s-new-squared be the updated mean and latent variance at that same query. Gaussian conditioning gives the next two formulas.

$$\mu_{\text{new}} = \mu_* + \frac{s_*^2}{s_*^2 + 0.25}(1.5 - \mu_*) \approx 1.467 \tag{10.C6}$$

$$s^2_{\text{new}} = \frac{0.25 s_*^2}{s_*^2 + 0.25} \approx 0.109 \tag{10.C7}$$

The update balances the old latent prediction against a noisy label, so the posterior mean moves partway and latent variance contracts without vanishing. Recomputing the three-observation Cholesky solution should agree with this sequential update under the same fixed model. Agreement checks implementation, not query optimality. Select only eligible pool points without reading unacquired labels, and keep final-test labels out of assimilation.

Rasmussen and Williams (2006), Gaussian Processes for Machine Learning, Chapter 2, Algorithm 2.1: posterior computation by Cholesky factorization and triangular solves. Source

Essential

10.4 Active learning asks which label improves prediction

Active learning chooses which label to buy next. Start with labeled examples and a separate eligible, unlabeled pool. Select an input, obtain its label from an oracle, add the pair to training, and repeat. The oracle may be a person, simulator or measurement process. The objective is predictive quality for a defined population within a labeling budget, not simply finding a large response.

Largest latent variance selects the most uncertain candidate under the model, not necessarily the most useful label. An isolated point can have high variance but little covariance with deployment inputs, so measuring it may help few predictions. Integrated-variance or density-weighted criteria incorporate that relevance; cost and diversity add other constraints. When noise varies by input, separate latent variance from observation variance: the latter can be large mainly because of irreducible measurement noise.

The evaluation set must stay outside the query pool. In a simulation, labels for every pool point may already exist in memory, but the policy should only access the label after selecting its index. Compare methods using identical starting labels, labeling budgets, candidate pools, and multiple random seeds where feasible. Plot performance against labels acquired, not just iteration number when batches differ. Use validation for acquisition design and reserve a final test population for the completed protocol.

Cohn, Ghahramani, and Jordan [10.3] study statistical criteria for selecting informative observations. Settles [10.4] organizes active-learning strategies and their limitations, supporting the distinction between querying uncertain cases and selecting cases expected to improve prediction. Krause, Singh, and Guestrin [10.8] study GP observation selection through mutual information between chosen and unobserved locations. Their criterion differs from choosing the single largest marginal variance, illustrating why informativeness depends on the stated design objective.

Notation for equation (10.11)

t	Acquisition iteration indexing labels available for posterior fitting.
$\mathcal{U}_t$	Eligible unlabeled pool at iteration t, excluding untouched test inputs.
x	Current pool input whose label is unavailable to selection.
$s^2_{f,t}(x)$	Latent posterior variance using only labels acquired through iteration t.
$\arg\max$	Eligible highest-score input, with declared tie-breaking.
x_{t+1}	Next input for oracle labeling; removed from the pool after acquisition.

$$x_{t+1} = \arg\max_{x \in \mathcal{U}_t} s^2_{f,t}(x) \tag{10.11}$$

$\mathcal{U}_t$ is the eligible unlabeled pool at iteration t. The latent posterior variance s_f ,t squared is computed from labels acquired so far. The selected item is removed from the pool after its oracle label is acquired.

The score is latent uncertainty, not future-observation variance or a guaranteed reduction in average prediction error.

◇ Proposition 10.3: How one additional label reduces posterior variance

Notation for this result

c, c_+	Latent posterior covariance functions before/after the selected observation, already conditional on existing data.
u, v, x	Two query inputs and selected candidate, all in the same feature space.
τ^2	Positive independent measurement-noise variance of the new observation.
$\Delta(x)$	Weighted latent-variance reduction on declared reference inputs if x is measured.
M, r, u_r	Positive reference-set size, index from one through M and corresponding input.
w_r	Nonnegative reference-population weights summing to one.

Assumptions. Condition on all currently labeled data and hold GP hyperparameters fixed. A candidate observation has independent zero-mean Gaussian noise with known positive variance. Candidate inputs are selected without viewing their labels.

With the model fixed, a candidate's covariance reduction can be calculated before its label is acquired. Conditioning subtracts an outer product divided by positive predictive observation variance, so the reduction is nonnegative in every direction. Its size depends on covariance with the query, not the numerical label received. That label changes the posterior mean; it does not enter this fixed-model variance update.

□ Proof *Recommended*

Apply Gaussian conditioning again, now to the current posterior rather than the original prior. The new observation is a scalar, so its covariance inverse is simply a positive reciprocal.

$$c_+(u,v) = c(u,v) - \frac{c(u,x)\,c(x,v)}{c(x,x)+\tau^2} \tag{10.T13}$$

At one query, the covariance-update numerator becomes a square, so the variance reduction cannot be negative. Across several queries it becomes an outer product, yielding the stronger covariance ordering. The acquired label changes the posterior mean through its residual, but no label appears in this covariance update. One can therefore calculate its reduction before acquisition only under the stated fixed-hyperparameter, Gaussian-observation assumptions.

$$\Delta(x) = \sum_{r=1}^{M} w_r \frac{c(u_r,x)^2}{c(x,x)+\tau^2} \tag{10.T14}$$

To value a query for a population, sum its pointwise variance reductions using the fixed nonnegative population weights.

Cohn, Ghahramani and Jordan [10.3] select queries to reduce predictive variance over a target input distribution. For the Gaussian-process setting, Rasmussen and Williams [10.1], Chapter 2, distinguish uncertainty in the latent function from uncertainty in a new noisy observation. With fixed kernel and noise parameters, posterior variance depends on input locations, not observed response values. If those parameters are refitted, responses can affect it indirectly. Decide which uncertainty and target region matter before comparing query rules.

□

Reference inputs and weights describe the deployment population, not permission to query test labels. The variance objective measures model uncertainty, and is not automatically equivalent to lower real prediction error.

▶ Example: Selection and evaluation use different rows

Start with 8 labeled examples, 100 eligible pool inputs, and 200 independent test examples. Before acquisition, the policy sees the pool inputs but only the 8 training targets. It selects pool index 37 by latent standard deviation, retrieves its label, and removes that index from eligibility. The state is now 9 labeled examples, 99 eligible inputs, and the same 200 test examples. Give a random-policy baseline the identical initial labels and one acquisition too. Compare test errors at this equal budget; choosing or training on a difficult test example would contaminate the evaluation.

Interpretation and limits

Do not query the evaluation set. Hidden labels in a simulated pool are acquisition data; test labels remain evaluation-only even when all arrays are locally available.

Algorithm 10.1. Finite-pool GP uncertainty sampling

Source and adaptation. Gaussian posterior variance uses Rasmussen and Williams [10.1], Chapter 2; pool-based uncertainty sampling is described by Settles [10.4]. Request accounting, failures and tie order are explicit policies of this teaching loop.

Notation and assumptions

$\mathcal{D} = (X, y),\ U,\ \mathcal{E}$	Nonempty labeled data, finite eligible inputs, and disjoint evaluation data.
B, q	Nonnegative additional-request budget and requests spent, including failures.
k, σ^2, L	Declared kernel, observation-noise variance, and lower Cholesky factor of the noisy training covariance.
$k_{Xx},\ s^2(x)$	Kernel column between training inputs and candidate x; latent posterior variance.
$P,\ R$	Current fitted GP; record containing request log, posterior history and evaluation-only curve.
$x, y_x, \perp$	Chosen candidate, returned label, and explicit oracle-failure marker.

GP fits only labeled data under the declared mean, kernel, noise and refitting policy. The current posterior provides factor L and triangular solves. Start records the initial posterior; Success appends the acquired pair, new posterior and predeclared evaluation metric; Failure logs the unsuccessful request. These helpers never use evaluation outcomes for acquisition.

Algorithm 10.1

Input: Labeled data, eligible pool, evaluation set and labeling oracle.

$\mathcal{D}, U, \mathcal{E}$

Budget, GP/noise/refit settings and deterministic tie order.

$B \in \{0, 1, \ldots\}$

Output: Acquired data, complete request record and count. $\mathcal{D}, R, q$

01 $q \leftarrow 0$

02 $P \leftarrow \mathrm{GP}(\mathcal{D})$

03 $R \leftarrow \mathrm{Start}(P)$

Algorithm 10.1 (continued)

While $q < B \ \wedge \ U \neq \varnothing$
  For every eligible candidate. $s^2(x) \leftarrow k(x,x) - \|L^{-1}k_{Xx}\|_2^2$
  Use the fixed tie order. $x \leftarrow \arg\max_{u \in U} s^2(u)$
  $q \leftarrow q + 1$
  $U \leftarrow U \setminus \{x\}$
  Exactly once. $y_x \leftarrow \mathrm{Oracle}(x)$
  If $y_x = \perp$
    $R \leftarrow \mathrm{Failure}(R, x, q)$
    Continue To the next request.
  End if
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{(x, y_x)\}$
  $P \leftarrow \mathrm{GP}(\mathcal{D})$
  $R \leftarrow \mathrm{Success}(R, x, y_x, q, P)$
End while
Return $\mathcal{D}, R, q$

Why this works. The displayed variance is latent variance: observation noise enters the training covariance, not an extra candidate-variance term. Read L from the current P; its inverse notation means a triangular solve, not explicit inversion.

Reject nonfinite quantities or materially negative variance. Clip only declared roundoff; stop with the retained record if stabilization fails. Zero budget preserves the initial posterior and empty log. Failed requests consume budget; removed candidates are never queried again. Compare with matched random acquisition; smaller model uncertainty need not mean smaller prediction error.

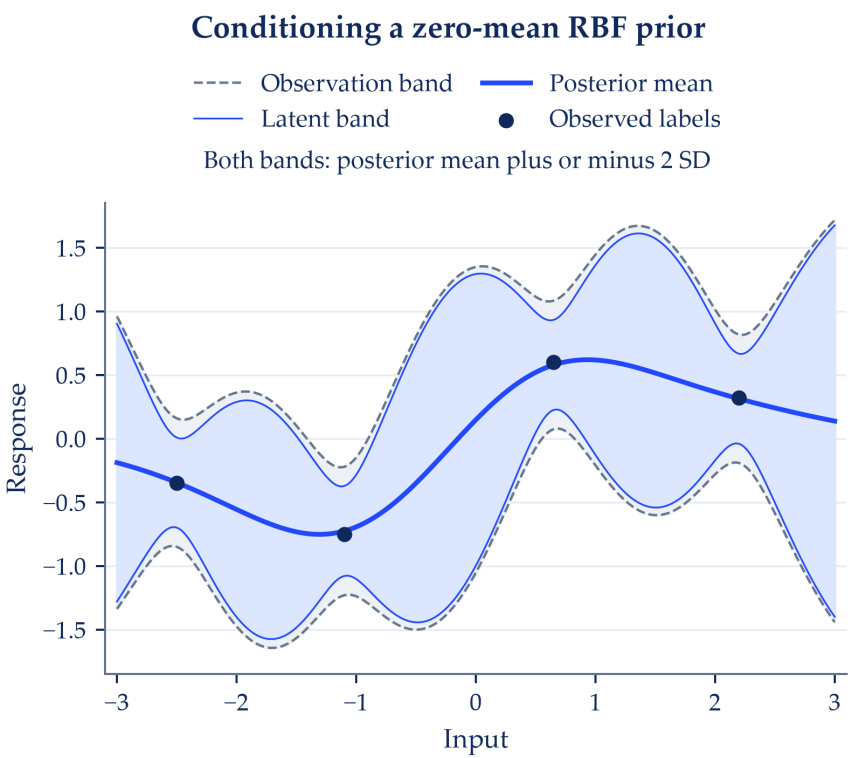


Figure 10.1. Both bands show the blue posterior mean plus or minus two posterior standard deviations (SD). The future-observation band includes observation-noise variance. Conditioning uses four labels, a zero-mean GP, unit signal variance, RBF length scale 0.85 and independent noise standard deviation 0.18. The bands express that model, not empirical coverage under arbitrary shift.

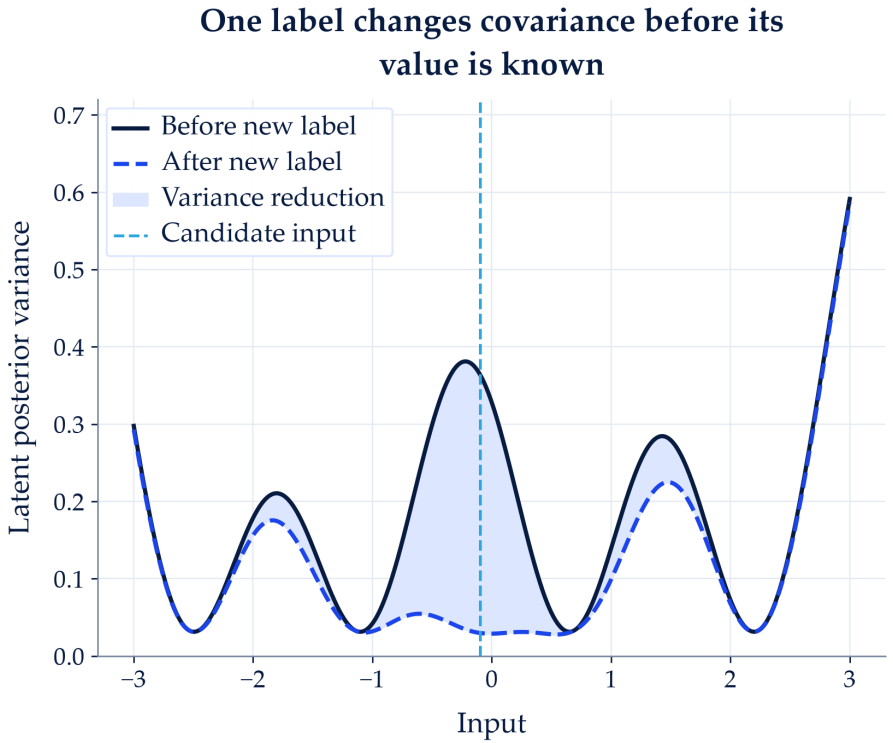


Figure 10.2. Acquiring a label at input -0.1 contracts the latent posterior variance by the exact rank-one covariance update. The reduction is independent of the not-yet-seen label when hyperparameters stay fixed. Direct numerical checks verify nonnegative variance and non-increase at every plotted input.

Essential

10.5 Bayesian optimization asks where the objective is high

Notation for this section

UCB, EI	Upper confidence bound and expected improvement; both maximize the objective here.

Bayesian optimization chooses where to evaluate an expensive objective. Here larger values are better. A cheaper acquisition function scores candidates using the surrogate posterior. Upper confidence bound adds a multiple of latent standard deviation to the posterior mean; kappa sets that optimism. The rule trades predicted value against uncertainty, but a numerical kappa alone supplies no universal confidence or regret guarantee.

Expected improvement weights every possible latent response by how far it exceeds a fixed incumbent threshold, assigning zero below that threshold. Under a Gaussian posterior this average has the displayed density-and-distribution-function formula. A nonnegative offset xi raises the threshold. If posterior standard deviation is zero, use the deterministic positive-improvement limit rather than divide by zero. With noisy observations, the largest observed target may overstate the true incumbent; the fixed-incumbent formula does not resolve that uncertainty.

Expected improvement [10.5] targets optimization rather than uniform predictive accuracy. Its original deterministic setting does not justify treating the largest noisy observation as a noise-free incumbent; the acquisition rule must account for that uncertainty.

Noise-aware Bayesian optimization can integrate uncertainty in the incumbent and pending evaluations, but that requires a more elaborate acquisition model. Even the simple loop must enforce feasible inputs, remove or explicitly permit repeats, record failed evaluations, and respect a budget. Compare search methods using best feasible value versus evaluations and report multiple runs when possible. A high acquisition score is not a safety guarantee, and a synthetic success is not measured evidence of universal sample efficiency.

Jones, Schonlau, and Welch [10.5] developed efficient global optimization for expensive functions, including an improvement-based acquisition strategy. Snoek, Larochelle, and Adams [10.6] apply Bayesian optimization to machine-learning hyperparameters; both address objective improvement rather than general-purpose label informativeness. Balandat and colleagues [10.9] develop Monte Carlo acquisition functions and sample-average optimization in BoTorch. This provides computational tools beyond the fixed-incumbent Gaussian EI calculation here, not evidence that the chapter's simple policy is universally sample-efficient.

Notation for equation (10.12)

x	Eligible input for the next expensive objective evaluation.
$a_{\mathrm{UCB}}(x)$	Upper-confidence-bound score for scalar-objective maximization; larger is preferred.

$\mu(x)$	Current latent objective posterior mean at x.
$s_f(x)$	Nonnegative latent posterior standard deviation, not noisy-response deviation.

$$a_{\mathrm{UCB}}(x) = \mu(x) + \kappa s_f(x) \tag{10.12}$$

For maximization, mu is the latent posterior mean, s_f is its standard deviation, and kappa is a nonnegative exploration coefficient. Select an eligible input maximizing this score.

This is a maximization convention. The name UCB alone does not establish a coverage, safety, or regret guarantee.

Notation for equations (10.13), (10.14), (10.15)

x	Eligible input for expected-improvement evaluation.
$\mu(x)$	Current latent scalar-objective posterior mean at x.
f_{best}	Fixed incumbent objective value for maximization.
Δ	Posterior mean minus incumbent and offset: signed mean advantage.
$s_f(x)$	Latent posterior standard deviation, positive in the standardized expression.
z	Dimensionless Delta divided by latent posterior standard deviation.
$\Phi(z), \phi(z)$	Standard-normal cumulative distribution and density at the standardized advantage.
$a_{\mathrm{EI}}(x)$	Posterior expected positive improvement over fixed incumbent plus offset, for maximization.

$$\Delta = \mu(x) - f_{\mathrm{best}} - \xi \tag{10.13}$$

$$z = \Delta / s_f(x) \tag{10.14}$$

$$a_{\mathrm{EI}}(x) = \Delta\Phi(z) + s_f(x)\phi(z) \tag{10.15}$$

The fixed incumbent f_{best} is a reference objective value, xi is a nonnegative improvement offset, and Phi and phi are the standard normal CDF and density. This formula applies when s_f is positive; at zero use max(Delta,0).

When latent standard deviation is zero, EI is max(Delta, 0). With noisy data, a fixed observed incumbent may be misleading.

◇ Derivation 10.4: Expected improvement from its defining integral

Notation for this result

F, μ, s	Latent objective at the candidate, its scalar posterior mean, and its strictly positive posterior standard deviation.
Z	Standard normal scalar random variable used to represent the posterior objective.
b, f_{best}, ξ	Improvement threshold, fixed incumbent value, and nonnegative required improvement offset, respectively.
δ, z	Posterior mean advantage in objective units, and its dimensionless ratio to posterior standard deviation.
I	Nonnegative scalar random improvement, not an identity matrix; max selects the larger argument.
$t, \phi(t)$	Real integration variable and standard normal density evaluated at that variable.
$\mathbb{E}$	Expectation over the candidate's latent Gaussian posterior, holding the improvement threshold fixed.
$\Phi(z)$	Standard normal cumulative distribution function at the standardized advantage.
EI	Expected improvement for maximization of the latent objective with the declared fixed threshold.

Assumptions. The objective is maximized. At one eligible candidate its latent posterior is Gaussian, and the incumbent and nonnegative improvement offset are fixed numbers when taking the expectation. The first calculation uses strictly positive posterior standard deviation.

Expected improvement is the posterior mean of positive improvement over the fixed threshold. It has a closed Gaussian expression and a continuous deterministic limit when posterior variance vanishes.

Read the complete supporting proof . *Advanced*

The formula balances mean advantage with posterior uncertainty under explicit maximization and fixed-threshold assumptions. It supplies neither a universal sample-efficiency guarantee nor an operational safety constraint.

▶ Example: Exploration can change the next evaluation

For candidates A and B, posterior means are 0.8 and 0.6, with latent standard deviations 0.05 and 0.3. UCB with kappa 2 gives 0.9 and 1.2, choosing the more uncertain B despite its lower mean. For EI, incumbent 0.75 and zero offset give mean improvements 0.05 and negative 0.15, hence standardized improvements

1 and negative 0.5. Integrating positive improvement gives approximately 0.054 and 0.059: B can win even with negative mean improvement. With kappa zero, UCB chooses A. These are assumed-posterior maximization calculations, not measured search performance.

Interpretation and limits

Do not apply a maximization EI formula to a minimization task without reversing the improvement definition. With noisy targets, also avoid treating the largest noisy observation as the true optimum.

Algorithm 10.2. Finite-pool noiseless Bayesian optimization

Source and adaptation. Expected improvement follows Jones, Schonlau and Welch [10.5]; the GP acquisition workflow and EI/UCB comparison are discussed by Snoek and colleagues [10.6]. This finite-pool version uses the maximization convention and explicit failure-budget policies.

Notation and assumptions

$\mathcal{D}, U, n_0$	Nonempty cache of distinct feasible input-value pairs, remaining finite feasible pool, and initial cache size.
B, q, b	Nonnegative integer additional-request budget, requests spent including failures, and best cached objective value.
$\mu(x), s(x), a(x)$	GP latent mean, nonnegative latent standard deviation, and mean advantage at candidate x.
ξ, κ, Φ, ϕ	Nonnegative EI offset, nonnegative UCB multiplier, standard-normal CDF and density.
$e(x), u(x), m$	Expected-improvement score, upper-confidence score, and selected mode: EI, UCB or random.
$R, r, \perp$	Request/incumbent record, returned objective value, and failure marker.
$f, \quad z = a(x)/s(x) \quad (s(x) > 0)$	Noiseless objective and standardized advantage, defined only at positive standard deviation.

GP uses only cached pairs and Cholesky solves under the declared numerical policy. Start returns the cache maximum and records it at count n0. Select maximizes e or u with the fixed tie order, or samples uniformly from U using the recorded seed. Failure logs an attempt without inventing a value; Success records the new incumbent at count n0+q.

Algorithm 10.2

Input: Cached noiseless evaluations and remaining feasible inputs.

$\mathcal{D}, U$

Budget, GP settings, acquisition mode, tie order and random seed.

$B \in \{0, 1, \ldots\}, \quad \xi, \kappa \geq 0$

Output: Cache, request/incumbent record and total attempted evaluation count.

$\mathcal{D}, R, n_0 + q$

$q \leftarrow 0$

$(b, R) \leftarrow \mathrm{Start}(\mathcal{D})$

While $q < B \ \wedge \ U \neq \varnothing$

  Latent predictions only. $(\mu, s) \leftarrow \mathrm{GP}(\mathcal{D}, U)$

  For every candidate. $a(x) \leftarrow \mu(x) - b - \xi$

  Use the positive-deviation branch only when defined.

  $$e(x) \leftarrow \begin{cases} a(x)\Phi(z) + s(x)\phi(z), & s(x) > 0, \\ \max\{a(x), 0\}, & s(x) = 0. \end{cases}$$

  $u(x) \leftarrow \mu(x) + \kappa s(x)$

  $x \leftarrow \mathrm{Select}(U, e, u, m)$

  $q \leftarrow q + 1$

  $U \leftarrow U \setminus \{x\}$

  Exactly once. $r \leftarrow f(x)$

  If $r = \perp$

    $R \leftarrow \mathrm{Failure}(R, x, n_0 + q)$

    Continue To the next request.

  End if

  $\mathcal{D} \leftarrow \mathcal{D} \cup \{(x, r)\}$

  $b \leftarrow \max\{b, r\}$

  $R \leftarrow \mathrm{Success}(R, x, r, n_0 + q, b)$

End while

Return $\mathcal{D}, R, n_0 + q$

Why this works. The objective f is noiseless; cached values, not new oracle calls, define the incumbent. Numerical jitter is not objective noise. A noisy objective requires a separately declared noise-aware incumbent/acquisition policy.

Abort on failed stabilization, materially negative variance or nonfinite calculations, retaining the record. Clip only roundoff-sized negative variances. Compare tail EI on a stable log scale. Zero budget returns the initial incumbent. Failed requests count but contribute no value; the returned count includes initial evaluations. The random arm uses the same initial cache and budget.

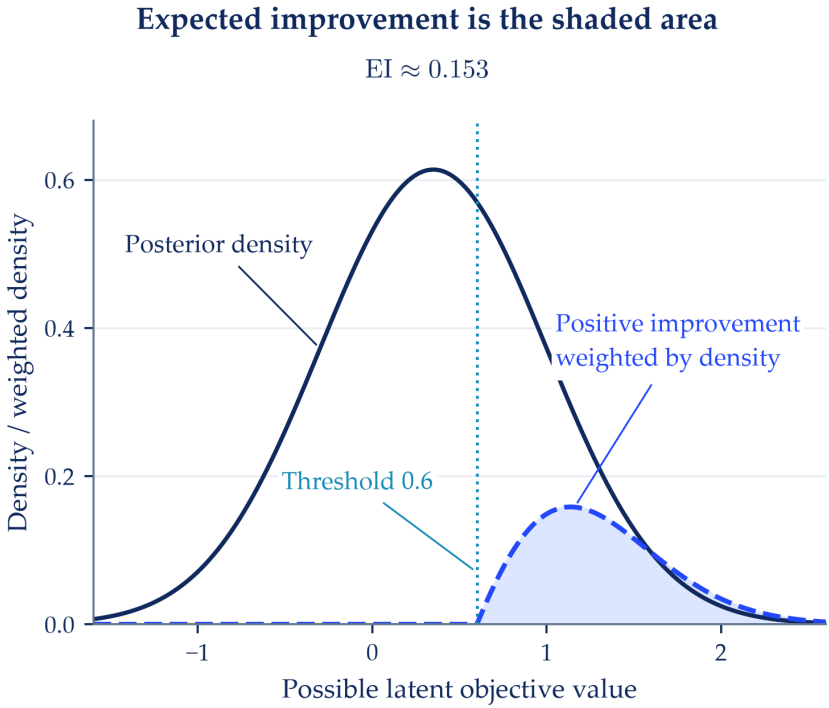


Figure 10.3. Expected improvement for maximization is the integral of positive improvement times posterior density. Here the Gaussian mean is 0.35, standard deviation 0.65 and fixed improvement threshold 0.6. The shaded area agrees with the analytical Gaussian-tail formula to numerical integration tolerance.

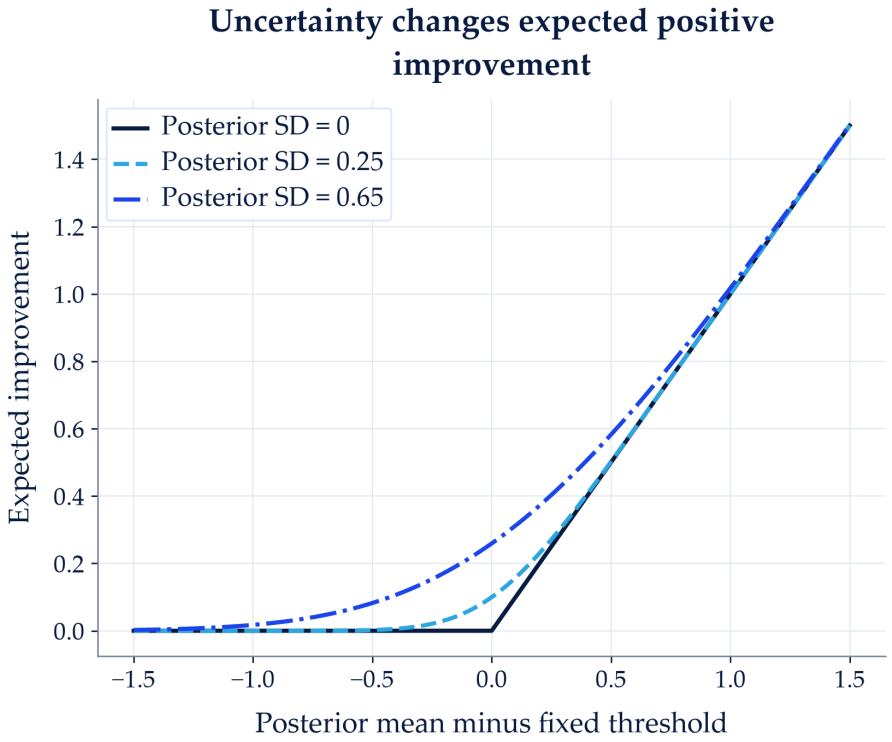


Figure 10.4. Gaussian expected improvement against posterior mean minus a fixed threshold. Curves differ in the latent objective's posterior standard deviation (SD). Zero uncertainty gives the positive-part limit. These curves do not integrate uncertainty in a noisy incumbent.

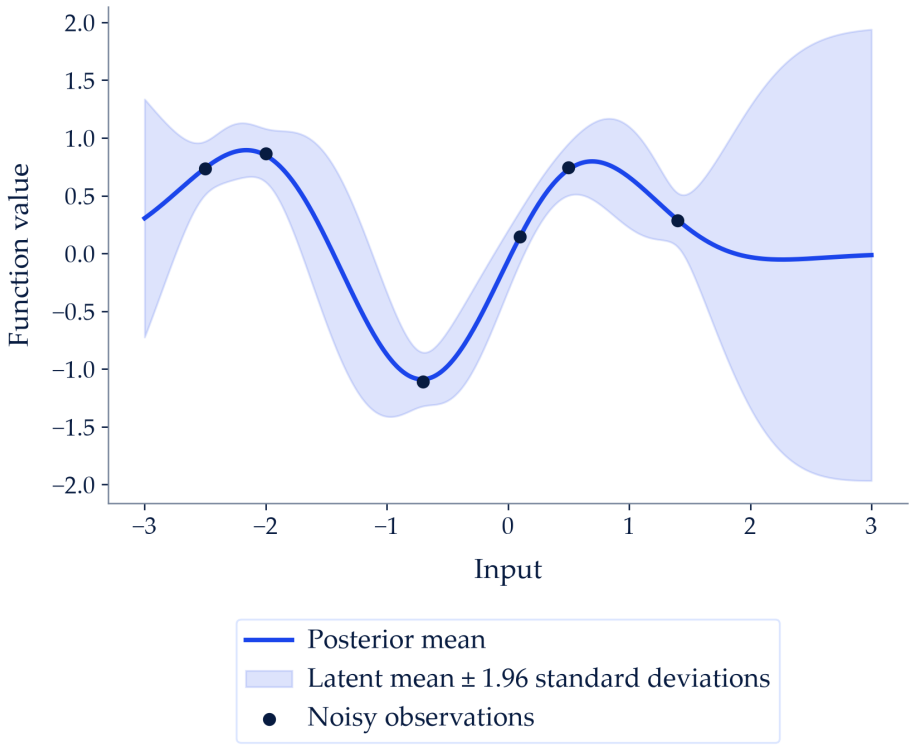


Figure 10.5. Actual zero-mean GP regression with a fixed RBF kernel and known synthetic observation-noise variance. The interval is for the latent function under that model, not for a future noisy measurement and not a guarantee under distribution shift.

Table 10.1. Uncertainty can guide different experiments

Method	Question answered	Decision quantity
GP regression	What values remain plausible after conditioning on observed data?	Posterior mean and covariance under a kernel and noise model.
Active learning	Which label best improves a prediction task?	An acquisition measure aligned with prediction uncertainty or information.

Method	Question answered	Decision quantity
Bayesian optimization	Where should an expensive objective be evaluated next?	An acquisition balancing objective value and exploration.
Random acquisition	What happens without model-directed selection?	A necessary baseline for judging whether acquisition adds value.

Table 10.2. Chapter synthesis

Concept	Main conclusion	Engineering check
Kernel prior	Covariance encodes smoothness and relationships between inputs.	State units, length scales and a justified mean function.
Conditioning	Posterior uncertainty depends on model assumptions.	Separate latent-function variance from future observation noise.
Numerics	Cholesky solves avoid explicit covariance inversion.	Check positive definiteness, jitter and conditioning.
Sequential choice	The acquisition objective must match the scientific goal.	Compare against random choice under the same labeling budget.

Dataset: Diabetes progression. See its first-use portrait [10.10].

Case Study: Replay a limited-label GP study on real diabetes measurements

Problem and data

Real recorded outcomes make it possible to replay a limited-label learning problem without pretending that measurements are being collected live. The task is to compare GP uncertainty sampling with random selection when only eight outcomes are initially revealed and ten further labels can be requested. Eligible patients, validation patients, and final-evaluation patients remain disjoint. The result concerns a retrospective prediction procedure, not a treatment policy, causal experiment, or recommendation about patient care.

Experimental design

Fix a 100-row eligible pool and separate validation and final subsets. For each of three seeds, use the same eight starting observations for both policies. Fit feature and target scaling on those starting observations and retain fixed RBF and noise parameters. Recompute the posterior after each revealed label; the uncertainty policy sees candidate inputs and posterior variance only. Record validation RMSE at every budget, then evaluate both predeclared final-budget policies once on final observations.

Measured Reference Run

The retrospective replay fixed 100 eligible, 100 validation and 100 final rows. For each of three seeds, random and maximum-latent-variance acquisition shared eight initial labels, initial-only scaling, an RBF kernel and ten queries. There was no hyperparameter search. The table gives progression-score RMSE after eighteen labels and the same seed's initial-label-mean reference.

Table 10.3. Measured case-study evidence

Seed	Uncertainty RMSE	Random RMSE	Mean reference RMSE
1001	66.646	66.647	82.515
1002	69.564	79.048	92.552
1003	65.566	66.652	107.782

Uncertainty acquisition had lower final RMSE in every matched pair; the first was nearly tied. At the same label budget, mean validation RMSE favored random acquisition, 59.638 versus 62.739. These finite evaluation and starting sets do not establish a universal acquisition hierarchy.

Failure Analysis And Limits

Nominal 95% observation intervals achieved uncertainty-policy coverage of 0.97, 0.85 and 0.85 across the three final runs. Covariance calculations and informative-looking acquisitions therefore did not guarantee nominal coverage or uniformly improving predictive error. The noise variance was declared, not empirically validated, and replayed historical outcomes are not prospective interventions.

Historical saved run with seeds 1001-1003. Preserve both policies and every queried row; three starting sets are not a population confidence interval.

▶ Example: inspecting the real data

After eight initial labels and ten queries, the uncertainty policy had 18 labels. Its nominal 95% observation intervals covered 97, 85 and 85 of the 100 final outcomes across the three runs: 3, 15 and 15 misses. Exact 95% coverage would imply five misses on average, not exactly five in every sample. These counts assess the declared uncertainty model, not prospective intervention effects.

Dataset source and documentation

Companion: Week 10, Real-Data Case Study notebook. Reproduce the declared run, then investigate a changed condition. No fixed benchmark score is promised.

A Controlled Learning Cycle

The Diabetes replay compares acquisition under matched label-access budgets. Week 10 notebooks connect stable GP solves and acquisition units to prediction improvement versus objective maximization. The Iris laboratory exposes exact RBF conditioning and fixed-order/maximum-variance acquisition. Record one controlled comparison with metric, data roles and explanation; the kernel, optimization and Bayesian-optimization limits below keep this a mechanism transfer rather than a reproduction of every chapter procedure.

Matched practice and simulator protocol · Packaged edition; launch instructions in the guide.

Laboratory 10 | Gaussian Processes And Experiment Selection

Exact RBF GP and fixed-order/maximum-variance pool acquisition.

Browser laboratory dataset: Iris. See its first-use portrait [10.11].

An exact RBF Gaussian process predicts measured Iris petal width and replays which training-pool label was selected next.

Open Laboratory 10 | Use the live view for readable controls.

Transfer: Diabetes limited-label replay → Iris petal length predicting petal width. No hyperparameter optimization, Matérn kernel or Bayesian-optimization execution.

Chapter Capability

Update a Gaussian process with an explicit noise model and compare active acquisition against a budget-matched random baseline.

Week 10: separate practice, solutions and assessment

References

[10.1] Rasmussen, C. E.; Williams, C. K. I. (2006). Gaussian Processes for Machine Learning. MIT Press. Source / DOI

[10.2] Stein, M. L. (1999). Interpolation of Spatial Data: Some Theory for Kriging. Springer. Source / DOI

[10.3] Cohn, D. A.; Ghahramani, Z.; Jordan, M. I. (1996). Active Learning with Statistical Models. Journal of Artificial Intelligence Research 4, 129-145. Source / DOI

[10.4] Settles, B. (2012). Active Learning. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool. Source / DOI

[10.5] Jones, D. R.; Schonlau, M.; Welch, W. J. (1998). Efficient Global Optimization of Expensive Black-Box Functions. Journal of Global Optimization 13, 455-492. Source / DOI

[10.6] Snoek, J.; Larochelle, H.; Adams, R. P. (2012). Practical Bayesian Optimization of Machine Learning Algorithms. Advances in Neural Information Processing Systems 25. Source / DOI

[10.7] Titsias, M. (2009). Variational Learning of Inducing Variables in Sparse Gaussian Processes. Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, PMLR 5, 567-574. Source / DOI

[10.8] Krause, A.; Singh, A.; Guestrin, C. (2008). Near-Optimal Sensor Placements in Gaussian Processes: Theory, Efficient Algorithms and Empirical Studies. Journal of Machine Learning Research 9(8), 235-284. Source / DOI

[10.9] Balandat, M.; Karrer, B.; Jiang, D. R.; Daulton, S.; Letham, B.; Wilson, A. G.; Bakshy, E. (2020). BoTorch: A Framework for Efficient Monte-Carlo Bayesian Optimization. Advances in Neural Information Processing Systems 33. Source / DOI

[10.10] Efron, B.; Hastie, T.; Johnstone, I.; Tibshirani, R. (2004). Least angle regression. The Annals of Statistics, 32(2), 407-451. Source / DOI

[10.11] Fisher, R. A. (1936). Iris. UCI Machine Learning Repository. Source / DOI

Advanced

Expected Improvement From Its Defining Integral

Supporting proof for Section 10.5.

Notation For This Result

F, μ, s	Latent objective at the candidate, its scalar posterior mean, and its strictly positive posterior standard deviation.
Z	Standard normal scalar random variable used to represent the posterior objective.
b, f_{best}, ξ	Improvement threshold, fixed incumbent value, and nonnegative required improvement offset, respectively.
δ, z	Posterior mean advantage in objective units, and its dimensionless ratio to posterior standard deviation.
I	Nonnegative scalar random improvement, not an identity matrix; max selects the larger argument.
$t, \phi(t)$	Real integration variable and standard normal density evaluated at that variable.
$\mathbb{E}$	Expectation over the candidate's latent Gaussian posterior, holding the improvement threshold fixed.
$\Phi(z)$	Standard normal cumulative distribution function at the standardized advantage.
EI	Expected improvement for maximization of the latent objective with the declared fixed threshold.

Assumptions. The objective is maximized. At one eligible candidate its latent posterior is Gaussian, and the incumbent and nonnegative improvement offset are fixed numbers when taking the expectation. The first calculation uses strictly positive posterior standard deviation.

Expected improvement is the posterior mean of positive improvement over the fixed threshold. It has a closed Gaussian expression and a continuous deterministic limit when posterior variance vanishes.

☐ Derivation *Advanced*

$$F = \mu + sZ \tag{10.T15}$$

$$Z \sim \mathcal{N}(0, 1) \tag{10.T16}$$

$$b = f_{\text{best}} + \xi \tag{10.T17}$$

$$\delta = \mu - b \tag{10.T18}$$

$$z = \delta / s \tag{10.T19}$$

$$I = \max(F - b, 0) \tag{10.T20}$$

Improvement is positive exactly when the standard normal variable exceeds the negative standardized advantage. Integrating only over that region makes the truncation explicit.

$$\mathbb{E}[I] = \int_{-z}^{\infty} (\delta + st)\phi(t)\, dt \tag{10.T21}$$

The normal density's derivative equals negative argument times density. Integrating that derivative gives the first moment in the truncated tail. Symmetry gives the probability of the same tail.

$$\int_{-z}^{\infty} \phi(t)\, dt = \Phi(z) \tag{10.T22}$$

$$\int_{-z}^{\infty} t\phi(t)\, dt = \phi(z) \tag{10.T23}$$

$$\text{EI} = \delta\Phi(z) + s\phi(z) \tag{10.T24}$$

The positive-part operation changes by at most the change in its argument. Hence the difference between expected improvement and deterministic positive advantage is bounded by posterior standard deviation times the finite expected absolute standard normal value. It therefore tends to zero as the standard deviation vanishes; use the positive deterministic advantage directly at zero variance. For minimization, define improvement as the incumbent minus the offset minus the objective, and repeat the same derivation with the sign reversed. With noisy data, a random uncertain incumbent requires an expectation over its joint posterior instead of plugging an observed maximum into this fixed-threshold proof.

□

The formula balances mean advantage with posterior uncertainty under explicit maximization and fixed-threshold assumptions. It supplies neither a universal sample-efficiency guarantee nor an operational safety constraint.

Return to the chapter and its references

Advanced

The Marginal Likelihood Balances Residual Fit And Covariance Volume

Supporting proof for Section 10.2.

Notation For This Result

n	Positive number of training observations entering the marginal likelihood.
e, y, m_X	Length-n residual, observation, and fixed prior-mean columns; e is a residual here, not the exponential base.
A, R	Positive-definite n-by-n observation covariance and its lower-triangular Cholesky factor with positive diagonal.
$\mathcal{L}$	Natural logarithm of the marginal probability density of the observations, conditional on inputs and model parameters.
$\det, \log, \pi$	Matrix determinant, natural logarithm, and the circle constant, respectively.
θ_j, ∂_j	Selected scalar covariance hyperparameter and partial differentiation with respect to it; j indexes the fitted kernel/noise parameters.
A_j	n-by-n derivative of the covariance matrix with respect to the selected scalar parameter.
tr	Trace: the sum of diagonal entries of a square matrix.
α	Length-n coefficient column obtained by solving the observation covariance system for the residual.

Assumptions. The observation covariance is positive definite and differentiable in each scalar kernel parameter. The mean is held fixed for the derivative below. Noise parameters are included in the covariance if they are being estimated.

Taking the logarithm of the multivariate Gaussian density yields a quadratic residual term, a log-determinant term, and a normalization constant. Its covariance-parameter gradient follows from differentiating an inverse and a determinant.

☐ Derivation *Advanced*

$$e = y - m_X \tag{10.T1}$$

$$A = RR^T \tag{10.T2}$$

Whitening by the Cholesky factor maps the residual to a standard normal vector. Change of variables contributes the reciprocal determinant of that factor. Taking the logarithm therefore produces the exact normalized density below, rather than an arbitrarily weighted combination of fit and complexity.

$$\mathcal{L} = -\tfrac{1}{2} e^T A^{-1} e - \tfrac{1}{2} \log\det A - \tfrac{n}{2} \log(2\pi) \tag{10.T3}$$

Differentiate the identity relating a matrix and its inverse. The derivative of the log determinant is obtained from its first-order determinant expansion. Both calculations are local and require invertibility, but neither requires diagonal covariance.

$$\partial_j A^{-1} = -A^{-1} A_j A^{-1} \tag{10.T4}$$

$$\partial_j \log\det A = \operatorname{tr}(A^{-1} A_j) \tag{10.T5}$$

$$\partial_j \mathcal{L} = \tfrac{1}{2} \operatorname{tr}\left[(\alpha\alpha^T - A^{-1}) A_j\right] \tag{10.T6}$$

$$A\alpha = e \tag{10.T7}$$

Substituting the inverse derivative into the quadratic term gives the positive residual contribution. Rewriting that scalar as a trace and combining the determinant derivative proves the gradient. When the mean also depends on the parameter, differentiation supplies the additional scalar contribution alpha-transpose times the derivative of the prior-mean column; it must not be silently omitted. Cholesky solves compute the residual term and twice the sum of log diagonal entries computes the log determinant. Optimizing this expression can still have multiple local optima and weakly identified noise and length-scale combinations.

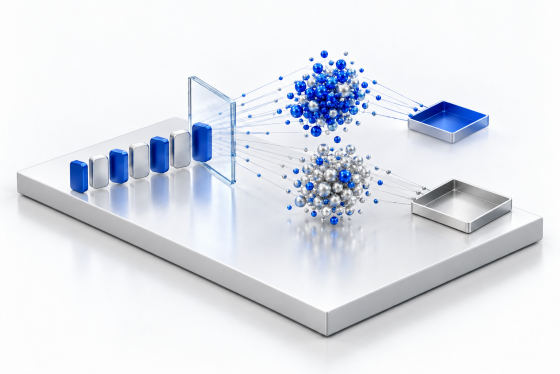
□

Marginal-likelihood optimization learns a covariance model from the observed data, but it does not validate the assumed model family or replace held-out predictive checks.

Return to the chapter and its references

CHAPTER 11

Machine Learning for Text



Reading path

Start with Essential. The other levels offer more depth.

- **Essential**
  11.1 11.2 11.3 11.5
- **Recommended**
  11.4
- **Advanced**
  Proofs 11.3

Worked example Lab task

A document must become a numerical representation before a classifier can use it. Start with a strong lexical baseline and ask what information a different representation preserves or loses.

What You Will Be Able To Do

- Build training-only count and TF-IDF representations for an offline text-classification task.

- Compare multinomial Naive Bayes, logistic regression, and a linear SVM using identical partitions and metrics.

- Understand a fixed pretrained representation as an optional input transformation, while retaining a leakage-safe lexical baseline.

- Diagnose representation failures and distinguish validation selection from final test evaluation.

Before you begin: Classification metrics and validation protocols; Sparse vectors, linear models and regularization; Linear support-vector classification.

Complete and freeze the lexical pipeline before final evaluation.

Algorithm scope: A complete leakage-safe lexical selection procedure with count or TF-IDF vectorization and a classical classifier.

Chapter notation

API	Application programming interface; an API key is a service-access credential.
TF-IDF	Term frequency times inverse document frequency; lexical weighting normalized as specified in Section 11.2.
SVM	Support vector machine, here with linear scores for sparse-text classification.
CPU	Central processing unit: general-purpose processor in the deployment comparison.

Text classification maps variable-length language to a fixed task: route a request, categorize a document or detect a stated message property. A classifier cannot recover distinctions discarded by its representation, and greater representation size does not guarantee useful information. We therefore establish transparent lexical features before considering representations learned from external text.

The core notebook bundles a small literal synthetic corpus, so its required exercises need no downloads, accounts or API keys. It supports software and reasoning checks, not production language-understanding claims.
The chapter experiment uses lexical features.

- *Essential*

11.1 From documents to count vectors

Notation for this section

n, d	Positive document count and fixed fitted-vocabulary size.
X	n-by-d raw document-term counts, one document per row.

Bag of words turns each document into term counts. With n documents and d vocabulary terms, X has n rows and d columns. A document uses few vocabulary terms, so most entries are zero and sparse storage is natural. Tokenization, case normalization, punctuation and vocabulary filtering determine which occurrences count as the same term. These are input-definition choices, not cosmetic cleanup.

In inductive evaluation, learn the vocabulary only from training documents, separately inside each cross-validation fold. Apply that fixed mapping to held-out documents; ignored new terms can leave an all-zero row, which must be tested. Fitting vocabulary on held-out inputs leaks feature information even without labels. Transductive access can be legitimate only under an explicitly different, evaluated information boundary.

Choose token features by asking which distinctions the task needs. Unigrams discard order; word bigrams can distinguish not approved from approved, while fixed-length character sequences capture fragments and spelling variants. Added expressiveness costs vocabulary size and may alter robustness. Stop-word removal or stemming can erase negation, names or punctuation carrying the label signal. Inspect tokenized examples and compare changes on validation data rather than assuming a standard preprocessing recipe is harmless.

Wang and Manning [11.3] show task-dependent benefits from sparse bigram features. Fit vocabulary and weighting only on training text, then hold the split fixed while comparing representation choices; a richer encoding is not automatically better.

Manning, Raghavan, and Schütze [11.1] develop the tokenization, vocabulary, and vector-space foundations used for text classification. A document representation is a fitted transformation of the corpus, so its vocabulary and statistics must respect the same training boundary as the classifier.

Notation for equations (11.1), (11.2)

X	n-by-d document-term counts, usually sparse because most entries are zero.
X_{ij}	Nonnegative integer count of term j in document i under the chosen tokenizer.
$\mathcal{D}_i$	Document-i text before tokenization and feature extraction.
v_j	Vocabulary term j: token or fixed-length token sequence defined by the vectorizer.
count	Occurrences of the selected term in the tokenized document.
i, j, n, d	Document index, vocabulary index, document count and vocabulary size; indices run from one through their counts.
$\mathbb{R}^{n \times d}$	Real n-by-d matrices; raw counts are a nonnegative integer-valued subset.

$$X_{ij} = \mathrm{count}(v_j, \mathcal{D}_i) \tag{11.1}$$

$$X \in \mathbb{R}^{n \times d} \tag{11.2}$$

Document D_i is example i and vocabulary term v_j is feature j. X_{ij} counts occurrences of that term under the chosen tokenizer. The vocabulary of size d is learned from training documents only.

Learn the vocabulary on training documents only. The same mapping is then applied unchanged to validation and test documents.

▶ Example: A sparse representation makes some differences disappear

Fix the coordinate order as invoice, payment, delayed. Then payment payment invoice maps to 1, 2, 0, and delayed payment maps to 0, 1, 1. Reorder the first document or append a word outside this vocabulary: its three coordinates remain unchanged. Any classifier receiving only that vector must treat these inputs identically, because the information was discarded before classification. Test the representation before changing the classifier: relevant bigrams can recover selected ordering distinctions, while a revised tokenizer or vocabulary can retain previously ignored terms.

Interpretation and limits

Do not fit a vectorizer on training plus test documents just because it uses no labels. Vocabulary and corpus statistics are learned preprocessing and belong inside the training partition or fold.

Essential

11.2 TF-IDF reweights lexical evidence

Notation for this section

IDF	Inverse document frequency: training-corpus weight decreasing with term document frequency.
ℓ_2	Euclidean norm; L2 normalization divides a nonzero document vector by its norm.

TF-IDF combines two questions: how often does a term occur here, and how widespread is it across training documents? Term frequency answers the first; inverse document frequency downweights widespread terms. Their product is TF-IDF. Formulas differ, so specify the convention. The laboratory uses raw counts, smoothed inverse document frequency and rowwise L2 normalization, matching a common scikit-learn configuration for manual checking.

Document frequency measures how widely a term occurs, not how often it repeats. Twenty repetitions in one document contribute one to document frequency. Smoothing keeps the weighting well-defined and moderates rare-term behavior. Row normalization then makes nonzero vectors comparable in length, not in informativeness. An empty document or one containing only unknown terms has zero norm and stays zero; it cannot be normalized into a unit vector.

TF-IDF remains a lexical representation. It can emphasize an uncommon misspelling or identifier just as readily as a meaningful domain term. It does

not infer that different words are synonyms or reliably understand negation. Keep vocabulary learning and inverse-document-frequency fitting inside each training fold when tuning. Sparse linear models are often practical partners because high-dimensional text matrices can be informative without being converted to a dense array; unnecessary densification can make an otherwise modest workflow run out of memory.

Salton and Buckley [11.2] analyze term weighting for text retrieval, and Manning and colleagues [11.1] explain its vector-space interpretation. TF-IDF emphasizes informative lexical evidence, while normalization and tokenization determine which comparisons those weights actually support.

Notation for equations (11.3), (11.4)

n	IDF-fitting training-document count, not prediction-batch size.
i, j	Document index over the transformed batch, and vocabulary index from one through d.
d	Positive training-fitted vocabulary size, not document count.
df_j	Count of training documents containing term j at least once, irrespective of repetitions.
tf_{ij}	Raw term-j count in document i under the selected tokenizer and vocabulary.
idf_j	Smoothed term-j IDF, fitted on training documents and reused unchanged.
$\log$	Natural logarithm for the declared smoothed-IDF convention.
$\widetilde{X}_{ij}$	Document-i/term-j TF-IDF before optional rowwise L2 normalization.

$$\mathrm{idf}_j = \log\left(\frac{1+n}{1+\mathrm{df}_j}\right) + 1 \tag{11.3}$$

$$\widetilde{X}_{ij} = \mathrm{tf}_{ij}\,\mathrm{idf}_j \tag{11.4}$$

Here n is the number of training documents, df_j is the number containing term j, and tf_{ij} is its raw count in document i. The smoothed idf and unnormalized TF-IDF value $\widetilde{X}$ use only the training corpus statistics.

This convention uses raw term counts and additive smoothing by one; document frequency counts documents, not total token occurrences.

Notation for equation (11.5)

i	Document row normalized using training-fitted vocabulary and IDF.
$\widetilde{X}_{i,:}$	Length-d unnormalized TF-IDF row; colon selects all vocabulary columns.

$X_{i,:}$	Length-d normalized TF-IDF row, unit Euclidean norm if originally nonzero.
$\|\widetilde{X}_{i,:}\|_2$	Unnormalized row's Euclidean norm across its d vocabulary entries.
d	Vocabulary size and lexical-feature count per transformed row.

$$X_{i,:} = \frac{\widetilde{X}_{i,:}}{\|\widetilde{X}_{i,:}\|_2} \tag{11.5}$$

Each nonzero document row is divided by its Euclidean norm for L2 normalization. A zero row is left zero. Other TF-IDF conventions exist and should not be silently mixed with this one.

Apply the division only to nonzero rows. Empty documents or documents containing only unknown terms remain zero rows.

◇ Proposition 11.1: The geometry and information loss of normalized TF-IDF

Notation for this result

D, diag	d-by-d diagonal matrix of fitted positive inverse-document-frequency weights and the diagonal-matrix construction operator.
x, d	Nonzero length-d nonnegative document-count column and vocabulary size.
j	Vocabulary-term index from one through d.
$z(x), \mathrm{idf}_j$	Unit-normalized TF-IDF column and training-fitted positive weight for vocabulary item j.
c	Positive count-scaling factor, not a class label in this proposition.
u, v	Two unit-normalized nonzero TF-IDF columns in the same fitted feature space.
$R, \mathcal{D}_a, \mathcal{D}_b$	Deterministic fitted representation and two input documents, respectively.
a, b	Identifiers of the two documents being compared, not numerical coordinates in this statement.
h	Any deterministic downstream classifier whose only document input is the represented feature vector.

Assumptions. Fit positive inverse-document-frequency weights on a fixed training corpus, use raw term counts, and normalize each nonzero weighted document by its Euclidean norm. The claims compare documents transformed with exactly the same vocabulary and fitted weights.

Normalized TF-IDF is unchanged by multiplying every count in a document by the same positive factor. For its unit vectors, squared Euclidean distance is twice one minus cosine similarity. Documents with identical unigram counts remain indistinguishable to every deterministic classifier using only these features.

Proof *Recommended*

$$D = \operatorname{diag}(\mathrm{idf}_1, \dots, \mathrm{idf}_d) \tag{11.T1}$$

$$z(x) = \frac{Dx}{\|Dx\|_2} \tag{11.T2}$$

With fixed IDF weights, multiplying every token count by a positive factor multiplies the weighted vector by that factor. Its norm grows by the same factor, so normalization cancels both. This proves invariance to proportional repetition of all counts, not invariance to arbitrary text edits. Editing can change tokenization, n-grams or relative frequencies and thereby change the normalized representation.

$$z(cx) = z(x) \quad (c > 0) \tag{11.T3}$$

$$\|u - v\|_2^2 = 2 - 2u^T v \tag{11.T4}$$

For unit vectors, expanding squared distance gives one plus one minus twice their inner product. That inner product is cosine similarity, so smaller distance and larger cosine give identical neighbor rankings under this normalization. Nonnegative lexical counts and weights restrict their cosine to zero through one. Do not carry that restriction to arbitrary dense embeddings, whose coordinates and cosine can be negative.

$$R(\mathcal{D}_a) = R(\mathcal{D}_b) \implies h(R(\mathcal{D}_a)) = h(R(\mathcal{D}_b)) \tag{11.T5}$$

A deterministic classifier must give the same output for equal feature vectors. Unigram counts ignore word order, and fixed weighting or normalization cannot reconstruct that lost order. Exchanging subject and object while retaining the tokens therefore defeats every classifier restricted to those features, however expressive. This is a representation limit, not a fitting defect. Separately, an all-zero row has no unit-vector geometry and must not be divided by its norm.

□

Normalization changes the metric and deliberately discards absolute length. It does not create semantic equivalence, and fitting new IDF weights on evaluation text would change the representation being assessed.

▶ Example: Document frequency differs from repetition

Use three training documents: payment payment invoice; payment delayed; invoice updated. Payment and invoice each appear in two documents, giving smoothed IDF log of four thirds plus one, approximately 1.288. Delayed appears in one, giving log of two plus one, approximately 1.693. For the first document, payment and invoice weights have ratio two to one. Their common unrounded IDF cancels during L2 normalization, leaving two divided by the square root of five and one divided by the square root of five, approximately 0.894 and 0.447. Repetition changes term frequency, not document frequency.

Interpretation and limits

TF-IDF is not semantic understanding. Rare tokens can be artifacts, and fitting IDF on the complete dataset leaks future corpus information into an inductive evaluation.

Algorithm 11.1. Leakage-safe TF-IDF model selection

Source and adaptation: Manning, Raghavan and Schütze [11.1] for lexical representations and classification; the scikit-learn leakage and pipeline guidance [11.13] for fitting preprocessing within each training fold. The invalid-candidate policy, fixed score/tie convention and final refit are declared orchestration choices in this book.

Notation and assumptions

$\mathcal{D}, \mathcal{D}_{\text{dev}}, \mathcal{D}_{\text{test}}$	Labeled documents and their group/time-respecting development/final-test split.
$\mathcal{C}, c, \mathcal{C}_{\text{ok}}$	Finite fixed configurations, current configuration and configurations valid on every inner fold.
$I_j, V_j, J,\ y_A, y_{\text{dev}}$	Development training/validation subsets for fold j; finite positive fold count; labels of a named subset A and of all development data.
v_{cj}, h_{cj}	Training-fitted vocabulary/IDF vectorizer and classifier for configuration c, fold j.
s_{cj}, s_c, a	Fold score, aggregated configuration score, and direction sign: +1 maximizes, -1 minimizes.

$c^*, v^*, h^*, \widehat{y}$ Selected configuration, final refitted vectorizer/classifier and final-test prediction vector.

Split enforces the declared group/time boundary. VecFit learns vocabulary and IDF only from input documents; transformation stays sparse and leaves zero rows zero. Fit uses its fixed iteration budget. Score fixes labels, metric and undefined-score convention; Aggregate fixes equal-fold or pooled-prediction reporting before results. Invalid excludes and logs empty vocabulary, nonconvergence or other fitting/scoring failures. Valid retains configurations with every fold valid. Save serializes the complete pipeline.

Algorithm 11.1

Input: Labeled documents and declared group/time split and inner folds.

$\mathcal{D}, \{(I_j, V_j)\}_{j=1}^{J}$

Configurations, finite fitting budgets, score convention and tie order.

$\mathcal{C}, a$

Output: Frozen complete pipeline, development scores and final predictions.

$(v^*, h^*, c^*), \{s_c\}, \widehat{y}$

Split $(\mathcal{D}_{\text{dev}}, \mathcal{D}_{\text{test}}) \leftarrow \text{Split}(\mathcal{D})$

For $c \in \mathcal{C}$

  For $j = 1, \dots, J$

    Training documents only. $v_{cj} \leftarrow \text{VecFit}(I_j, c)$

    Include training labels. $h_{cj} \leftarrow \text{Fit}(v_{cj}(I_j), y_{I_j}, c)$

    Validation labels only. $s_{cj} \leftarrow \text{Score}(h_{cj}, v_{cj}(V_j), y_{V_j})$

    On failure Skip its remaining folds. $\text{Invalid}(c)$

  End for

  Aggregate Only when every fold is valid. $s_c \leftarrow \text{Aggregate}(c)$

End for

$\mathcal{C}_{\text{ok}} \leftarrow \text{Valid}(\mathcal{C})$

Return failure if $\mathcal{C}_{\text{ok}} = \varnothing$

Use the fixed tie order. $c^* \leftarrow \arg\max_{c \in \mathcal{C}_{\text{ok}}} a\, s_c$

$v^* \leftarrow \text{VecFit}(\mathcal{D}_{\text{dev}}, c^*)$

Algorithm 11.1 (continued)

15 $h^* \leftarrow \mathrm{Fit}(v^*(\mathcal{D}_{\mathrm{dev}}), y_{\mathrm{dev}}, c^*)$

16 **Freeze** Include label mapping and preprocessing. $\mathrm{Save}(v^*, h^*, c^*)$

17 Exactly once. $\widehat{y} \leftarrow h^*(v^*(\mathcal{D}_{\mathrm{test}}))$

18 **Return** $(v^*, h^*, c^*), \{s_c\}, \widehat{y}$

Why this works. The full development-set label vector is denoted y_{dev}. Labels never enter VecFit. Aggregate retains fold predictions when the declared summary is pooled, rather than averaging incompatible fold scores. Final refitting must pass the same checks; otherwise return failure before predicting final test.

Finite search, folds and fitting budgets bound termination. All development data may be used for refitting after selection, but no final-test documents or labels enter fitting. Test-driven changes require a new development cycle; splitting an already full-corpus-fitted matrix does not repair leakage.

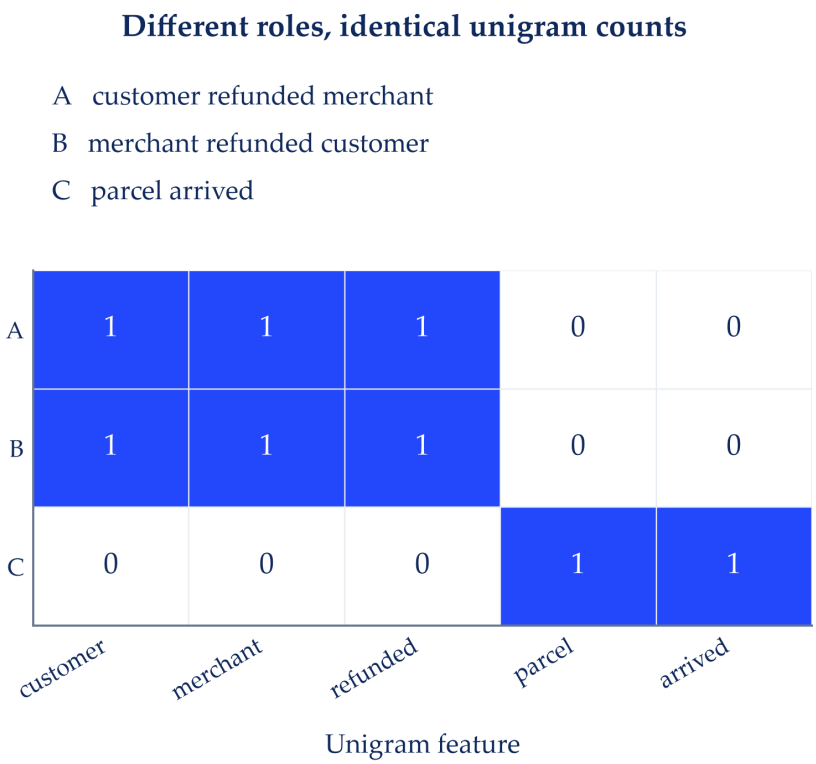


Figure 11.1. Unigram counts for three teaching documents: A is customer refunded merchant; B is merchant refunded customer; C is parcel arrived. Swapping subject and object leaves the unigram counts unchanged. No classifier can recover discarded role information from these count vectors alone.

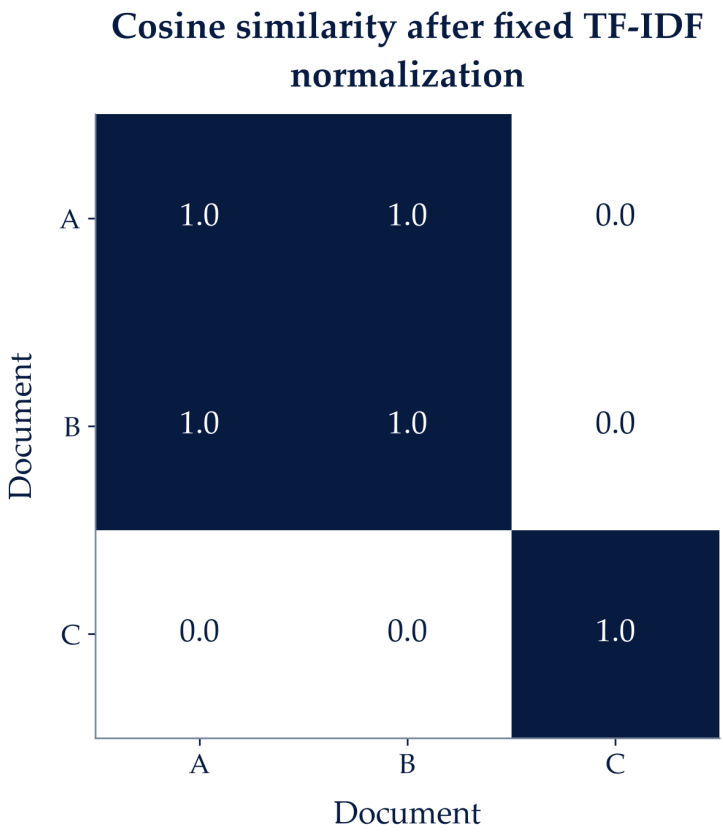


Figure 11.2. Cosine similarity after smoothed-IDF weighting and unit normalization of the same three documents. Documents A and B are identical in this representation despite their different subject-object roles. Vocabulary and IDF are fitted to this tiny corpus solely for arithmetic illustration.

Essential

11.3 Naive Bayes, logistic regression, and linear SVMs

Notation for this section

F_1	Per-class precision/recall harmonic mean; macro F1 gives each class equal weight.

Multinomial Naive Bayes scores documents with class priors and class-specific token probabilities. Given class and document length, token identities are assumed independent; their counts are dependent because the total is fixed. Language violates that assumption, yet classification may work. Additive smoothing prevents unseen class-term pairs from forcing zero probability. Counts support the literal generative model; nonnegative TF-IDF is a practical substitute, not observed token counts or restored independence.

Logistic regression learns discriminative linear class scores and converts them into probabilities through a logistic or softmax link. A linear SVM learns scores using a margin-based objective instead. Both can handle sparse, high-dimensional lexical inputs, and both need regularization choices selected on validation data. SVM decision values are margins, not class probabilities. If calibrated probabilities are required, add an appropriate calibration procedure inside the development data protocol rather than relabeling raw margins as confidence percentages.

Compare representation and classifier as a complete pipeline. Count features with Naive Bayes and TF-IDF with a linear classifier are useful starting combinations, not rules that predetermine a winner. Report a majority-class baseline, class-aware metrics, and a confusion matrix. Macro F1 gives every class equal weight and exposes failures hidden by a dominant class. Accuracy can remain useful when class prevalence and error costs make it relevant, but no single metric substitutes for examining consequential mistakes.

Joachims [11.4] demonstrates support-vector methods for high-dimensional text. Wang and Manning [11.3] motivate serious sparse linear and naive-Bayes-inspired baselines. McCallum and Nigam [11.7] distinguish Bernoulli word presence, including absence, from multinomial occurrence counts. Repeated and absent words affect these scores differently: they are different event models, not interchangeable encodings or a claim that arbitrary count coordinates are independent. State the event model before interpreting its likelihood.

Notation for equations (11.6), (11.7)

x	Length-d nonnegative document vector, typically term counts.
x_j	Term-j count or practical nonnegative feature weight.
c	Candidate label in the declared finite class set.

$s_c(x)$	Unnormalized multinomial Naive Bayes log-score for class c.
$p(c)$	Class prior from training labels or explicitly specified.
v_j	Term at coordinate j of the fixed training vocabulary.
$p(v_j \mid c)$	Class-conditional term-j probability under declared smoothing.
j, d	Vocabulary index from one through d and vocabulary size.
$\log$	Natural logarithm turning class/token probability products into additive scores.
$\arg\max_c, \hat{c}$	Highest-score class selector and resulting predicted label.

$$s_c(x) = \log p(c) + \sum_{j=1}^{d} x_j \log p(v_j \mid c) \tag{11.6}$$

$$\hat{c} = \arg\max_c s_c(x) \tag{11.7}$$

The score s_c for class c combines its prior probability with vocabulary-term probabilities; $\hat{c}$ is the class with largest score. The nonnegative feature x_j is a term count or practical weight, and v_j is term j. Class-independent log-normalization is unnecessary for this classification decision.

The literal multinomial model describes token counts conditional on document length. Its class comparison must therefore use either class-independent length or class priors conditional on that length. For the same document, the multinomial coefficient does not depend on class and cancels from the comparison. Fractional feature weights are a practical scoring extension, not literal counts generated by this model.

Notation for equation (11.8)

$p(v_j \mid c)$	Smoothed term-j probability in the class-c multinomial token distribution.
v_j, c	Vocabulary term and conditioning candidate class.
N_{cj}	Total term-j training count across class-c documents.
N_{cr}	Term-r training count in class c; summing r counts all class-c tokens.
α	Positive smoothing pseudocount per vocabulary term in each class.
j, r, d	Selected-term index, summation index and vocabulary size; both indices run from one through d.

$$p(v_j \mid c) = \frac{N_{cj} + \alpha}{\sum_{r=1}^{d} N_{cr} + \alpha d} \tag{11.8}$$

N_{cj} is the training count of term j among documents in class c. Alpha is a positive additive-smoothing parameter, d is vocabulary size, and r indexes vocabulary terms.

The denominator includes one pseudocount per vocabulary term, so the class-conditional token probabilities sum to one.

◇ Derivation 11.2: Why multinomial Naive Bayes gives a linear log-odds rule

Notation for this result

x, x_j, d	Length-d nonnegative integer token-count column, its j-th count, and fixed vocabulary size.
c, N, j	Class label zero or one, document token count, and vocabulary index from one through d.
θ_{cj}	Strictly positive probability of vocabulary term j in class c, summing to one across that class's vocabulary.
$!, p(x \mid c, N)$	Integer factorial and the multinomial count probability conditional on class and document length.
$p(c \mid x, N)$	Posterior probability of class zero or one, conditional on the observed token-count column and document length.
β_0, β_j	Prior log-odds intercept and vocabulary-term log likelihood-ratio coefficient, respectively.
$\log$	Natural logarithm, turning products of positive probabilities into sums.
π_1, π_0	Positive prior probabilities of the two classes under the declared document-length conditioning convention.

Assumptions. The vocabulary is fixed. Conditional on class and document length, token draws are independent with class-specific strictly positive probabilities. Both classes have positive prior probabilities. To obtain one affine rule with a length-independent intercept, assume document length has the same distribution in both classes, or use a fixed-length conditional classification model with priors specified for that length.

Binary multinomial Naive Bayes has posterior log odds affine in the token-count vector. Its classifier therefore has a linear decision boundary in count space, although its parameters are estimated generatively rather than by logistic-regression optimization.

□ Derivation *Recommended*

$$p(x \mid c, N) = \frac{N!}{\prod_{j=1}^{d} x_j!} \prod_{j=1}^{d} \theta_{cj}^{x_j} \tag{11.T6}$$

$$N = \sum_{j=1}^{d} x_j \tag{11.T7}$$

A count vector hides token order. Under the independent-token model, every ordering with those counts has the same product probability, and counting the orderings yields the multinomial coefficient. Compare two classes for the same document: that coefficient is identical in both likelihoods and cancels from their ratio. Bayes' rule then multiplies the remaining token-likelihood ratio by the prior odds, leaving the class-dependent evidence.

$$\log \frac{p(c = 1 \mid x, N)}{p(c = 0 \mid x, N)} = \beta_0 + \sum_{j=1}^{d} x_j \beta_j \tag{11.T8}$$

$$\beta_0 = \log \frac{\pi_1}{\pi_0} \tag{11.T9}$$

$$\beta_j = \log \frac{\theta_{1j}}{\theta_{0j}} \tag{11.T10}$$

After cancellation, taking logarithms turns token-probability products into a weighted sum of counts plus prior log odds: an affine score. Solving for posterior probability gives a logistic transform. This prediction form resembles logistic regression, but its parameters came from estimating class-specific token probabilities. Discriminative logistic regression instead fits the conditional objective directly. The same functional shape does not imply the same estimates or assumptions.

Check what the model assumes about document length. If length has a class-dependent likelihood, unconditional class priors require its log likelihood ratio in the score. Using priors conditional on length is equivalent, but lets the intercept vary with length; one affine boundary need not serve every length. Fractional TF-IDF weights likewise extend the rule beyond literal multinomial counts. Token independence is a modeling assumption, not a denial of syntax: correlated terms can yield overconfident probabilities despite useful classification.

□

Token coefficients can be interpreted as accumulated evidence under a declared generative model. This is not causal evidence, and a linear score alone does not establish calibrated probabilities.

◇ Derivation 11.3: Additive smoothing is a posterior predictive calculation

Notation for this result

θ, θ_j	Length-d vocabulary probability column for one fixed class and its j-th strictly positive coordinate.
$p(\theta)$	Prior Dirichlet density on the simplex of positive vocabulary probabilities summing to one.
d, j, α	Vocabulary size, coordinate index from one through d, and positive shared prior concentration.
$\propto$	Proportionality up to a normalization constant independent of the probability vector.
N_j, N	Observed training count for term j in this class and the length-d count column; N is a vector here.
a_j, a, A	Posterior concentration for term j, its length-d concentration column, and its positive scalar total.
$p(\theta \mid N)$	Posterior Dirichlet density after observing the class-specific count column.
$B(a)$	Multivariate beta normalizer: product of gamma functions of concentrations divided by gamma of their sum.
e_j	Length-d coordinate vector containing one at position j and zero elsewhere.
$\mathbb{E}, r$	Expectation under the posterior probability distribution and a summation index over vocabulary terms.
Γ, t, u	Gamma function, positive real function argument, and nonnegative integration variable.

Assumptions. Within one class, token counts follow a multinomial model. Token probabilities have a symmetric Dirichlet prior with strictly positive concentration for every vocabulary item. The vocabulary is fixed before this calculation.

The predictive probability of the next token is its observed count plus the prior concentration, divided by total observed tokens plus total prior concentration. The result is positive and sums to one; it is a posterior mean, not in general the posterior mode.

Read the complete supporting proof . *Advanced*

Smoothing prevents unseen class-term pairs from assigning zero probability to a document. It does not repair a wrong tokenizer, data leakage, or conditional dependence among tokens.

▶ Example: Smoothing makes unseen terms possible

A class contains token counts 8, 2, and 0 for invoice, payment, and delayed. With smoothing alpha 1, add one to each count and three to the total count of 10. Probabilities are therefore 9/13, 3/13, and 1/13, which sum to one. One delayed token contributes negative log of 13, approximately negative 2.565, rather than negative infinity. Classification still compares complete scores, including all terms and class priors. A discriminative classifier instead learns separating weights directly, without estimating these class-specific token distributions.

Interpretation and limits

A linear SVM margin is not a probability, and a Naive Bayes probability is not automatically calibrated. Evaluate or calibrate probability quality separately when decisions require it.

Recommended

11.4 Optional bridge: fixed pretrained representations

Counts and TF-IDF provide transparent lexical baselines. Another option is to turn each document into a fixed-length vector produced by an identified pretrained encoder, then fit an ordinary classifier to those vectors. The encoder is a feature transformation learned from external data. Keeping it frozen means its parameters are not updated by the present classification task.

This remains a familiar supervised-learning pipeline: documents become feature rows, a classifier learns from the training rows, and validation selects the declared configuration. A linear classifier on frozen vectors is still a linear classifier in that feature space. Using such vectors does not require deriving or implementing the encoder's training procedure. Mikolov and colleagues develop learned word representations; Reimers and Gurevych develop sentence representations suitable for comparison and downstream tasks. Their papers are listed in this chapter's references.

▶ Example: Count the trainable part

An encoder with 384 output coordinates maps 600 training documents to a 600-by-384 feature matrix. A three-class affine classifier adds 384 times 3 = 1,152 weights and 3 biases, for 1,155 trainable scalars. Freezing the encoder leaves those 1,155 classifier parameters trainable. If the encoder is later adapted using task labels, that is a different training procedure and must be evaluated as such.

The extra representation can carry useful distinctions that a unigram vocabulary discards. It can also be inappropriate for the task's language,

domain or document length. Inspect its model card, license, tokenization and truncation rule. A long message may lose its decisive final sentence if only its beginning is encoded. Caching vectors helps repeated classifier experiments, but the model identifier, revision and inference settings must accompany that cache.

Keep the comparison honest. Use the same eligible messages, labels and held-out partitions for the lexical pipeline and the encoder-based pipeline. Fit any additional scaler, dimension reduction or classifier only on the allowed training data. Tune on validation data and retain the test set for the frozen final choice. Declare the encoder's external pretraining advantage and whether the corpus could overlap with its pretraining data; the compared systems did not necessarily learn from the same information.

Report more than a single accuracy number. Compare the relevant classwise errors, memory, encoding time and later prediction time. A fixed representation may justify its cost in one application while a sparse lexical model is preferable in another. Do not assign arbitrary vectors to words and label the resulting plot as measured pretrained behavior.

The required chapter experiment remains the classical lexical comparison.

Essential

11.5 Compare representations fairly and inspect failures

Compare methods on the same prediction problem: target, eligible examples, partitions and metrics. Keep duplicates, near-duplicate templates and related conversation messages together when they could leak information. Use time-based evaluation when vocabulary or label policies change. Match tuning budgets where practical and disclose pretrained models' external data advantage. That advantage is legitimate, but changes what the comparison measures.

Compare complete systems, not just classifier accuracy. Report fit time separately from feature extraction, prediction latency at a stated batch size, memory use, and deployment dependencies. A lexical model can offer inspectable coefficients and simple CPU serving. A dense encoder may improve wording transfer, but its evaluation must include truncation, model revision, download requirements, and encoding cost. Cache use and hardware

must be stated for timing comparisons. Select against measured task behavior and operational requirements, not representation novelty.

Inspect validation errors by class, length, language, rare terms, negation and source. First ask whether the representation distinguishes the disputed inputs, then whether the label is reliable or the example genuinely ambiguous. Challenge cases test a failure hypothesis; they do not replace a representative evaluation sample. Freeze the full selected pipeline before final testing. Any response to test mistakes starts a new development cycle. The small bundled corpus teaches this process; production claims need representative data and uncertainty estimates.

Wang and Manning [11.3], Sentence-BERT [11.6] and SimCSE [11.9] motivate controlled pipeline comparisons, but their results do not establish performance on this chapter's text-classification split. The original MTEB study [11.12], Sections 4.2–4.3, found different model rankings across embedding tasks and measured speed–quality tradeoffs. Treat those 2023 results as evidence that evaluation must match the intended task—not a current model leaderboard. Compare shared splits and metrics, external pretraining exposure, latency and storage before preferring a richer representation.

Notation for equation (11.9)

$\widehat{R}_{\text{test}}$	Frozen selected pipeline's mean loss on untouched test cases.
$\mathcal{T}$	Test indices excluded from fitting, representation choice and hyperparameter tuning.
n_{test}	Positive test-document count, equal to test-set size.
i	Index over untouched test cases.
$\mathcal{D}_i, y_i$	Test-example-i text and true target.
$\hat{f}$	Frozen pipeline: tokenizer, learned representation, preprocessing and classifier.
ℓ	Declared example loss comparing prediction and true test target.

$$\widehat{R}_{\text{test}} = \frac{1}{n_{\text{test}}} \sum_{i \in \mathcal{T}} \ell(\hat{f}(\mathcal{D}_i), y_i) \tag{11.9}$$

The fixed selected pipeline is $\hat{f}$, T contains untouched test indices, n_{test} is its size, and ell is the declared evaluation loss. Pipeline selection must be completed before test targets influence any modeling decision.

Select the pipeline before inspecting test outcomes. A loss is minimized; do not confuse this risk expression with a larger-is-better score.

▶ Example: A tiny accuracy difference needs context

On 12 test documents, 10 versus 11 correct means one net gain, about 8.333 percentage points, not necessarily one differing decision. The second model could fix two errors and introduce one. A paired correctness table reveals both directions of change. Inspect them without retuning on the test set; a small net count cannot establish general superiority. Use larger representative evaluation samples and repeated development experiments for selection, accounting for wording and ambiguous labels. Compare latency and maintenance separately from uncertain predictive differences.

Interpretation and limits

Do not retune after inspecting test errors and still call the same test score unbiased final evidence. Move into a new development cycle with fresh evaluation data or explicitly report the reuse.

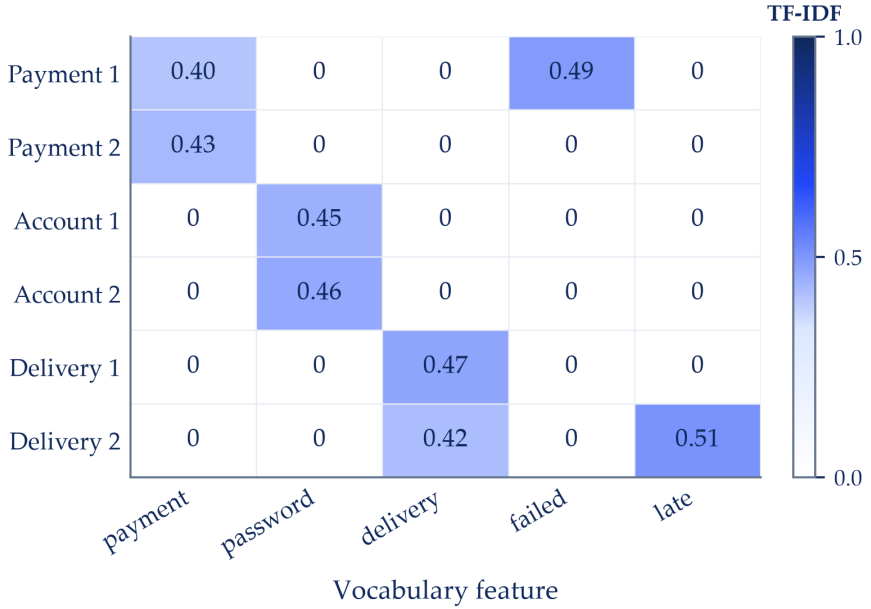


Figure 11.3. Actual TF-IDF values for six explicitly constructed teaching sentences. Shared words create similar coordinates, but synonyms with no lexical overlap need not be close. These are TF-IDF features, not pretrained semantic embeddings. Vocabulary fitting belongs inside the training boundary.

Table 11.1. Text representations and baseline estimators

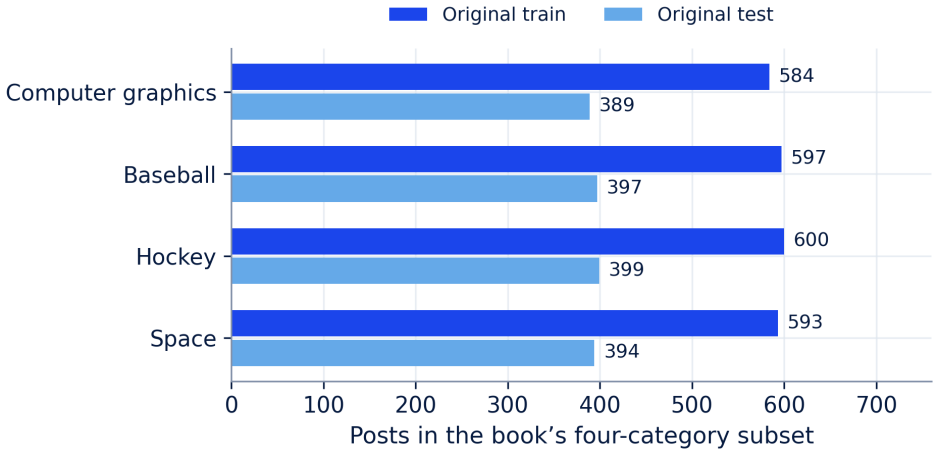
Component	Strength	Constraint
Counts / TF-IDF	Transparent lexical evidence with sparse storage.	Vocabulary misses paraphrases and out-of-vocabulary terms.
Naive Bayes	A compact count-based class model and fast baseline.	Conditional token independence is an approximation.
Linear classifier	An effective sparse-feature decision rule.	Regularization and vocabulary fitting must remain inside development boundaries.
Frozen embeddings	Reuse semantic representations learned by an identified model.	Domain, pooling, truncation and licensing must be checked.

Table 11.2. Chapter synthesis

Concept	Main conclusion	Engineering check
Document contract	Source, target and prediction time determine allowable text.	Remove label-revealing templates and split related documents together.
Vectorization	Vocabulary and IDF statistics are fitted transformations.	Fit them on training folds only, then transform validation documents.
Embeddings	A model identifier alone does not specify the whole representation.	Record revision, tokenizer, pooling, length limit and normalization.
Comparison	Fair representation comparisons share the same evaluation cases.	Report error types and cost, not only one aggregate score.

20 Newsgroups — by-date distribution

18,846 by-date posts	**20** source topics	**Text** variable length



The four categories used in the book, with 2,374 training and 1,579 test posts before the book's additional filtering. The complete by-date distribution contains 20 categories.

The by-date 20 Newsgroups distribution contains 18,846 discussion posts across 20 topics. Training and test posts fall on opposite sides of a date boundary. The book selects four topics and removes metadata and duplicate text; feature dimensions depend on the vocabulary fitted to training posts. [11.10]

Here: four topics supply 2,374 training and 1,579 test posts before additional blank and duplicate filtering. Headers, footers and quotations are removed. Vocabulary size is learned from fitting data, not a fixed property of the corpus.

20news-bydate distribution; scikit-learn loader documentation. Aggregate counts only; no posts reproduced. Source / terms.

Case Study: Classify real newsgroup topics without metadata shortcuts

Problem and data

A real text classifier can exploit mailing metadata instead of the intended topic evidence. Four 20 Newsgroups categories provide an inspectable archive task with two closely related sports topics. Remove headers, footers, and quoted blocks through the declared loader, preserve the official temporal test origin, and document exact-text filtering before model selection. The aim is a defensible lexical comparison on historical messages, not a claim of broad language understanding or successful deployment on contemporary discussions.

Experimental design

Normalize whitespace for exact-duplicate detection, remove blank messages, keep the first occurrence of each training text, and exclude final texts identical to retained original-training texts. Split the remaining original-training records for fitting and validation. Fit one sparse unigram-and-bigram TF-IDF representation on fitting text only, checking one observed term's IDF manually. Compare Naive Bayes, logistic regression, and linear SVM on shared validation features, freeze the macro-F1 winner, then evaluate the filtered official-test subset once.

Measured Reference Run

Four topics retained the official train/test boundary. After heuristic metadata removal and fixed blank/exact-duplicate filtering, there were 2,307 training-origin and 1,532 test-origin messages. The former split into 1,730 fitting and 577 validation documents. A fitting-only vocabulary capped at 15,000 unigram/bigram features served Naive Bayes, logistic regression and linear SVM. Validation macro F1 selected Naive Bayes without refitting.

Table 11.3. Measured case-study evidence

Final procedure	Accuracy	Macro F1	Test count
TF-IDF Naive Bayes	0.884	0.884	1532
Fitting majority	0.255	0.101	1532

Validation macro F1 selected Naive Bayes, 0.884 versus logistic regression's 0.867 and linear SVM's 0.881. The manual IDF check verified feature

construction separately. Final evaluation concerns this frozen lexical pipeline and filtered task, not semantic understanding.

Failure Analysis And Limits

A validation message mentioning Yankee Stadium and GIFs was labeled baseball but predicted graphics: mixed lexical cues remained problematic. That example illustrates ambiguity without proving the classifier's causal reason. Two final messages had no retained features. Exact-text filtering did not detect near duplicates or guarantee removal of metadata shortcuts; the scores are not results for the unmodified twenty-topic benchmark.

Preserve filtering counts, source indices and fitted vocabulary for this saved lexical run. It used no pretrained encoder or external inference service.

▶ Example: inspecting the real data

The saved fitting vocabulary contains space in 176 of 1,730 documents. Smoothed IDF is log((1 + 1,730)/(1 + 176)) + 1, about 3.280, matching the library. Document frequency counts documents containing the term, not total occurrences. A rerun with different fitting documents must recompute the vocabulary and counts; the notebook falls back to another observed term if needed.

Dataset source and documentation

Companion: Week 11, Real-Data Case Study notebook. Reproduce the declared run, then investigate a changed condition. No fixed benchmark score is promised.

A Controlled Learning Cycle

The Newsgroups case audits vocabulary boundaries and lexical shortcuts. Week 11 notebooks fit counts and TF-IDF within training folds, compare Naive Bayes, logistic regression and a linear SVM, and inspect classwise errors. The SMS laboratory isolates count weighting, Naive Bayes contributions and LSA. Record one controlled change with its metric, data roles and explanation.

Matched practice and simulator protocol · Packaged edition; launch instructions in the guide.

Laboratory 11 | Machine Learning for Text

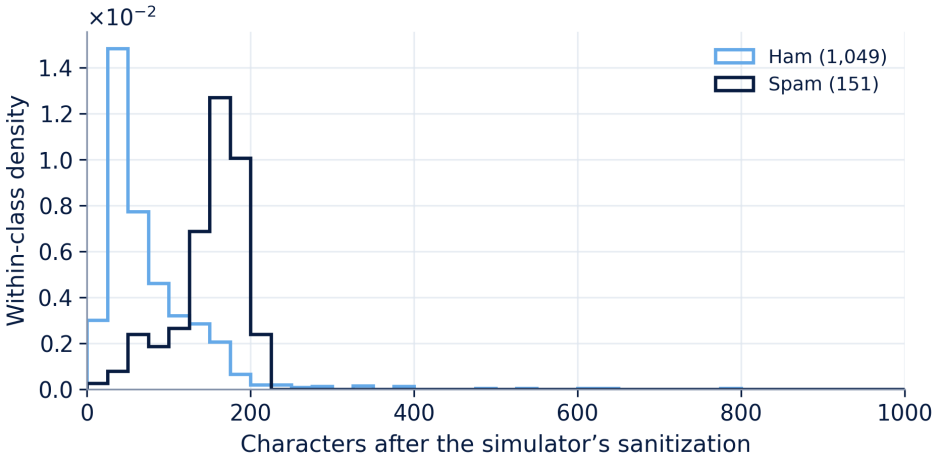
Counts/TF-IDF, Naive Bayes contributions and a training-fitted LSA display.

Training-fitted counts or TF-IDF drive Naive Bayes spam decisions, while token contributions and an LSA projection reveal the fitted representation.

Open Laboratory 11 | Use the live view for readable controls.

SMS Spam Collection

5,574 source lines	**1,200** lab messages	**2** message labels



Lengths in the simulator's 1,200-message sanitized subset, normalized within each class. No message text or contact details are reproduced.

SMS Spam Collection is a historical corpus of short messages labelled legitimate or spam. The simulator uses a deterministic sanitized 1,200-message subset, not the full source collection. Counts and token representations make class imbalance and feature evidence visible without republishing private-looking message content. [11.11]

Here: the archive parser records 5,574 lines; sanitization and deduplication leave 5,136 unique messages. A deterministic subset has 1,049 ham and 151 spam messages. Vocabulary dimensions depend on training; historical messages do not represent current traffic.

Almeida and Hidalgo / UCI, CC BY 4.0. Sanitized subset; no messages reproduced. Source / terms.

Transfer: Four-topic 20 Newsgroups; separate frozen-MiniLM extension → Sanitized historical SMS subset. No pretrained embeddings or fine-tuning; scores are not calibrated probabilities.

Chapter Capability

Fit leakage-safe count and TF-IDF pipelines, compare multinomial Naive Bayes, logistic regression and a linear SVM, and explain their classwise failures on real text.

Week 11: separate practice, solutions and assessment

References

[11.1] Manning, C. D.; Raghavan, P.; Schütze, H. (2008). Introduction to Information Retrieval. Cambridge University Press. Source / DOI

[11.2] Salton, G.; Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. Information Processing & Management 24(5), 513-523. Source / DOI

[11.3] Wang, S.; Manning, C. D. (2012). Baselines and Bigrams: Simple, Good Sentiment and Topic Classification. Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics, Volume 2, 90-94. Source / DOI

[11.4] Joachims, T. (1998). Text categorization with Support Vector Machines: Learning with many relevant features. Machine Learning: ECML-98, 137-142. Source / DOI

[11.5] Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G. S.; Dean, J. (2013). Distributed Representations of Words and Phrases and their Compositionality. Advances in Neural Information Processing Systems 26. Source / DOI

[11.6] Reimers, N.; Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. Proceedings of EMNLP-IJCNLP, 3982-3992. Source / DOI

[11.7] McCallum, A.; Nigam, K. (1998). A Comparison of Event Models for Naive Bayes Text Classification. AAAI-98 Workshop on Learning for Text Categorization. Source / DOI

[11.8] Devlin, J.; Chang, M.-W.; Lee, K.; Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Proceedings of NAACL-HLT 2019, Volume 1 (Long and Short Papers), 4171-4186. Source / DOI

[11.9] Gao, T.; Yao, X.; Chen, D. (2021). SimCSE: Simple Contrastive Learning of Sentence Embeddings. Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, 6894-6910. Source / DOI

[11.10] Rennie, J.; scikit-learn contributors (n.d.). The 20 Newsgroups data set: by-date distribution. Original distribution and official loader documentation. Accessed 8 October 2026. Source / DOI

[11.11] Almeida, T.; Hidalgo, J. (2011). SMS Spam Collection. UCI Machine Learning Repository. Source / DOI

[11.12] Muennighoff, N.; Tazi, N.; Magne, L.; Reimers, N. (2023). MTEB: Massive Text Embedding Benchmark. Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics, 2014–2037. Source / DOI

[11.13] scikit-learn developers (2026). Common pitfalls and recommended practices: Data leakage; Pipelines and composite estimators. Official scikit-learn documentation. Accessed 30 October 2026. Source / DOI

[11.14] van den Oord, A.; Li, Y.; Vinyals, O. (2018). Representation Learning with Contrastive Predictive Coding. arXiv:1807.03748. Source / DOI

[11.15] Chen, T.; Kornblith, S.; Norouzi, M.; Hinton, G. (2020). A Simple Framework for Contrastive Learning of Visual Representations. Proceedings of the 37th International Conference on Machine Learning, PMLR 119, 1597-1607. Source / DOI

Advanced

Additive Smoothing Is A Posterior Predictive Calculation

Supporting proof for Section 11.3.

Notation For This Result

θ, θ_j	Length-d vocabulary probability column for one fixed class and its j-th strictly positive coordinate.
$p(\theta)$	Prior Dirichlet density on the simplex of positive vocabulary probabilities summing to one.
d, j, α	Vocabulary size, coordinate index from one through d, and positive shared prior concentration.
$\propto$	Proportionality up to a normalization constant independent of the probability vector.
N_j, N	Observed training count for term j in this class and the length-d count column; N is a vector here.
a_j, a, A	Posterior concentration for term j, its length-d concentration column, and its positive scalar total.
$p(\theta \mid N)$	Posterior Dirichlet density after observing the class-specific count column.
$B(a)$	Multivariate beta normalizer: product of gamma functions of concentrations divided by gamma of their sum.
e_j	Length-d coordinate vector containing one at position j and zero elsewhere.
$\mathbb{E}, r$	Expectation under the posterior probability distribution and a summation index over vocabulary terms.
Γ, t, u	Gamma function, positive real function argument, and nonnegative integration variable.

Assumptions. Within one class, token counts follow a multinomial model. Token probabilities have a symmetric Dirichlet prior with strictly positive concentration for every vocabulary item. The vocabulary is fixed before this calculation.

The predictive probability of the next token is its observed count plus the prior concentration, divided by total observed tokens plus total prior concentration. The result is positive and sums to one; it is a posterior mean, not in general the posterior mode.

☐ Derivation *Advanced*

$$p(\theta) \propto \prod_{j=1}^{d} \theta_j^{\alpha-1} \tag{11.T11}$$

$$\sum_{j=1}^{d} \theta_j = 1 \tag{11.T12}$$

The multinomial likelihood contributes one power of a token probability per observed occurrence. Multiplying likelihood and prior therefore adds observed counts to the exponents, giving a Dirichlet posterior with count-shifted concentrations.

$$a_j = N_j + \alpha \tag{11.T13}$$

$$A = \sum_{j=1}^{d} a_j \tag{11.T14}$$

$$p(\theta \mid N) = \frac{1}{B(a)} \prod_{j} \theta_j^{a_j-1} \tag{11.T15}$$

Integrating the next-token probability against this posterior multiplies the integrand by one coordinate. Its integral is the ratio of the same Dirichlet normalizer with that coordinate's concentration increased by one to the original normalizer. The gamma recurrence reduces the ratio to one concentration divided by their total.

$$\mathbb{E}[\theta_j \mid N] = \frac{B(a + e_j)}{B(a)} = \frac{a_j}{A} = \frac{N_j + \alpha}{\sum_r N_r + d\alpha} \tag{11.T16}$$

$$\Gamma(t+1) = t\Gamma(t) \tag{11.T17}$$

$$\Gamma(t) = \int_0^\infty u^{t-1} e^{-u}\, du \tag{11.T18}$$

Integration by parts proves the gamma recurrence because both boundary terms vanish for a positive argument. It establishes the normalizer-ratio simplification used above. Every numerator is positive, and the sum of numerators equals the denominator, proving positivity and normalization. As prior concentration grows, the result moves toward a uniform token distribution. A posterior mode, when interior, instead subtracts one from concentrations before normalization; confusing it with the predictive mean gives the wrong smoothing interpretation.

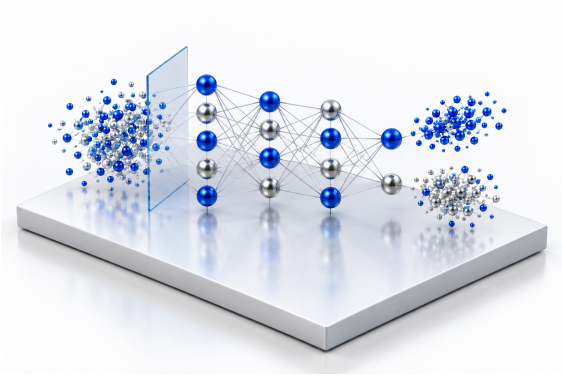
□

Smoothing prevents unseen class-term pairs from assigning zero probability to a document. It does not repair a wrong tokenizer, data leakage, or conditional dependence among tokens.

Return to the chapter and its references

CHAPTER 12

Neural Networks: Foundations and Practice



Reading path

Start with Essential. The other levels offer more depth.

Essential
12.1 12.2 12.3 12.4 12.5

Recommended
Proofs 12.1

Advanced
Proofs 12.2

Worked example Lab task

A neural network extends familiar regression ideas by learning intermediate features. Trace one prediction and its derivatives, then complete a small training and validation cycle before comparing it with a simple baseline.

What You Will Be Able To Do

- Construct a neural unit, a layer, and a complete two-layer network with explicit dimensions.
- Explain how activations turn affine combinations into learned nonlinear features.
- Derive every gradient of a mean-squared-error network and verify the calculation numerically.

Before you begin: Linear regression and gradient descent; Matrix multiplication, derivatives, and the chain rule; NumPy arrays and classification losses.

Trace the scalar network at fixed weights, check its derivatives and complete a small training and validation cycle.

Algorithm scope: A tanh-network gradient step, an independent central-difference audit and a small mini-batch training and validation procedure.

Shared chapter notation

e	Base of the natural exponential function.

$\mathbf{1}_n$ Length-n column of ones broadcasting a shared bias over examples; n is the positive batch-example count.

Chapter notation

$\tanh$ Hyperbolic tangent: a smooth, zero-centered scalar activation with values strictly between negative one and one.

Linear and logistic regression combine fixed input features with learned weights and a task-specific output rule. A neural network also learns intermediate features. It extends the earlier models by composing their operations so that the representation itself can adapt to the target.

A small tanh regressor supplies a running calculation from input to prediction, loss and every parameter derivative. Finite differences independently check the hand-derived gradients. The final section connects these gradients to a small training and validation procedure.

Essential

12.1 From linear models to a neural network

A neural unit reuses a familiar building block: an affine score followed by an activation. Weights set an input-space direction and bias an offset. Identity activation leaves an affine model; a probability-interpreted sigmoid gives binary logistic regression. A hidden unit supplies a feature to another learned operation instead of the final answer. Its new role, not a new kind of arithmetic, builds the network.

A layer evaluates several units on the same input. Arrange their weight vectors as columns of a matrix, so the number of columns is the layer width. For a batch, store examples as rows. Matrix multiplication then computes every unit's preactivation for every example at once, while the same bias vector is repeated over all rows. The forward calculation below first forms hidden scores, then hidden features, then one scalar regression prediction per example.

Track examples, input features, and hidden units separately. For 32 examples, 6 features, and 10 hidden units, the input matrix is 32 by 6 and hidden activations are 32 by 10. Parameters total 81: 60 hidden weights, 10 hidden biases, 10 output weights, and one output bias. Changing batch size changes neither that count nor the represented function. A library may store weight matrices transposed; check the multiplication and bias-broadcasting conventions rather than equating storage shape with a different model.

Keep both predictions and scalar targets as columns so each residual compares matching examples. A flat target array can broadcast against a prediction column to create an all-pairs residual matrix instead. That different loss can be

finite and differentiable, so successful execution is no safeguard. Assert matching prediction and target shapes before subtraction; the assertion enforces the intended pairing in the mathematical model.

Rosenblatt's perceptron [12.1] is a foundational example of a learned decision rule built from weighted inputs. Goodfellow, Bengio, and Courville [12.4] develop the modern feedforward view in which composing affine maps with nonlinear activations produces learned intermediate representations.

Notation for equation (12.1)

x, w	Input and weight columns of length d; d is the number of features entering this unit.
d	Positive number of scalar input features entering the unit.
b	Scalar bias of the unit, shared across inputs.
z	Scalar preactivation formed before the nonlinear response.

$$z = w^T x + b \tag{12.1}$$

One unit forms a scalar preactivation from an input column and a weight column of equal length.

Notation for equation (12.2)

z	Scalar preactivation of one unit.
ϕ	Chosen scalar activation function; the same rule is applied coordinatewise to a layer.
a	Scalar output of the unit, used as a feature or final response.

$$a = \phi(z) \tag{12.2}$$

The scalar activation becomes either an output or a learned feature passed to the next layer.

Notation for equation (12.3)

X	Real n-by-d matrix storing one input example per row.
W_1	Real d-by-h hidden-weight matrix, with one column per hidden unit.
b_1	Length-h hidden-bias column, shared over all examples.
Z	n-by-h matrix of hidden preactivations.
n, d, h	Positive batch size, input feature count, and hidden width.

$$Z = X W_1 + \mathbf{1}_n b_1^T \tag{12.3}$$

The input matrix has n rows and d columns. Hidden weights have d rows and h columns, so the preactivation matrix has n rows and h columns.

Notation for equation (12.4)

Z, H	n-by-h hidden preactivation and activation matrices, respectively.
ϕ	Activation applied independently to each matrix entry; it is tanh in the running example.
n, h	Number of batch examples and hidden units, respectively.

$$H = \phi(Z) \tag{12.4}$$

Apply the activation independently to each entry. The hidden feature matrix retains the n-by-h shape.

Notation for equation (12.5)

H	n-by-h hidden feature matrix from the preceding activation.
W_2	h-by-one column of output weights.
b_2	Scalar bias shared by all scalar output predictions.
$\widehat{y}$	n-by-one column of unrestricted scalar regression predictions.
n, h	Batch size and hidden width.

$$\widehat{y} = H\,W_2 + \mathbf{1}_n b_2 \tag{12.5}$$

An h-by-one output-weight column combines the hidden features; a scalar output bias gives an n-by-one prediction column.

▶ Example: A forward pass that we can follow backward

Use one input, one tanh hidden unit, and one scalar output: input 2, hidden weight 0.5, output weight 3, and both biases zero. First, the hidden preactivation is 0.5 times 2, or 1. Next, tanh of 1 is approximately 0.762. Multiplying its unrounded value by 3 gives prediction approximately 2.285. The two weights and two biases are four trainable parameters. Retain their initial values and target 1 for every derivative check below; rounded displayed intermediates are not substituted into later calculations.

Interpretation and limits

A compatible matrix product does not establish the intended example alignment. State the storage convention and match prediction and target columns before defining the loss.

Essential

12.2 Activation functions and learned features

Notation for this section

ReLU	Rectified linear unit: the activation returning zero for a nonpositive argument and returning the argument for a positive one.

The activation creates the network's nonlinearity. Matrix multiplication alone does not: composing affine layers still gives an affine map. With activation, a hidden coordinate responds differently across input regions. Training moves those regions and changes how later layers combine them. This is what learning a hidden representation means.

The sigmoid maps a real score into the interval between zero and one. Its derivative is largest near zero and becomes small in either saturated tail. Tanh is centered around zero and takes positive as well as negative values, but it also saturates. In the running example its input is 1, where the derivative is approximately 0.42. That local slope will determine how much prediction error reaches the hidden weight.

ReLU keeps positive preactivations and suppresses negative ones. Away from zero its derivative is either one or zero; at zero a derivative convention is required. Its output is unbounded above, unlike sigmoid and tanh. These properties explain distinct response and gradient patterns, not a universal ranking of activations. The separate plots show each function and its derivative on its own scale.

Separate hidden activations from output and loss conventions. Our regressor leaves its scalar output unrestricted. Binary cross-entropy expecting logits takes one raw score; multiclass cross-entropy takes one per class. An extra sigmoid or softmax changes the intended objective. Chapter 3 develops this distinction. Here the
numerical features show how nonlinearity changes the represented map.

Cybenko's approximation theorem [12.2] is an existence result, not a training or finite-data guarantee. Glorot, Bordes and Bengio [12.7], Section 3.1, explain how rectifiers produce exact-zero activations and input-dependent active paths. They also discuss zero-gradient and conditioning difficulties. Sparse activation is not the same as pruning weights, and more sparsity need not improve prediction. Examine active units and gradients rather than interpreting a ReLU network as automatically easy to optimize.

Cybenko [12.2] proved an approximation result for networks with a sigmoidal hidden layer under explicit function-domain assumptions. It establishes representational capacity, not an optimization guarantee or a claim that any

small trained network will generalize. Glorot, Bordes, and Bengio [12.7] empirically study rectifiers and their sparse activations. Leshno and colleagues [12.8] characterize approximation capacity under explicit activation assumptions and emphasize the role of thresholds; this is not a guarantee that training finds the required weights.

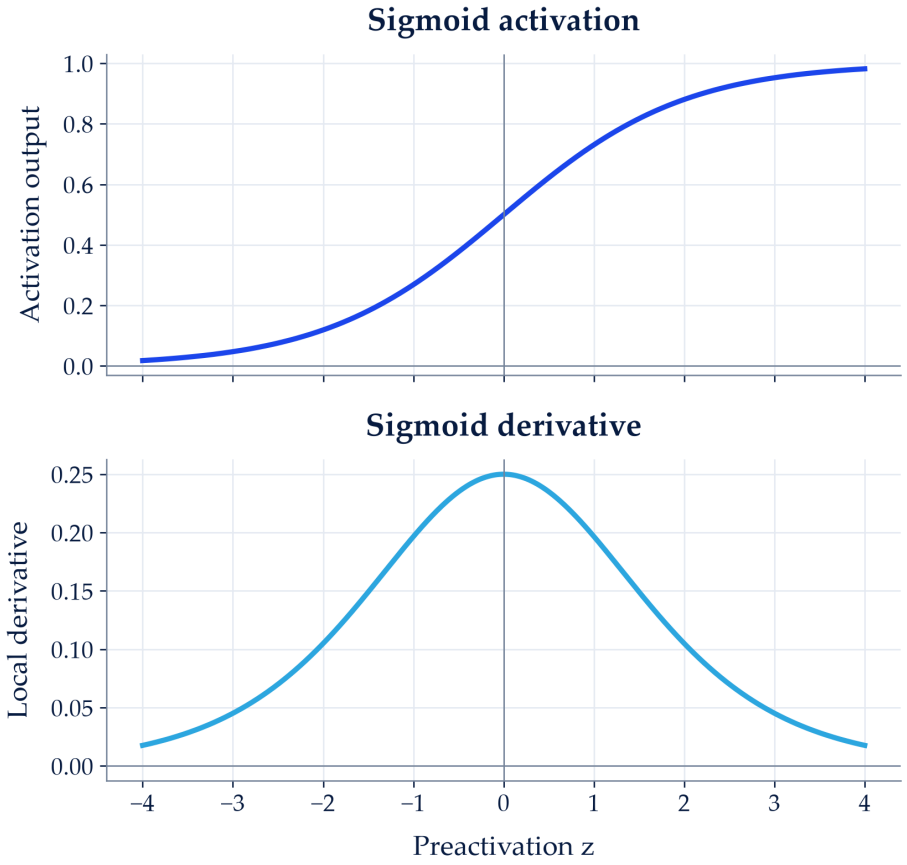


Figure 12.1. Sigmoid (top) and its derivative (bottom) are plotted against preactivation z. The output lies between zero and one; the derivative peaks at 0.25 and tends to zero in either saturated tail.

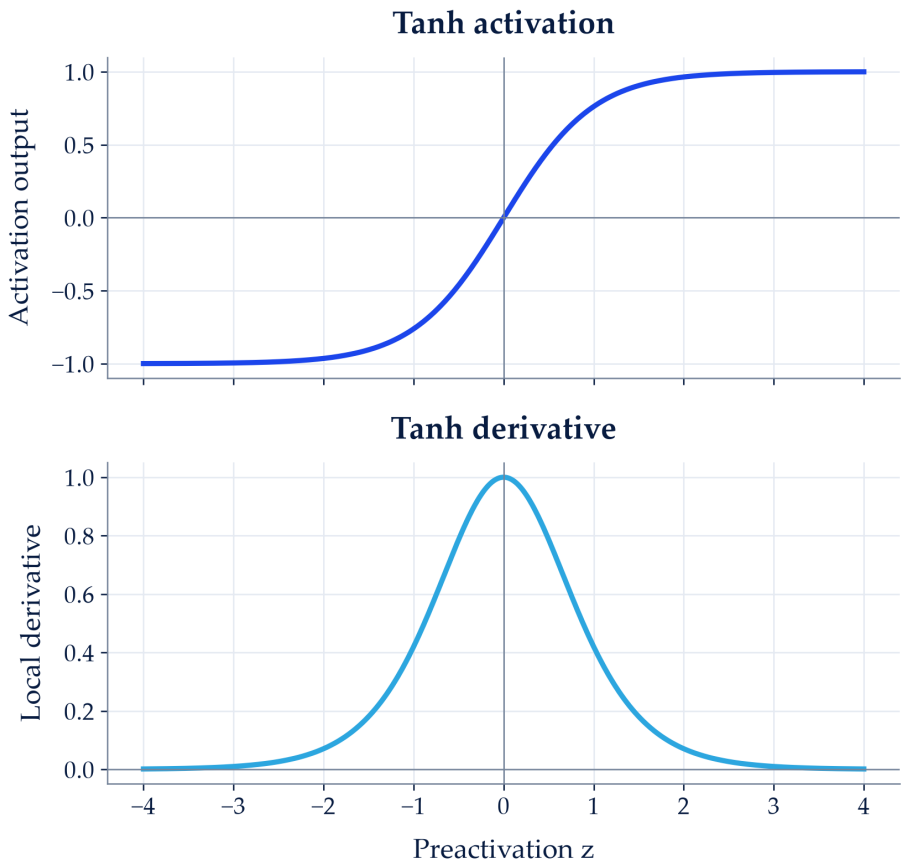


Figure 12.2. Tanh (top) and its derivative (bottom) are plotted against preactivation z. The output lies between minus one and one; the derivative peaks at one and tends to zero in either saturated tail.

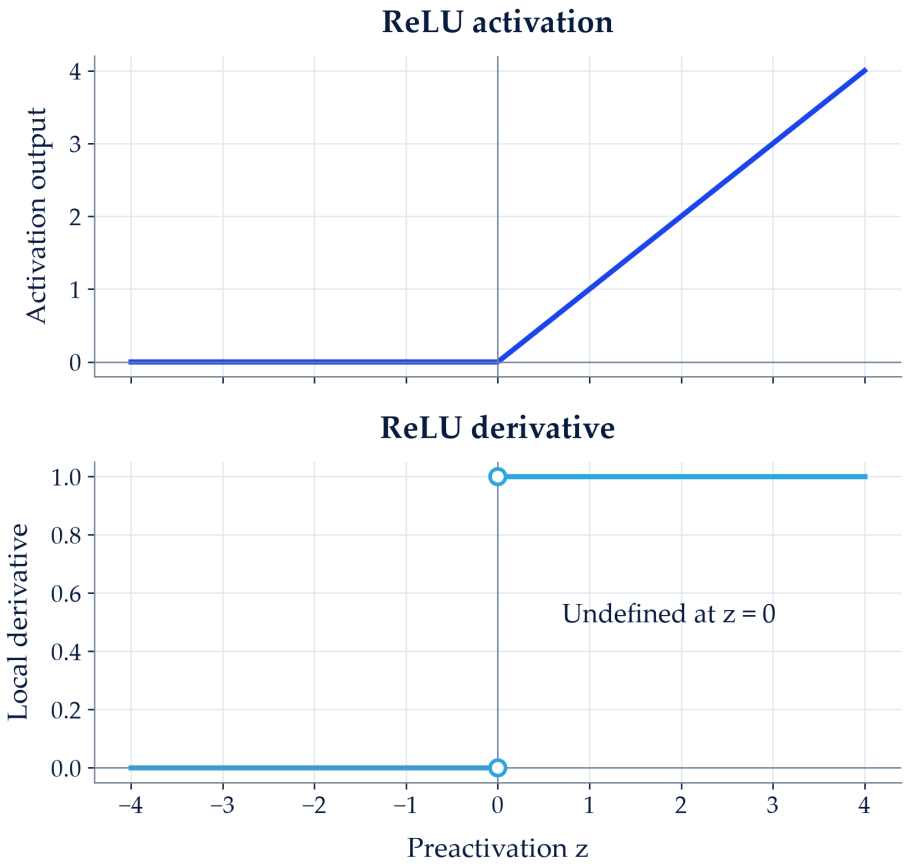


Figure 12.3. ReLU (top) and its derivative (bottom) are plotted against preactivation z. The output is zero for negative inputs and equals the input for positive inputs. Its derivative is zero on the negative side and one on the positive side, and is undefined at zero. Automatic differentiation selects a declared convention at the kink.

Notation for equation (12.6)

z	Real scalar preactivation of one unit.
σ	Logistic sigmoid, taking values strictly between zero and one.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{12.6}$$

The sigmoid is a smooth bounded activation and also the probability link used in binary logistic regression.

Notation for equation (12.7)

z	Scalar preactivation at which the derivative is evaluated.
$\sigma, \sigma^{'}$	Logistic sigmoid and its ordinary derivative with respect to its scalar argument.

$$\sigma^{'}(z) = \sigma(z)\big(1 - \sigma(z)\big) \tag{12.7}$$

Saturation makes the derivative small even though the function remains differentiable.

Notation for equation (12.8)

z	Real scalar preactivation of one unit.
$\tanh$	Hyperbolic tangent, taking values strictly between negative one and one.

$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{12.8}$$

Tanh is an odd, zero-centered activation with outputs strictly between negative one and one.

Notation for equation (12.9)

z	Scalar hidden preactivation.
$\tanh, \tanh^{'}$	Hyperbolic tangent and its derivative with respect to the preactivation.

$$\tanh^{'}(z) = 1 - \tanh^{2}(z) \tag{12.9}$$

This local derivative is the factor needed to pass an error signal through the running network's hidden activation.

The exponent squares the scalar activation value, not a matrix.

Notation for equation (12.10)

z	Real scalar preactivation.
ReLU	Rectified linear unit.
$\max$	Maximum of the displayed scalar arguments.

$$\mathrm{ReLU}(z) = \max(0, z) \tag{12.10}$$

The rectified linear unit is continuous and piecewise affine.

Notation for equation (12.11)

z	Scalar preactivation, restricted here to nonzero values.
ReLU'	Ordinary derivative of the rectified linear unit away from its corner.

$$\mathrm{ReLU}'(z) = \begin{cases} 0, & z < 0, \\ 1, & z > 0. \end{cases} \tag{12.11}$$

The classical derivative is undefined at zero; a software convention there is not a claim of differentiability.

At zero the classical derivative does not exist; a chosen software value only defines the backward convention there.

◇ Proposition 12.1: Affine collapse and the geometry of fixed ReLU regions

Notation for this result

x, a_0	Input and initial-activation columns of length d-zero, the feature count.
d_0	Positive input feature count, before the first affine layer.
l, L	Layer index from one through finite network depth L; here L counts layers, not loss.
d_{l-1}, d_l	Layer-l input and output widths.
a_l, b_l	Length-d-l activation and bias columns.
W_l	Weights with input-width rows and output-width columns.
D_l	Hidden-layer d-l-by-d-l diagonal active-ReLU selector; final output remains affine.

Assumptions. A feedforward network has finitely many affine layers. For the first claim all activations are identities. For the second claim hidden activations are coordinatewise ReLUs and the output layer is affine. A single example is represented by a column vector.

Stacking affine layers without nonlinear activations still gives one affine map. ReLUs change the picture by selecting which coordinates pass through: for a fixed activation pattern, replace each active unit by its affine input and each inactive unit by zero. The resulting formula is affine on a possibly empty polyhedral region. A finite network joins these regions continuously, so it is piecewise affine, not globally affine in general.

□ Proof *Recommended*

$$a_0 = x \tag{12.T1}$$

$$a_l = W_l^T a_{l-1} + b_l \tag{12.T2}$$

Compose the first two affine layers by substituting the first output into the second: the weight matrices multiply and the first bias is transformed before the next bias is added. The result is still affine. Repeating this substitution proves collapse through any finite number of affine layers. Depth changes parameterization and optimization, but without nonlinear activations it does not enlarge this function class.

$$a_l = D_l(W_l^T a_{l-1} + b_l) \tag{12.T3}$$

Freeze a proposed ReLU activation pattern by fixing its diagonal activation matrices. Within that pattern, induction makes every hidden activation affine in the original input. Check consistency: active preactivations must be nonnegative and inactive ones nonpositive. Each condition is then a linear inequality; their intersection is a possibly empty polyhedron. Finitely many hidden units give finitely many patterns, whose regions cover input space and may share boundaries.

Affine maps and ReLUs are continuous, so their composition is continuous. At a boundary, the changing ReLU coordinate has zero preactivation, so either adjacent formula yields the same value. Interior Jacobians are constant on each region. Derivatives at boundaries need not exist, and software's chosen derivative there does not make the represented function globally smooth. No exponential advantage of depth for every target function follows merely from this regional description.

□

ReLU expressivity can be inspected through learned breakpoints and activation regions. Piecewise affine does not mean globally linear, but it also does not provide nonzero classical second derivatives inside the regions.

▶ Example: Two learned features can change the slope

For a scalar input, add ReLU of the input to ReLU of its negative. At input -2, the features are 0 and 2; at input 3, they are 3 and 0. Their sum is therefore the absolute value: slope negative one for negative inputs and positive one for positive inputs. Without the activations, the two linear terms cancel everywhere. The corner at zero also explains why the running gradient check uses smooth tanh: a central difference at a ReLU kink is not evidence for an ordinary derivative there.

Interpretation and limits

An activation's derivative is evaluated at the current preactivation. Reusing a constant derivative or applying a matrix square instead of coordinatewise squaring differentiates a different computation.

- *Essential*

12.3 From prediction error to backpropagation

Begin backpropagation by specifying one scalar objective. Here each example has one real-valued target and the loss is mean squared error. Average over examples, not hidden units; its normalization enters the derivative once. Changing the output task changes the loss and the initial backward derivative. The hidden layers still propagate that derivative using the same chain rule.

Start with the prediction-column derivative of the mean loss, which already contains division by the number of examples. For each output weight, multiply each incoming error by that weight's hidden feature and sum over examples. For the output bias, sum the incoming errors because its input is constant one. Do not average again: that would divide by batch size twice. The matrix products below implement these feature-weighted sums, not an additional loss reduction.

The same error must also pass into the hidden features. Each hidden coordinate receives the prediction derivative multiplied by its output weight. Passing through tanh multiplies that signal by the activation's local slope. Finally, the hidden-weight derivative pairs that preactivation signal with the original input. The complete derivation below writes these steps as matrix operations and includes both bias derivatives, so every parameter in the forward model is accounted for.

Backpropagation is therefore reverse evaluation of local derivatives along an existing computation, with contributions added wherever paths meet. It is not a separate model or an approximation to the chain rule. A correct gradient has the same shape as its parameter. That necessary shape check must be followed by a numerical value check: a wrong sign or an extra batch-size factor can preserve every dimension.

Rumelhart, Hinton, and Williams [12.3] demonstrated how back-propagated errors can train intermediate representations. The derivation here follows the same chain-rule principle: each parameter receives the effect of its local change on the final scalar loss.

Notation for equation (12.12)

L	Scalar mean squared-error objective on the batch.
n, i	Positive batch size and example index from one through n.
$\widehat{y}_i, y_i$	Predicted and observed scalar response for the same example.

$$L = \frac{1}{n}\sum_{i=1}^{n}(\widehat{y}_i - y_i)^2 \tag{12.12}$$

The scalar objective averages squared residuals over the batch using one scalar target per example.

Notation for equation (12.13)

G_2	n-by-one loss derivative with respect to the prediction column.
$\widehat{y}, y$	Matching n-by-one prediction and target columns.
n	Positive number of batch examples.

$$G_2 = \frac{2}{n}(\widehat{y} - y) \tag{12.13}$$

The n-by-one incoming prediction gradient already includes the mean-loss normalization.

No additional division by n is applied when forming the later weight and bias gradients.

◇ Derivation 12.2: Backpropagation with an explicit batch and mean-loss convention

Notation for this result

X, n, d, h	n-by-d input matrix, positive batch size, input width, and hidden width, respectively.
W_1, b_1	d-by-h hidden weights and length-h hidden bias column, respectively.
Z, H	n-by-h preactivation and elementwise tanh activation matrices.
$W_2, b_2, \widehat{y}$	h-by-one output weights, scalar output bias, and n-by-one prediction column.
$\tanh$	Hyperbolic tangent applied independently to every preactivation entry.
L, y	Mean squared-error scalar and n-by-one target column.
G_2	n-by-one loss derivative with respect to predictions, already containing the mean-loss normalization.
$dL, d\widehat{y}$	First-order loss and prediction differentials; d here is not the input feature count.

$\nabla_{W_2} L$	h-by-one column of output-weight derivatives.
$\partial L / \partial b_2$	Scalar output-bias derivative, summing contributions over the batch.
$G_1, \odot$	n-by-h hidden-preactivation loss gradient and multiplication of corresponding array entries, respectively.
1	In the entrywise subtraction, scalar one is broadcast to every position of the n-by-h hidden array.
$\nabla_{W_1} L$	d-by-h hidden-weight gradient, matching the hidden-weight matrix.
$\nabla_{b_1} L$	Length-h column of hidden-bias derivatives.

Assumptions. Use the chapter's row-batch two-layer tanh regressor with scalar outputs. Targets and predictions are matching columns. Inputs are fixed, and the objective is mean squared error without an additional weight penalty.

The loss differential records how every small parameter change contributes to one scalar loss. Collect each parameter's coefficient to obtain its derivative. A shared bias affects every example, so its contributions add. If the incoming loss derivative already contains the batch-mean factor, averaging these contributions again would incorrectly divide every bias gradient a second time.

☐ Derivation *Advanced*

Use exactly the forward computation introduced in Section 12.1, with tanh applied to each hidden preactivation. Its three stages are recorded separately so each backward derivative can be matched to a forward operation.

$$Z = X W_1 + \mathbf{1}_n b_1^T \tag{12.T4}$$

$$H = \tanh(Z) \tag{12.T5}$$

$$\widehat{y} = H W_2 + \mathbf{1}_n b_2 \tag{12.T6}$$

$$L = \frac{1}{n} \|\widehat{y} - y\|_2^2 \tag{12.T7}$$

$$G_2 = \frac{2}{n} (\widehat{y} - y) \tag{12.T8}$$

$$dL = G_2^T \, d\widehat{y} \tag{12.T9}$$

A small change in the output can come from changing hidden activations, output weights or the shared bias. Apply the product rule to separate these

contributions, then substitute them into the scalar loss differential. To read off an output-weight derivative, collect the coefficient of that weight's change: its hidden feature times incoming error, summed over examples. The bias similarly collects all incoming errors because it is shared.

$$\nabla_{W_2} L = H^T G_2 \tag{12.T10}$$

$$\frac{\partial L}{\partial b_2} = \mathbf{1}_n^T G_2 \tag{12.T11}$$

An incoming prediction gradient must reach every hidden unit through that unit's output weight. Across a batch, those contributions form the outer product of the prediction-gradient column and output weights. Multiply entrywise by the tanh derivative to obtain preactivation gradients. Multiplying by the input transpose then forms each input-times-error contribution and sums it across examples, producing the hidden-weight gradient.

$$G_1 = (G_2 W_2^T) \odot (1 - H \odot H) \tag{12.T12}$$

$$\nabla_{W_1} L = X^T G_1 \tag{12.T13}$$

$$\nabla_{b_1} L = G_1^T \mathbf{1}_n \tag{12.T14}$$

These formulas include every path through the shared parameters. Their shapes match the original parameters, and the only averaging factor entered at the loss derivative. Repeating this local coefficient-collection operation backward through further layers is reverse-mode differentiation, not a new learning assumption.

□

Backpropagation differentiates the coded objective. Shape checks, manual cases, and numerical derivative checks are still needed to verify that this objective is the intended one.

▶ Example: The same forward pass, now backward

Using unrounded values from Section 12.1, the squared loss is approximately 1.651 and its prediction derivative is approximately 2.570. Follow the graph backward. Multiplication by the hidden value gives output-weight derivative approximately 1.957; the output-bias derivative remains approximately 2.570. The hidden preactivation receives the prediction derivative times output

weight 3 times the tanh slope, giving approximately 3.237. This is also the hidden-bias derivative. Multiplying by input 2 gives hidden-weight derivative approximately 6.475. Each multiplication reverses one forward operation; no parameter update occurs during this derivative calculation.

Interpretation and limits

The shared bias receives a sum of per-example contributions. Dividing that sum by batch size after forming a mean-loss derivative would average twice.

Algorithm 12.1. One tanh-network gradient step

Source and adaptation: Rumelhart, Hinton and Williams [12.3], and Goodfellow, Bengio and Courville [12.4], Sections 6.5-6.5.7. This teaching specialization uses a tanh hidden layer, scalar mean-squared error, explicit shared-bias sums and simultaneous saved-gradient updates; it is not a verbatim reproduction of a source algorithm.

Notation and assumptions

$X \in \mathbb{R}^{n \times d},\ y \in \mathbb{R}^{n \times 1}$	Aligned batch inputs and scalar targets; positive batch size n and input width d.
$W_1 \in \mathbb{R}^{d \times h},\ b_1 \in \mathbb{R}^{h}$	Hidden weights and bias column; positive hidden width h.
$W_2 \in \mathbb{R}^{h \times 1},\ b_2 \in \mathbb{R}$	Output weights and scalar bias; theta denotes all four parameters; p is one parameter array.
$Z, H \in \mathbb{R}^{n \times h}$	Hidden preactivations and entrywise tanh activations.
$\widehat{y}, E, G_2 \in \mathbb{R}^{n \times 1}$	Predictions, residuals and output-loss gradient.
$G_1 \in \mathbb{R}^{n \times h},\ g_p$	Hidden-preactivation gradient; saved gradient array for parameter p.
$\mathbf{1}_n, \mathbf{1}_{n \times h}, \odot$	All-ones column, all-ones matrix and entrywise multiplication.
L, η	Pre-update mean squared error and positive finite learning rate.

Assert all input and parameter shapes before computation. All gradients are evaluated at the same unchanged parameter state. Save gradients in parameter shapes, with a scalar output-bias derivative. Reject nonfinite loss, gradients or proposed parameters. Run the update on a copy and discard it on failure.

Algorithm 12.1

Input: Finite aligned arrays and parameters with the stated shapes. X, y, θ

Positive learning rate. $\eta > 0$

Output: Updated parameter collection and pre-update loss. θ, L

Forward $Z \leftarrow X W_1 + \mathbf{1}_n b_1^T$

Activate $H \leftarrow \tanh(Z)$

$\widehat{y} \leftarrow H W_2 + \mathbf{1}_n b_2$

Assert $\operatorname{shape}(\widehat{y}) = \operatorname{shape}(y) = (n, 1)$

Residual $E \leftarrow \widehat{y} - y$

Loss $L \leftarrow \|E\|_2^2 / n$

Backward $G_2 \leftarrow 2E/n$

Save $g_{W_2} \leftarrow H^T G_2$

Save $g_{b_2} \leftarrow \mathbf{1}_n^T G_2$

Backward $G_1 \leftarrow (G_2 W_2^T) \odot (\mathbf{1}_{n \times h} - H \odot H)$

Save $g_{W_1} \leftarrow X^T G_1$

Save $g_{b_1} \leftarrow G_1^T \mathbf{1}_n$

For $p \in \{W_1, b_1, W_2, b_2\}$

    $p \leftarrow p - \eta g_p$

End for

Return θ, L

Why this works. Bias gradients sum over observations; do not divide by n again after G2. The hidden signal uses the old W2. The returned loss belongs to the forward pass before any update.

This is one gradient step, not an optimization convergence claim. Compare every saved gradient against deterministic central differences before scaling up. Shape equality prevents an n-by-one prediction from silently broadcasting against an n-vector target.

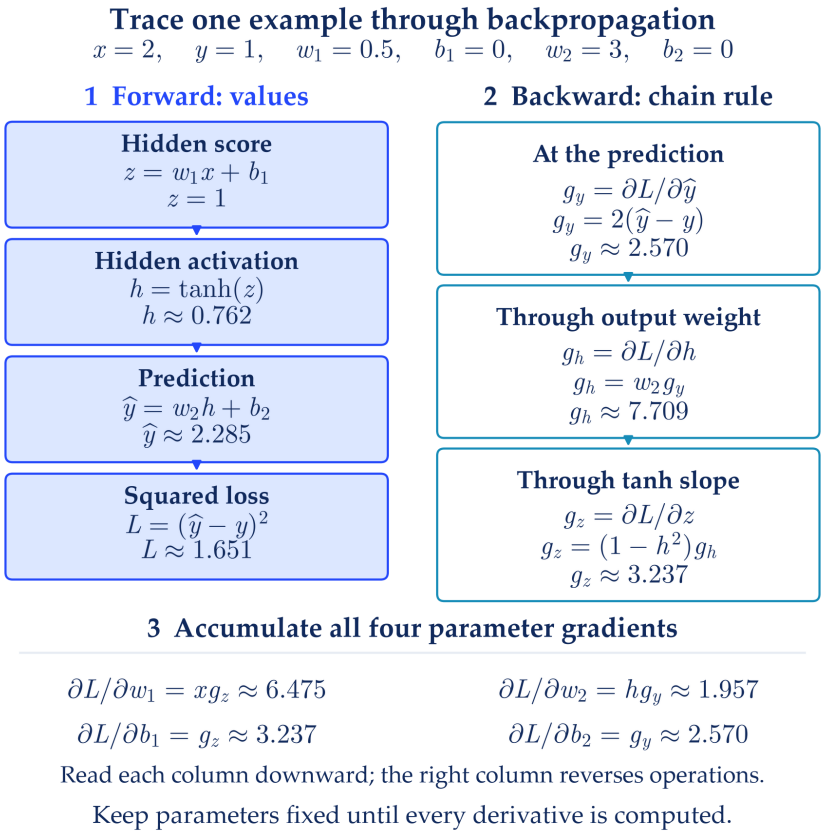


Figure 12.4. Read downward on the left for forward values, then downward on the right for loss derivatives with respect to the prediction, hidden activation and preactivation. The bottom collects both weight and both bias derivatives. This is the one-example regressor from Sections 12.1 and 12.3, calculated without rounding intermediates. For batches, use the matrix derivation: average once at the loss, then sum shared-parameter contributions.

Essential

12.4 Checking derivatives independently

Check gradients against changes in the actual computed loss. Fix the data and all nonparameter state, perturb one parameter equally in both directions, and compare the central difference with its analytic derivative. This independent numerical check can catch errors shared by hand derivation and code. It verifies the implemented scalar objective, not whether that objective matches the intended learning problem.

The perturbation cannot simply be made arbitrarily small. A large step introduces truncation error, whereas a very small step subtracts nearly equal floating-point values and amplifies evaluation error. Use double precision and inspect several steps, for example from one ten-thousandth to one millionth in a well-scaled tiny model. The accompanying result quantifies these two competing effects. For a ReLU model, avoid checking exactly at a corner, where an ordinary derivative may not exist.

A coordinate check costs two forward evaluations per parameter. A directional check costs two for the whole parameter vector: perturb along both signs of a fixed unit-length direction and compare the central difference with the gradient's inner product with that direction. Use absolute and relative tolerances near zero, and restore every parameter before another check. One passing direction is a useful diagnostic, not verification of every gradient coordinate.

Finite differences test whether the derivative matches the implemented forward loss, not whether that loss represents the intended task. Even the accidental all-pairs loss can pass this test. Combine derivative agreement with the shape assertions and hand-calculated forward values from Section 12.1. A small negative-gradient step is an optimization smoke test only after both the intended objective and its derivative have been checked.

Griewank and Walther [12.6] distinguish algorithmic differentiation from finite-difference approximations. Comparing the two provides an independent check of an implementation, provided that step-size error, floating-point cancellation, and nondifferentiable points are not mistaken for proof of an incorrect derivative.

Notation for equation (12.14)

L	Deterministic scalar loss with fixed data and nonparameter state.
θ	Length-P column containing every parameter in a fixed flattened order.
θ_j	Parameter coordinate selected for checking; j ranges from one through P.

e_j	Length-P coordinate basis vector with one at position j and zero elsewhere.
ε	Positive perturbation step.
P	Total number of scalar trainable parameters.

$$\frac{\partial L}{\partial \theta_j} \approx \frac{L(\theta + \varepsilon e_j) - L(\theta - \varepsilon e_j)}{2\varepsilon} \tag{12.14}$$

Perturb one coordinate of the flattened parameter vector while holding every other coordinate and the data fixed.

Notation for equation (12.15)

L	Deterministic scalar objective used at both perturbed parameter states.
θ	Length-P parameter column at which the gradient is checked.
$\nabla L(\theta)$	Length-P column of loss derivatives at the unperturbed parameters.
v	Fixed length-P unit direction for the perturbation.
ε	Positive step along the chosen direction.
P	Number of trainable scalar parameters.

$$\nabla L(\theta)^T v \approx \frac{L(\theta + \varepsilon v) - L(\theta - \varepsilon v)}{2\varepsilon} \tag{12.15}$$

A fixed direction checks a linear combination of parameter derivatives using only two loss evaluations.

▶ Example: Check the hidden weight, not a new model

Perturb only the hidden weight around 0.5, once upward and once downward by 10^{-5}. Subtract the two losses and divide by twice the step: the derivative is approximately 6.475. Restore the weight, then repeat for the other three parameters. Next, keep the numerical objective as a four-example mean but deliberately omit the mean factor in the analytic gradient. Analytic values become four times the finite differences, diagnosing inconsistent loss reduction. Check both perturbation signs and the restored state before blaming numerical precision.

Interpretation and limits

A stochastic forward pass cannot be compared by independently resampling noise at each perturbation. Freeze the evaluated computation before interpreting derivative discrepancies.

Algorithm 12.2. Central-difference gradient audit

Source and adaptation: Griewank and Walther [12.6], and the real-input central-difference construction in PyTorch Gradcheck mechanics [12.12]. The step-size sweep, nonfinite failure flags and exact state-restoration checks form the book's diagnostic protocol, not a proof that the task objective is correct.

Notation and assumptions

$\theta \in \mathbb{R}^P, \theta_0, e_j$	Flattened parameter column, independent saved original, and coordinate basis vector; j ranges from one through P.
L, g	Deterministic scalar loss and analytic gradient column at the original state.
$\mathcal{H}, \varepsilon$	Finite nonempty list of positive perturbation sizes and its current member.
τ_a, τ_r	Nonnegative absolute and relative comparison tolerances.
L_+, L_-, d, Δ	High/low losses, central-difference derivative, and absolute discrepancy.
T, b	Per-coordinate/per-step report table and Boolean tolerance outcome.

Use double precision at a differentiable point. Freeze data, random masks and all nonparameter state. Each loss call receives a separate copy of θ_0 and cannot mutate it. Store records steps and discrepancies. Flag nonfinite values or collapsed steps. Use representable symmetric steps.

Algorithm 12.2

Input: Loss, analytic derivative, fixed batch and unchanged parameter state.

L, θ

Positive step list and nonnegative tolerances. $\mathcal{H}, \tau_a, \tau_r$

Output: Discrepancy/flag table and exactly preserved original parameters.

T, θ_0

01 **Copy** $\theta_0 \leftarrow \mathrm{Copy}(\theta)$

02 $g \leftarrow \nabla L(\theta_0)$

03 **Assert** $\mathrm{shape}(g) = \mathrm{shape}(\theta_0)$

04 $T \leftarrow [\,]$

Algorithm 12.2 (continued)

For $\varepsilon \in \mathcal{H}$
  For $j = 1, \ldots, P$
    Independent copy. $L_+ \leftarrow L(\theta_0 + \varepsilon e_j)$
    Independent copy. $L_- \leftarrow L(\theta_0 - \varepsilon e_j)$
    Difference $d \leftarrow (L_+ - L_-)/(2\varepsilon)$
    Discrepancy $\Delta \leftarrow |d - g_j|$
    Compare Require finite inputs.
    $b \leftarrow [\Delta \leq \tau_a + \tau_r \max\{|d|, |g_j|\}]$
    $T \leftarrow \mathrm{Store}(T, j, \varepsilon, d, \Delta, b)$
  End for
End for
Restore $\theta \leftarrow \mathrm{Copy}(\theta_0)$
Verify Exact state equality. $\theta = \theta_0$
Return T, θ_0

Why this works. Unflatten using the original parameter shapes when the loss expects arrays. Never perturb a cumulatively modified state. Report agreement over useful step sizes, not just one favorable epsilon: truncation and roundoff compete.

The finite loops make exactly 2P times the number of step sizes loss calls. A passing audit checks derivative consistency for the implemented deterministic loss, not the correctness of the task or generalization.

Automatic differentiation supports the same calculation

An automatic differentiation library applies local derivative rules to the program that actually ran. It can return the derivatives of the coded scalar loss with respect to all requested parameters. It does not decide whether that loss, target alignment or information boundary is appropriate. Compare its results with the hand calculation on the same small deterministic network before relying on a longer training run.

The comparison must use the same parameter values, data, loss reduction and numerical precision. Recompute the forward pass after changing parameters. Treat the library result as another implementation to audit: agreement is evidence of computational consistency.

Essential

12.5 From a correct gradient to a trained predictor

The forward and backward calculations specify one model and one objective. Training repeats that calculation at changing parameters. The aim is to choose a predictor that works on new examples, using the validation principles already developed in Chapter 4. Begin with a small network, plain stochastic gradient descent and a short run whose behavior can be inspected. A more elaborate optimizer cannot establish that the inputs, targets and loss have been paired correctly.

Keep the objective and update convention explicit

Use the preceding mean squared-error convention for the running regressor: average the squared residuals, without a factor of one half. For a nonempty mini-batch, compute every example derivative at the same current parameters, average those derivatives, and subtract the learning rate times that mean from each parameter. The number of batches in an epoch depends on batch size and on whether a short final batch is retained. A batch of one gives a stochastic update; using all training examples gives the full gradient.

▶ Example: Finish the running network's first update

Keep the example from the preceding sections: input 2, target 1, hidden weight 0.5, output weight 3 and zero biases. Its prediction is about 2.284782 and its squared loss is 1.650666. Backpropagation gives the four derivatives below. Apply a learning rate of 0.01 to all four parameters, using the same unchanged forward and backward pass.

Parameter	Before	Derivative	After one update
Hidden weight	0.500000	6.474908	0.435251
Hidden bias	0.000000	3.237454	-0.032375
Output weight	3.000000	1.956966	2.980430
Output bias	0.000000	2.569565	-0.025696

Recompute the network with the unrounded updated parameters. The prediction becomes about 2.015350 and loss becomes 1.030936. This one controlled decrease checks the direction and scale of an update. It does not guarantee that every mini-batch step decreases full-data loss. Updating the hidden weight before calculating the other derivatives would mix two

parameter states and would no longer execute the stated simultaneous gradient step.

A complete small training procedure

1. Fix the data boundary. Reserve test examples and use nonempty training and validation partitions within development data. Fit scaling on training inputs and reuse that fitted transformation. Decide the metric, learning-rate candidates, positive finite epoch budget and checkpoint rule before viewing test results.

2. Initialize one reproducible model. Set the random seed before creating parameters and the batch order. Break hidden-unit symmetry at initialization so updates can differ. Use an initializer suited to the activation and record its settings. A seed supports a controlled comparison; it is not a guarantee of identical results across every software and hardware configuration.

3. Execute an epoch. Require at least one nonempty training batch in every executed epoch. For independent examples, use the declared seeded shuffle; preserve any grouping or time constraints. For each batch, calculate predictions and mean loss, compute its gradients, and update the parameters. Check prediction/target shapes and finite loss, gradients and parameters. A missing or nonfinite value, or an empty epoch, stops the run without test evaluation.

4. Measure the current model. Evaluate training and validation loss at the same fixed epoch-end parameters. Use deterministic inference behavior and do not perform parameter updates during evaluation. Reject nonfinite required evaluation losses. This separates the model being evaluated from the changing sequence of models encountered during training.

5. Save an independent best state. When the declared validation criterion improves sufficiently, copy the complete selected predictor state and record the epoch. A variable that merely points to the current parameters will continue changing and is not a saved checkpoint.

6. Apply the stopping rule. Stop at the epoch budget or after the declared number of evaluations without accepted improvement. Restore the saved state, rather than using whichever state happened to execute last. If no valid state was saved, report failure and leave the test set untouched.

7. Evaluate once and retain the predictor. Compare the frozen choice with the declared simple baseline on the held-out test set. Retain preprocessing, feature order, architecture, parameters and the output interpretation needed for future predictions. Repeatedly changing the model after seeing test results turns that set into development data.

Procedure sources: Goodfellow, Bengio and Courville (2016), Chapters 8 and 11, for mini-batch optimization and practical methodology; Prechelt (1998) for validation-based early stopping. Both sources are listed in this chapter's references.

▶ Example: A short batch must not receive a full batch's weight

Suppose three batches contain 2, 2 and 1 examples, with mean losses 0.4, 0.5 and 0.9. Their loss totals are 0.8, 1.0 and 0.9. The example-weighted mean is 2.7/5 = 0.54; the unweighted mean of the three batch means is 0.60. If parameters were updated between these batches, 0.54 describes losses along the training trajectory. To report the training loss of the selected predictor, run those examples again without updates at its fixed parameters.

Choose an update scale before increasing the model

Standardized inputs and a reasonable initializer make a learning-rate check easier to interpret. An update that is large relative to curvature can overshoot even when its gradient is correct; Chapter 2 establishes this mechanism on a quadratic. Inspect several plausible rates with the same partitions and starting state. A decreasing training loss combined with a worsening validation loss suggests a different problem from a loss that diverges immediately.

▶ Example: Variance and standard deviation are different inputs

If a selected normal initializer prescribes weight variance 0.04, its standard deviation is the square root, 0.2. Passing 0.04 to an interface that expects standard deviation produces variance 0.0016. State which quantity an interface accepts, and inspect actual activation and gradient magnitudes on a small batch. The correct conversion does not establish that this particular initializer is suitable for every network.

▶ Example: Select the earlier state

Suppose validation losses over six epochs are 0.52, 0.44, 0.40, 0.41, 0.43 and 0.45. With strict improvement, zero minimum-improvement margin and patience two, epoch 3 is selected. Epochs 4 and 5 are two consecutive non-improvements, so training stops after epoch 5 and restores epoch 3. Epoch 6 would not be executed under this rule. Smaller training loss at epoch 5 does not override the stated validation criterion.

Use a short diagnostic sequence

First inspect the data and the loss definition. Confirm feature order, target alignment, class mapping and the mean-versus-sum convention. Next test one deterministic forward pass and compare a few analytic derivatives with finite differences. Then try to fit a tiny, well-defined training subset. A small network's inability to fit a suitable tiny subset can expose an implementation or optimization problem, although conflicting labels or insufficient model capacity are genuine alternatives.

If training loss remains high, inspect the update scale, input scale, activation saturation and model capacity. If training loss falls while validation loss rises, compare a smaller model, earlier stopping or the regularization already introduced in Chapter 2. If both curves look implausibly good, audit duplicates, grouping and preprocessing boundaries. A curve is evidence to investigate, not a unique diagnosis.

Choose the output and loss together

The running regressor has an unrestricted scalar output and squared loss. A threshold can turn its score into a class decision, but does not turn it into a calibrated probability. For probabilistic binary classification, use a declared binary logit loss; for multiclass classification, use one logit per class with the corresponding cross-entropy convention from Chapter 3. Keep the output definition fixed while checking gradients and training. The MNIST case deliberately retains squared loss and reports the limitations of its thresholded regression score.

Table 12.1. Activation functions have different local behavior

Activation	Output and derivative behavior	Practical consequence
Sigmoid	Output between 0 and 1; derivative at most 1/4 and small at extremes.	Useful for binary probability output; saturated hidden units transmit small gradients.
Tanh	Output between -1 and 1; derivative 1 minus output squared.	Zero-centered hidden output, but saturation still occurs.

Activation	Output and derivative behavior	Practical consequence
ReLU	Zero on negative inputs and linear on positive inputs.	Simple active-region gradient; negative preactivations have zero derivative.
Affine only	Composing affine layers gives another affine function.	Depth alone does not produce nonlinear decision boundaries.

Table 12.2. Chapter synthesis

Concept	Main conclusion	Engineering check
Learned features	Hidden layers learn representations jointly with the output mapping.	Trace one input from preactivation through hidden values to prediction.
Shapes	Array dimensions encode example alignment and parameter sharing.	Assert prediction and target shapes before loss evaluation.
Backward pass	Reverse chain-rule calculations give the gradient of the implemented loss.	Include the loss-reduction factor exactly once.
Verification	Finite differences and automatic differentiation check the handwritten gradient.	Use deterministic double precision and avoid nondifferentiable kink points.

Dataset: MNIST handwritten digits. See its first-use portrait [12.10].

Case Study: Verify a handwritten network on actual MNIST threes and eights

Problem and data

MNIST supplies a recognizable real-image setting for the chapter's explicit network mathematics. The task is intentionally restricted to threes versus eights so that one unrestricted scalar output and mean squared loss retain the original derivative convention. Before asking whether nonlinear features help, verify that the same real-image batch produces consistent handwritten and numerical derivatives. Preserve original train and test origins and do not confuse a thresholded regression score with a calibrated digit probability.

Experimental design

Select fixed stratified subsets and center normalized pixels using fitting images only. Implement a 784-input, twelve-hidden-unit tanh network and its full backward pass. Check three declared unit parameter directions with central differences.
Train for a bounded mini-batch schedule,
copying the lowest-validation-loss state. Compare that state with a ridge affine control, freeze the lower-validation-MSE choice, and evaluate the original-test subset once at a fixed threshold.

Measured Reference Run

The two-digit task used 1,600 fitting and 400 validation images from original training, plus 600 original-test images. A twelve-hidden-unit tanh network trained for fifty epochs with batch size 128 and learning rate 0.03. Validation MSE selected its saved epoch 49 over an affine ridge control. The fixed 0.5 threshold converted unrestricted regression scores to classes.

Table 12.3. Measured case-study evidence

Final procedure	MSE	Accuracy	Test count
Tanh network	0.056	0.953	600
Fitting-mean reference	0.25	0.508	600

Validation MSE selected the network, 0.050 versus ridge's 0.069. Separately, three directional finite differences agreed closely with its handwritten

gradients. Prediction and derivative correctness are distinct checks: inspected-batch agreement neither establishes predictive superiority nor verifies every program state.

Failure Analysis And Limits

Final scores ranged from -0.466 to 1.373. Despite useful thresholded accuracy, these outputs cannot be interpreted directly as probabilities. The result also concerns only threes versus eights, not all digit classes. Selecting a network state on validation and checking its gradients do not establish calibration or robustness to new handwriting conditions.

Historical CPU saved run; preserve source-image subsets, fitting-only centering, initialization, batch order and copied checkpoint. The core audit uses handwritten forward and backward calculations and three independent directional finite differences.

▶ Example: inspecting the real data

A 28-by-28 MNIST image supplies 784 inputs. The 784-to-12-to-1 network has 784 times 12 = 9,408 hidden weights, 12 hidden biases, 12 output weights and one output bias: 9,433 trainable scalars. Each parameter has a gradient entry. Three directional checks compare projections of this gradient with finite differences.

Dataset source and documentation

Companion: Week 12, Real-Data Case Study notebook. Reproduce the declared run, then investigate a changed condition. No fixed benchmark score is promised.

A Controlled Learning Cycle

The MNIST case connects a declared objective and parameter shapes to classification evidence. Week 12 notebooks compare handwritten derivatives with finite differences and complete a small NumPy training and validation cycle. The Iris laboratory exposes forward and backward traces with linear, tanh or ReLU activations. Record one controlled change with its metric, data roles and explanation.

Matched practice and simulator protocol · Packaged edition; launch instructions in the guide.

Laboratory 12 | Neural Networks: Foundations and Practice

Actual forward/backward traces and linear, tanh or ReLU activations.

Browser laboratory dataset: Iris. See its first-use portrait [12.11].

A trained Iris network exposes an observation's forward pass and backpropagated derivatives through its hidden layer.

Open Laboratory 12 | Use the live view for readable controls.

Transfer: MNIST threes and eights → Three-class, four-feature Iris classification. No autograd or MNIST reproduction; derivative correctness does not establish generalization.

Chapter Capability

Implement a small neural network, verify its handwritten derivatives numerically, complete a bounded training cycle and evaluate the validation-selected predictor against a simple baseline.

Week 12: separate practice, solutions and assessment

References

[12.1] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review 65(6), 386-408. Source / DOI

[12.2] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals and Systems 2, 303-314. Source / DOI

[12.3] Rumelhart, D. E.; Hinton, G. E.; Williams, R. J. (1986). Learning representations by back-propagating errors. Nature 323, 533-536. Source / DOI

[12.4] Goodfellow, I.; Bengio, Y.; Courville, A. (2016). Deep Learning. MIT Press. Source / DOI

[12.5] Baydin, A. G.; Pearlmutter, B. A.; Radul, A. A.; Siskind, J. M. (2018). Automatic Differentiation in Machine Learning: a Survey. Journal of Machine Learning Research 18(153), 1-43. Source / DOI

[12.6] Griewank, A.; Walther, A. (2008). Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation. 2nd ed., SIAM. Source / DOI

[12.7] Glorot, X.; Bordes, A.; Bengio, Y. (2011). Deep Sparse Rectifier Neural Networks. Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, PMLR 15, 315-323. Source / DOI

[12.8] Leshno, M.; Lin, V. Y.; Pinkus, A.; Schocken, S. (1993). Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. Neural Networks 6(6), 861-867. Source / DOI

[12.9] Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Kopf, A.; Yang, E.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. Advances in Neural Information Processing Systems 32. Source / DOI

[12.10] LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324. Source / DOI

[12.11] Fisher, R. A. (1936). Iris. UCI Machine Learning Repository. Source / DOI

[12.12] PyTorch contributors (2026). Gradcheck mechanics. Official PyTorch developer documentation, real-input numerical evaluation. Accessed 30 October 2026. Source / DOI

[12.13] PyTorch contributors (2026). CrossEntropyLoss. Official PyTorch API documentation, input logits, class-index targets and reduction conventions. Accessed 30 October 2026. Source / DOI

[12.14] Prechelt, L. (1998). Early Stopping - But When?. Neural Networks: Tricks of the Trade, Lecture Notes in Computer Science 1524, 55-69, Springer. Source / DOI

Conceptual Glossary

Definitions summarize the main meaning in this book. Consult the linked section for assumptions, notation and examples.

Active learning. Choosing which unlabeled observation to label next to improve a predictor under a stated acquisition budget. It is not the same objective as finding a maximum. 10.4

Algorithm. A finite, explicit procedure with inputs, update rules, stopping conditions and outputs. A model family does not by itself specify how its parameters are fitted. 2.3; 8.1

Anomaly detection. Scoring departures from a chosen reference distribution or structure. An unusual observation is not automatically an error or a harmful event. 8.5

Area under the ROC curve (ROC AUC). A ranking measure based on the ROC curve. With the usual half-credit treatment of ties, it estimates how often a positive example scores above a negative one. 3.3

Automatic differentiation. Computing derivatives by applying the chain rule to the operations used in a numerical program. It differs from symbolic algebra and finite-difference approximation. 12.4

Bagging. Fitting learners to resampled training data and aggregating their predictions. Its variance reduction depends on the individual learners and the dependence between their errors. 7.3

Baseline. A deliberately simple reference procedure evaluated under the same data and metric rules as the proposed model. It makes the value of added complexity measurable. 1.3; 11.3

Batch. A set of training observations used together to compute an update. The aggregation convention for the loss determines the scaling of its gradient. 12.5

Bayesian optimization. Sequential selection of evaluations to optimize an unknown objective, typically balancing predicted value and uncertainty through an acquisition function. 10.5

Bias-variance decomposition. For squared-error loss with finite second moments and an independent fresh outcome conditional on the input, this separates squared bias, variation across training samples and outcome noise. 1.5

Calibration. Agreement between predicted probabilities and observed outcome frequencies under the relevant population. Strong ranking alone does not imply good calibration. 9.1; 9.2

Checkpoint. A saved computational state. Weights can support inference with the correct architecture and preprocessing; exact training resumption also requires relevant optimizer, random-generator and data-order state. 12.5

Class imbalance. Unequal frequencies of target classes. Appropriate evaluation depends on the population and error costs, not on forcing the final test set to be balanced. 3.5

Clustering. Grouping observations according to a specified geometry or similarity criterion without training on target class labels. Different definitions can produce different valid partitions. 8.1; 8.3; 8.4

Conformal prediction. Constructing prediction sets using a calibrated score and a finite-sample rank rule. Standard marginal coverage depends on exchangeability; it is not a per-case certainty claim. 9.5

Confusion matrix. Counts of predicted versus observed classes at a fixed decision rule. These counts depend on the threshold and the evaluated population. 3.3

Cross-validation. Repeatedly fitting a procedure on training folds and evaluating it on held-out folds. Every learned preprocessing step must respect each fold boundary. 4.1; 4.3

Data leakage. Using information in fitting or selection that would not be available under the intended evaluation or prediction contract. Leakage can affect preprocessing as well as labels. 1.3; 4.3

DBSCAN. A density-based clustering method built from neighborhoods and core-point connectivity. It can mark observations as noise and need not assign every point to a cluster. 8.4

Decision threshold. A rule that converts a score or probability into a decision. Its selection should reflect stated costs or constraints using development evidence, not repeated final-test tuning. 3.4

Dimensionality. The number of coordinates in a representation. Increasing it changes distance geometry and data requirements; it is not a direct measure of independent information. 1.4; 5.1

Embedding. A vector representation intended to make selected relationships useful for a downstream task. A pretrained embedding is not automatically appropriate for every domain. 11.4

Empirical risk. Average loss on a specified finite sample. It estimates a population quantity only under suitable sampling and evaluation assumptions. 1.2

Ensemble. A predictor formed by combining multiple fitted predictors. Bagging, boosting, voting and stacking use different fitting and aggregation rules. 7.3; 7.4; 7.5

Exchangeability. Invariance of the joint distribution under permutations of observation order. Standard conformal guarantees require an appropriate exchangeability assumption. 9.5

Feature. An input coordinate or derived quantity made available to a predictor. Its definition includes how and when it is obtained, not only its column name. 1.1; 2.1

Finite difference. A numerical derivative approximation from nearby function evaluations. Step-size, floating-point error and nondifferentiable points limit its use as a gradient check. 12.4

Gaussian process. A distribution over functions for which every finite collection of function values has a joint Gaussian distribution. Mean, covariance and observation-noise assumptions determine predictions. 10.1; 10.3

Generalization. Performance on observations beyond those used for fitting and selection, relative to a declared population and loss. Low training error alone is insufficient evidence. 1.2; 1.3

Gradient boosting. Building an additive predictor by sequentially fitting corrections guided by the chosen loss. Learning rate, learner capacity and stopping affect the result. 7.4

Gradient descent. Updating parameters in the negative gradient direction. Step size and objective assumptions determine whether a mathematical decrease guarantee applies. 2.3

Hierarchical clustering. Constructing nested groups using a linkage or merge criterion. Different linkages encode different notions of between-cluster proximity. 8.3

Hyperparameter. A choice defining a model family or fitting procedure that is not fitted by the same inner optimization as its parameters. Its selection belongs within the evaluation protocol. 4.4

Inductive bias. Assumptions or preferences that make some fitted explanations easier to select than others. No learning method avoids such preferences entirely. 1.5

K-means. Alternating assignment and centroid updates to reduce within-cluster squared distance. Initialization matters, and the algorithm does not generally find the global optimum. 8.1

K-nearest neighbors (KNN). Predicting from a chosen number of nearby training observations. Scaling, the distance function and the neighborhood size all influence what counts as nearby. 6.1

Kernel. A function used to express pairwise relationships. For standard kernel methods, positive semidefiniteness enables an inner-product interpretation on finite collections. 6.4; 10.2

Likelihood. A probability or density for the observed data viewed as a function of model parameters under a specified probabilistic model. 2.2; 3.2

Logistic regression. A linear score transformed by the logistic function to model a binary conditional probability. The probability model and the eventual decision threshold are distinct. 3.1; 3.2

Loss function. A numerical penalty for a prediction and an observed outcome. Its aggregation and scale define what an optimization procedure is asked to improve. 1.2; 3.2

Margin. A separation quantity between a decision boundary and observations, under a specified normalization. Soft-margin methods trade margin size against constraint violations. 6.2

Naive Bayes. A probabilistic classifier using conditional factorization assumptions for the inputs. Its classification usefulness does not imply that its independence assumptions are literally true. 11.3

Nested cross-validation. An outer evaluation loop enclosing the entire inner model-selection procedure. It estimates the behavior of selection, not just one fixed parameter setting. 4.4

Neural network. A composition of parameterized transformations, commonly with nonlinear activations, fitted to an objective. Hidden representations extend the expressiveness of a linear predictor. 12.1; 12.2

Overfitting. Fitting sample-specific variation that does not improve the intended out-of-sample objective. It can arise in parameter fitting or through repeated model selection. 1.5; 4.4

Partial least squares (PLS). A supervised latent-component method that uses the response while constructing directions. Component selection and preprocessing must remain inside the training boundary. 5.4

Pipeline. A composed procedure including preprocessing and prediction. Evaluating only the final estimator while preprocessing all data first can leak held-out information. 2.1; 4.3

Population risk. Expected loss under the specified data-generating population. This is a property of the predictor and distribution, not simply its loss on one observed dataset. 1.2

Precision and recall. At a fixed decision rule, precision concerns the fraction of positive predictions that are positive; recall concerns the fraction of positive cases detected. 3.3

Principal component analysis (PCA). An orthogonal representation of centered data chosen to maximize retained variance, equivalently minimize squared reconstruction error at a fixed rank. 5.2; 5.3

Proper scoring rule. A probability-evaluation rule whose expected score is optimized by reporting the true probability distribution. It addresses probabilities, not just hard decisions. 9.1

Random forest. A bagged tree ensemble that also randomizes candidate features during splitting. The extra randomness can reduce dependence between trees. 7.3

Regularization. A constraint, penalty or fitting choice that restricts effective flexibility. A regularization parameter is meaningful only with its loss-scaling convention. 2.4

ROC curve. The trade-off between true-positive rate and false-positive rate as a score threshold varies. Its visual shape does not specify the operational cost of false positives. 3.3

Support vector. A training observation associated with the fitted margin constraints through the SVM dual solution. Free and bound support vectors satisfy different KKT cases. 6.3

Support-vector machine (SVM). A margin-based method with regularization and, for soft margins, penalized constraint violations. Kernels allow nonlinear decision boundaries without explicitly constructing all features. 6.2; 6.4

Support-vector regression (SVR). A regression method with an epsilon-insensitive loss region and regularized function complexity. The tolerance and target scale must be interpreted together. 6.5

t-SNE. A nonlinear visualization that emphasizes selected neighborhood relationships. It is not a general proof of clusters, global distances, or predictive usefulness. 5.5

Test lock. The decision to freeze the fitting and selection procedure before accessing final-test results. A repeatedly optimized test set has become development data. 4.5

TF-IDF. A lexical representation combining within-document term frequency with a training-corpus inverse-document-frequency factor. The vocabulary and weighting are fitted quantities. 11.2

Validation set. Data reserved for choices such as model settings, calibration methods or thresholds. Repeated use affects how independently it can support a performance claim. 1.3; 4.4

Subject Index

Page numbers lead to the substantive explanation, not every incidental mention. All section references are clickable.

Active learning - 10.4, p. 377
Algorithm - 2.3, p. 70; 8.1, p. 301
Anomaly detection - 8.5, p. 320
Area under the ROC curve (ROC AUC) - 3.3, p. 114
Automatic differentiation - 12.4, p. 450
Bagging - 7.3, p. 278
Baseline - 1.3, p. 22; 11.3, p. 413
Batch - 12.5, p. 454
Bayesian optimization - 10.5, p. 383
Bias-variance decomposition - 1.5, p. 28
Calibration - 9.1, p. 331; 9.2, p. 334
Checkpoint - 12.5, p. 454
Class imbalance - 3.5, p. 134
Clustering - 8.1, p. 301; 8.3, p. 311; 8.4, p. 314
Conformal prediction - 9.5, p. 349
Confusion matrix - 3.3, p. 114
Cross-validation - 4.1, p. 153; 4.3, p. 160
Data leakage - 1.3, p. 22; 4.3, p. 160
DBSCAN - 8.4, p. 314
Decision threshold - 3.4, p. 128
Dimensionality - 1.4, p. 26; 5.1, p. 189
Embedding - 11.4, p. 418
Empirical risk - 1.2, p. 18
Ensemble - 7.3, p. 278; 7.4, p. 281; 7.5, p. 289
Exchangeability - 9.5, p. 349
Feature - 1.1, p. 16; 2.1, p. 60
Finite difference - 12.4, p. 450
Gaussian process - 10.1, p. 365; 10.3, p. 371
Generalization - 1.2, p. 18; 1.3, p. 22
Gradient boosting - 7.4, p. 281
Gradient descent - 2.3, p. 70
Hierarchical clustering - 8.3, p. 311

Hyperparameter - 4.4, p. 162
Inductive bias - 1.5, p. 28
K-means - 8.1, p. 301
K-nearest neighbors (KNN) - 6.1, p. 227
Kernel - 6.4, p. 241; 10.2, p. 368
Likelihood - 2.2, p. 64; 3.2, p. 105
Logistic regression - 3.1, p. 104; 3.2, p. 105
Loss function - 1.2, p. 18; 3.2, p. 105
Margin - 6.2, p. 229
Naive Bayes - 11.3, p. 413
Nested cross-validation - 4.4, p. 162
Neural network - 12.1, p. 433; 12.2, p. 436
Overfitting - 1.5, p. 28; 4.4, p. 162
Partial least squares (PLS) - 5.4, p. 204
Pipeline - 2.1, p. 60; 4.3, p. 160
Population risk - 1.2, p. 18
Precision and recall - 3.3, p. 114
Principal component analysis (PCA) - 5.2, p. 192; 5.3, p. 202
Proper scoring rule - 9.1, p. 331
Random forest - 7.3, p. 278
Regularization - 2.4, p. 80
ROC curve - 3.3, p. 114
Support vector - 6.3, p. 233
Support-vector machine (SVM) - 6.2, p. 229; 6.4, p. 241
Support-vector regression (SVR) - 6.5, p. 250
t-SNE - 5.5, p. 213
Test lock - 4.5, p. 170
TF-IDF - 11.2, p. 404
Validation set - 1.3, p. 22; 4.4, p. 162