ArtificialGate

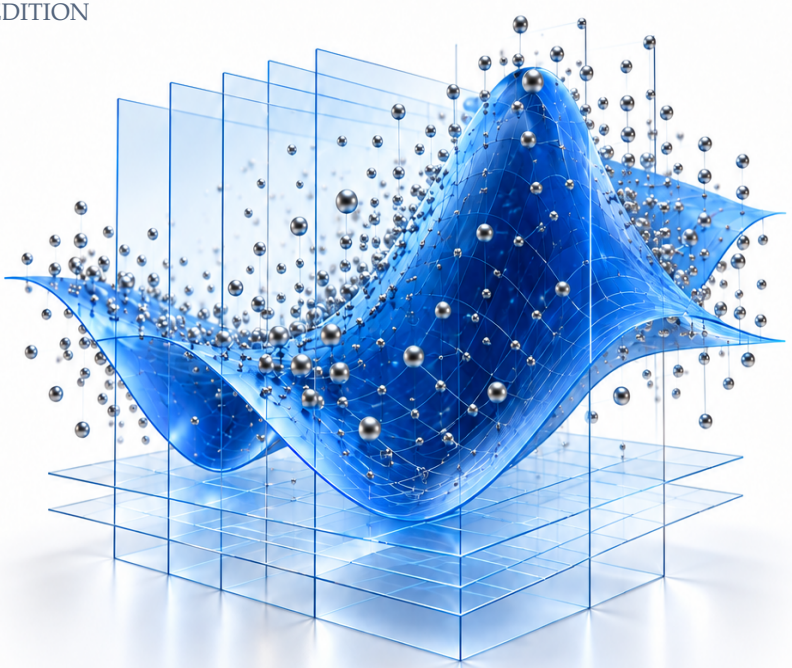
Modern

Machine Learning

Practical Course Notes

Dr. Barak Or

2027 EDITION



PREFACE

From statistical foundations to reliable prediction

Machine learning is the study of systems that infer useful structure from data. This book develops the mathematical ideas needed to formulate prediction problems, fit models, evaluate their generalization, and understand where they fail. Classical methods are treated as useful tools in their own right and as the foundation for deep learning.

The treatment assumes basic Python, linear algebra, probability and differentiation. It develops graduate-level understanding through broad applications in numerical prediction, classification, representation learning and experimental design. The next section explains the independent-study route and prerequisite support.

The first four chapters establish statistical risk, preprocessing, optimization, supervised prediction and evaluation. Chapters 5-8 develop representations, local and kernel methods, ensembles and clustering. Chapters 9-10 address interpretation, uncertainty and sequential data acquisition. Chapters 11-13 connect these foundations to neural networks, controlled training experiments and language representations.

Each section develops an explanation, explicitly defined mathematical notation and an example. Theoretical results state their assumptions and include proofs or derivations. Primary equations are numbered consecutively within each chapter; the T-series identifies steps in theoretical developments. The B-series marks additional derivations and examples that bridge theory and application. Independent formulas receive separate displays and numbers. A single long identity may continue on aligned lines, without reducing its mathematical type size.

Reading key

◇ Theory: assumptions, statement and mathematical result

The diamond marks a theorem, proposition or derivation. Its assumptions delimit the claim.

□ Proof: the argument supporting a result

The open square introduces a proof or derivation. A small square at the end marks its completion.

▶ Example: a concrete use of the idea

The triangle identifies a numerical or practical example. Algorithms use numbered, indented pseudocode with explicit inputs, outputs and checks. These markers are reading aids; they do not replace mathematical conditions.

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.

How To Study This Book

This edition is designed for independent study. Read each chapter's opening problem and capabilities first. Follow the core route for a first pass, reproduce the Examples, then examine the real-data Case Study. Return to the deepening route when the practical mechanism is clear. A deepening label marks reading order, not dispensable mathematics.

The mathematical prerequisites are basic Python, vectors and matrices, elementary probability and differentiation. The Mathematical Supplement at the end of this book provides a concise refresher and develops additional results on generalization and optimization. The chapters retain the theoretical statements and core proofs; internal links lead from selected statements to their complete supporting proofs in the supplement. Every chapter ends with comparison and synthesis tables, measured case evidence and references.

All exercises, questions, prerequisite diagnostics, solutions and video-production scripts are in the separate course-aids package. For self-study, attempt the Practice notebook before opening its Solution. Passing implementation assertions checks a contract; explaining a result and solving a changed task are separate capabilities. The guide gives the sequence and evidence required for those checks.

The notebook schedule totals approximately 54 hours before reading, proofs, reflection and troubleshooting. The 39 short-video scripts total about two hours of planned narration; they orient and demonstrate, but do not replace the book or practice. Proceed by demonstrated understanding rather than by video completion.

The reported Case Study results are measured reference runs on declared samples and software, not promised benchmark scores. Variations across splits, seeds, library versions and machines can be informative. The book and course aids have undergone internal content, numerical and layout checks; no independent external review or learner-outcome study is claimed.

Mathematical Supplement: end of this book | Open the separate course aids

Contents

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

How To Study This Book 4

Notation and conventions 11

01. ML Problems, Data, And Generalization 16

1.1 Define the prediction problem before selecting an algorithm 18

1.2 Population risk, empirical risk and the best squared-error prediction . . . 20

1.3 Generalization requires a disciplined data lifecycle 24

1.4 Dimensionality changes geometry and data requirements 28

1.5 Bias, variance and inductive bias 32

From A Finite List To A Continuous Linear Model 39

Case Study: Iris: write the prediction contract before chasing accuracy 45

02. Regression, Optimization, And Regularization 48

2.1 Preprocessing is part of the fitted model 50

2.2 Least squares and its Gaussian likelihood interpretation 56

2.3 Gradient descent as a discretized gradient flow 63

2.4 Regularization controls the fitted solution 71

2.5 Report errors in a form the application can use 79

Case Study: Diabetes: connect the ridge objective to held-out error 84

03. Classification And Imbalanced Data 87

3.1 Logistic regression models a conditional probability 89

3.2 Bernoulli likelihood leads to binary cross-entropy 91

3.3 Confusion matrices and ranking metrics answer different questions . . . 100

3.4 Thresholds connect predictions to decisions . . . 114
3.5 Handle imbalance without rewriting the evaluation population . . . 122
Case Study: WDBC: separate ranking, probability quality and decision cost . . . 129
What Zero Observed Misses Can Still Hide . . . 131

04. Validation And Model Selection **133**

4.1 Holdout, K-fold, stratification and leave-one-out . . . 134
4.2 Group and temporal splits must match the intended use . . . 139
4.3 The validation boundary surrounds the entire pipeline . . . 142
4.4 Search selects a model; nested validation evaluates that selection . . . 144
4.5 Uncertainty, reproducibility and the final test lock . . . 154
Case Study: California Housing: random interpolation is not northward transfer . . . 159

05. Representations, PCA, And PLS **162**

5.1 Representations, scale, and numerical conditioning . . . 164
5.2 PCA as variance maximization and reconstruction . . . 167
5.3 Choosing dimension for reconstruction or prediction . . . 177
5.4 Partial least squares uses target information . . . 179
PLS1 From Residual Directions To New Predictions . . . 183
5.5 t-SNE is a neighborhood visualization . . . 188
Case Study: Red Wine Quality: reconstruction variance versus predictive signal . . . 191

06. Distance, Margins, And Kernels **194**

6.1 Nearest neighbors and the meaning of distance . . . 196
6.2 Maximum margins and soft constraints . . . 198
6.3 Duality, KKT conditions, and support vectors . . . 202
6.4 Kernels and nonlinear support-vector classification . . . 208

From Binary Margins To A Multiclass SVC 214
6.5 Support-vector regression and practical model choice 217
Case Study: Digits: verify a margin, then compare local and kernel prediction 224
From Error Counts To Paired Evidence 226

07. Trees And Ensemble Learning **230**
7.1 Regression trees as recursive partitions 231
7.2 Classification impurity and complexity control 236
7.3 Bagging and random forests 241
7.4 Gradient boosting follows the loss gradient 245
7.5 Voting and leakage-free stacking 251
Case Study: Compare tree ensembles on historical Adult census records . . 258

08. Clustering And Anomaly Detection **261**
8.1 K-means and alternating optimization 262
8.2 Evaluating cluster structure without inventing labels 271
8.3 Hierarchical clustering and linkage choices 273
8.4 DBSCAN and density connectivity 276
8.5 Anomaly scores, thresholds, and drift 284
Case Study: Discover chemical groups without choosing them from cultivar labels 287

09. Probability Calibration And Model Diagnosis **291**
9.1 Proper scores and probability information 292
9.2 Estimating and correcting calibration 297
9.3 Error slices and conditional reliability 304
9.4 Interpreting predictive dependence 308
9.5 Distribution shift and the limits of uncertainty guarantees 311

Case Study: Audit probability claims on Wisconsin diagnostic measurements . . . 323

10. Gaussian Processes And Experiment Selection **326**

10.1 A distribution over related function values . . . 327

10.2 RBF and Matérn kernels encode smoothness . . . 330

10.3 Conditioning and interpreting posterior uncertainty . . . 333

10.4 Active learning asks which label improves prediction . . . 337

10.5 Bayesian optimization asks where the objective is high . . . 344

Case Study: Replay a limited-label GP study on real diabetes measurements . . . 354

11. Neural Network **358**

11.1 From linear models to a neural network . . . 359

11.2 Activation functions and learned features . . . 362

11.3 From prediction error to backpropagation . . . 373

From A Scalar Regressor To A Multiclass Neural Classifier . . . 380

11.4 Checking derivatives independently . . . 384

11.5 Automatic differentiation of the same network . . . 390

Case Study: Verify a handwritten network on actual MNIST threes and eights . . . 393

12. Training And Debugging Neural Networks **396**

12.1 A mini-batch loop has a precise order . . . 397

12.2 Initialization and optimizers shape the trajectory . . . 402

12.3 Regularization and early stopping constrain fitting . . . 409

12.4 Training mode, inference mode, and checkpoints are distinct . . . 418

12.5 Diagnose fitting failures before expanding the model . . . 420

Case Study: Select and reload a Fashion-MNIST training run honestly . . . 426

13. Text Classification And Embeddings **429**

13.1 From documents to count vectors 430

13.2 TF-IDF reweights lexical evidence 432

13.3 Naive Bayes, logistic regression, and linear SVMs 441

13.4 Frozen pretrained embeddings provide a different representation . . . 448

13.5 Compare representations fairly and inspect failures 454

Case Study: Classify real newsgroup topics without metadata shortcuts . . 458

Measured Extension: A Genuine Frozen Encoder On The Same Public-Text Task 459

From machine learning to deep learning **465**

Conceptual Glossary **466**

Subject Index **472**

About The Author **474**

Mathematical Supplement **475**

S1 Mathematical Refresher 475

S2 Finite-Class Generalization 479

S3 Gradient Descent Guarantees 484

S4 Applying The Guarantees 490

A1 Local Coverage 493

A2 ROC AUC And Ties 495

A3 Cross-Validation Estimands 499

A4 First PLS Direction 501

A5 Soft-Margin SVM Dual 503

A6 Ward Linkage 505

A7 Expected Improvement 506
A8 ReLU Approximation 509
A9 Mean-Pooling Stability 510
A10 Marginal Likelihood 513
A11 Gradient Amplification 515
A12 Additive Smoothing 517

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.

$a^b,\ \sqrt{a},\ |a|$ 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.

$\cup,\ \cap,\ \setminus,\ |S|$ 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.

$\Rightarrow,\ \Leftrightarrow,\ \rightarrow$ 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.
$\sum_{i=1}^{n},\ \prod_{i=1}^{n},\ \dots$	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.

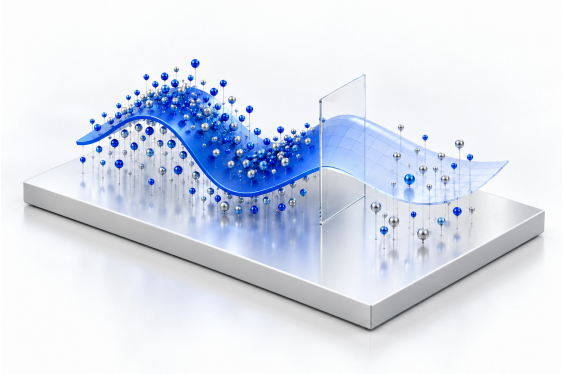
Data and numerical conventions

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



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.

Core route: 1.1, 1.2, 1.3, 1.5. Deepening route: 1.4.

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.

Machine learning constructs useful patterns from examples, but the pattern is only one part of the system. A forecasting service, a document classifier and an image recognizer all need a precisely defined target, information available at prediction time, and an evaluation that resembles intended use. This chapter introduces that common structure. Neither a particular industry nor a particular dataset size defines the field: the same questions matter with hundreds of observations and with millions.

Our central distinction is between fitting the observations we possess and performing well on observations we have not used to make choices. We develop this distinction through statistical risk, data boundaries, dimensionality and repeated-training experiments. The mathematics supplies a language for reasoning, not a promise that a theorem automatically certifies a deployed system. Throughout this book, simple baselines and controlled comparisons turn that language into testable claims.

1.1 Define the prediction problem before selecting an algorithm

A supervised problem pairs an input with an observed target. Regression predicts a numerical 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, while other learning settings involve additional feedback or interaction. The distinction concerns the training information and objective, not whether the application appears intelligent or whether its model is complex.

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.

Inputs are representations, not neutral copies of reality. A numerical column may encode a measurement, a category, an identifier or information created after the outcome. Features need not be statistically independent. The design matrix has n rows and d input columns; each observation can be written as a column vector even though it occupies a row in the matrix. Clear shapes and availability rules prevent both mathematical mistakes and misleading experimental success.

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.

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

A loss measures how undesirable an individual prediction is. Squared error assigns a penalty equal to the squared numerical difference; absolute error uses its magnitude. The appropriate loss depends on the question and the costs of mistakes. Population risk averages this loss over a specified joint distribution of inputs and targets. It describes the prediction problem we want to solve, but that 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.

For squared error, the conditional mean minimizes expected loss at each fixed input, provided the relevant second moment exists. Expand the squared difference around that mean: the conditional variance remains, while the squared distance between the chosen prediction and the mean is nonnegative. This result does not assert that the relationship is linear. Under absolute error a conditional median is optimal instead, illustrating that changing the loss can change the ideal prediction.

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 Mathematical Supplement S2 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.

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

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

Fix an input and write the target as its conditional mean plus a residual. Defining the residual this way is important: unconditional zero-mean noise alone would not justify the conditional calculation. The conditional mean of this residual is zero by construction.

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

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

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

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

For an arbitrary scalar prediction, expand the square before taking expectation. The cross term contains a constant at the fixed input multiplied by the zero-mean residual; therefore it vanishes.

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

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

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.

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

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

Equality of risks holds precisely when the candidate equals the conditional mean except on a set of inputs with probability zero. Predictions at unobserved, zero-probability locations are not identified by this risk. The proof avoids interchanging differentiation and integration, so finite second moments and conditional expectation suffice.

□

The result separates the prediction target from the model class used to approximate it. A linear model may fail to represent the conditional mean; extra optimization cannot repair that representational restriction. Conditional variance is irreducible relative to the currently available inputs, not a universal physical limit: an additional informative measurement can reduce it.

▶ Example: Choose a constant using the loss

At one input, let targets zero, two and six have probabilities 0.2, 0.5 and 0.3. Their weighted mean is 0.2 times zero plus 0.5 times two plus 0.3 times six: 2.8. The weighted mean of their squares is 12.8; subtracting 2.8 squared gives squared-error risk 4.96. Predicting two instead gives 5.60, an increase of 0.64, exactly the squared displacement from 2.8. Under absolute loss, two is a median and attains risk 1.6. These calculations describe the declared

distribution, not measured performance: changing the loss changes the preferred prediction.

Interpretation and limits

Low empirical risk on the fitting data does not establish low population risk. State both the evaluation population and the independence of the observations used to estimate performance.

1.3 Generalization requires a disciplined data lifecycle

A basic workflow assigns training data to parameter fitting, validation data to model-development choices, and test data to a final evaluation after those choices are frozen. The labels describe roles, not universal percentages. Reusing a validation set many times can itself cause selection effects. A test set stops being a clean final check when its results influence feature engineering, thresholds, architecture choices 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 held-out average fluctuates because its observations are a sample. Selecting the smallest of several averages can exploit those fluctuations. Supplement S2 proves a finite-class bound: with bounded loss and independent, identically distributed observations, one can control the estimation error for every predictor in a fixed finite collection simultaneously. The selection allowance is twice the simultaneous estimation allowance, plus any empirical optimization tolerance.

The practical condition is easy to miss: the collection must be independent of the sample used to score it, or belong to a fully specified fixed family covered by the bound. Counting only the fitted models tried on their own training data is not enough. Supplement S2 gives the proof and a numerical sample-size example; S1 refreshes the probability tools. This result complements, rather than replaces, the separate development and final-evaluation roles described here.

Mathematical Supplement: S2

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.

Treat records as indexed observations. Distinct records may still share entities or leaked information, so disjointness alone is insufficient.

▶ 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

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

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

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

Record Splits, fitted preprocessing, versions, sample counts and validity limits.

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

Why this works. Retain the selected training fit; do not silently refit on validation. Record the number of independent evaluation units separately from the row count when groups exist. The finite plan terminates after its declared fits. A model change motivated by test outcomes needs new independent evidence for a new final claim.

1.4 Dimensionality changes geometry and data requirements

Additional features may reveal important signal, but they also change the geometry used by a learning algorithm. Consider independent uniform coordinates inside a unit hypercube. A smaller aligned cube with side length r occupies a fraction r raised to d of the volume. Holding that fraction fixed while increasing d requires a progressively wider neighborhood. This simplified calculation explains why locally averaging nearby examples can become difficult when many coordinates matter.

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 a uniform unit hypercube, q is the volume fraction of an aligned subcube with side length r. This geometric illustration assumes independent uniform coordinates and is not a sample-complexity bound for every model.

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.

Complete supporting proof: Mathematical Supplement, A1 (in this book).

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.

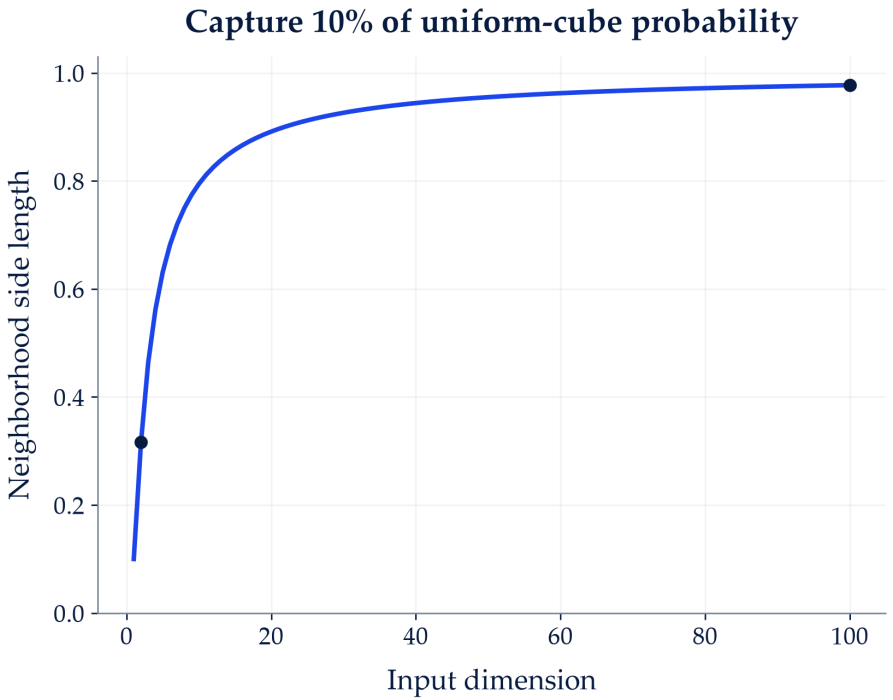


Figure 1.1. Exact side length needed for a fully contained axis-aligned neighborhood to hold 10% of a unit hypercube with independent uniform coordinates. As dimension increases, maintaining the same probability mass forces the neighborhood to span most of each coordinate. This is a geometric coverage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

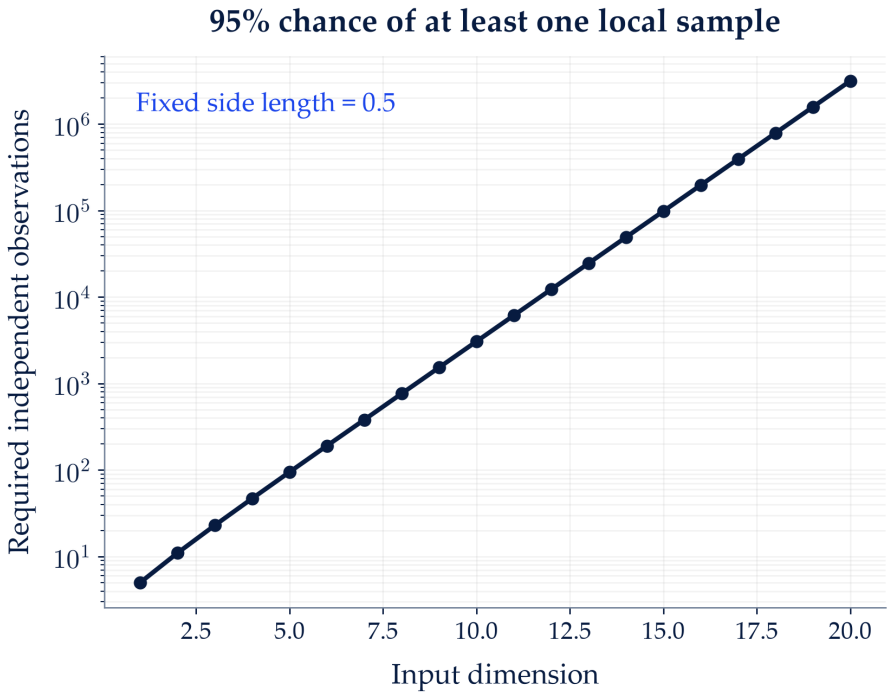


Figure 1.2. Exact smallest independent sample count giving at least 95% probability of one observation in a fully contained subcube of side 0.5. The probability of missing it in every draw is evaluated directly. A single local observation is not enough for accurate regression; the curve describes coverage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

1.5 Bias, variance and inductive bias

Imagine repeatedly drawing training datasets from the same population and refitting the same procedure. At a fixed input, the resulting predictions vary. Their average may also differ systematically from the conditional mean target. With finite second moments, squared loss, fresh outcome noise whose mean is zero conditional on the input, and conditional independence between that noise and the training procedure, expected prediction error separates into squared bias, prediction variance and conditional noise variance. The expectation is over repeated training and fresh outcomes, not just different 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.

Inductive bias is the set of preferences a learning procedure brings to incomplete evidence. Linearity, smoothness, locality, sparsity and shared parameters are examples. Such assumptions may support useful generalization across many related applications; they do not guarantee failure everywhere else. No-free-lunch results depend on particular averaging assumptions over possible problems and should not be turned into a universal ranking of real-world models. Compare plausible assumptions using evidence from the task rather than searching for a model that is best without qualification.

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.

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.
$\mathbb{E}_{D,Y\mid x}$	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.

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

Bias is the difference between the mean fitted prediction over training datasets and the conditional mean target. Fresh outcome noise must be independent of the trained predictor conditional on x. This is a squared-error decomposition.

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.

Expected squared error over repeated fitting and fresh target outcomes equals squared bias plus prediction variance plus conditional target variance. The two vanishing cross terms have different reasons and must both be justified.

□ **Proof**

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

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

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.

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

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

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

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

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

Integrating over independent evaluation inputs gives the corresponding population decomposition. Reusing training targets removes the independence needed for the first cancellation, which explains why the same formula cannot simply certify a training score. The decomposition is an identity, not a statement that bias must always decrease and variance must always increase with a chosen complexity parameter.

□

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. A learner observes labels at a strict subset of inputs. Its prediction at an unobserved input may depend on observed labels and independent algorithm randomness, but not on the hidden label.

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**

Condition on everything the learner has observed, including its random seed. The selected output at a hidden input is then fixed, but the hidden label remains an independent fair coin. If the learner predicts zero, half of the remaining possible labelings disagree; if it predicts one, the other half disagree.

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

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

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: 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.

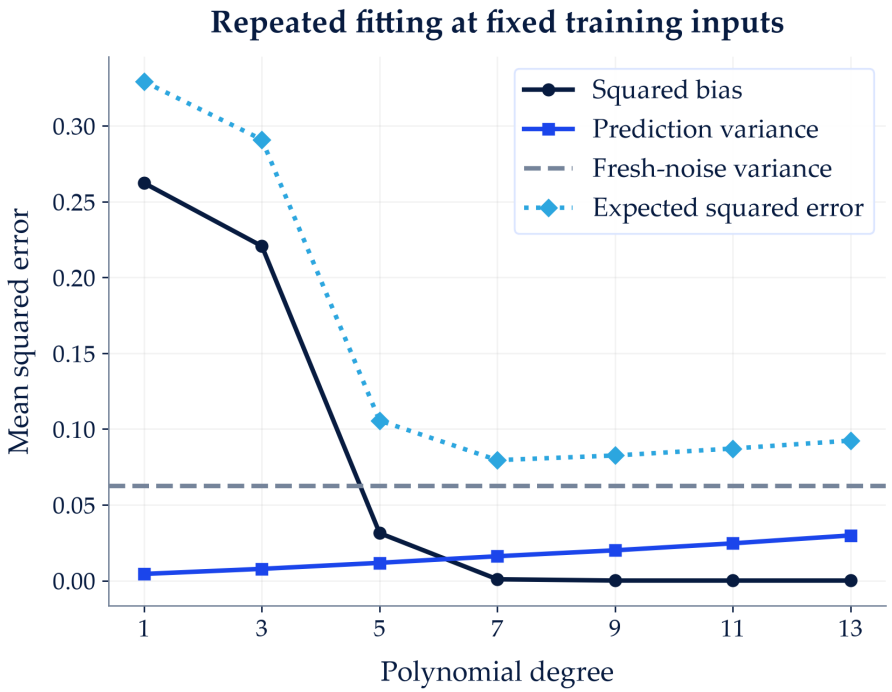


Figure 1.3. A repeated-fit experiment with 600 independent noisy target samples at the same 25 inputs. Legendre least-squares models are evaluated on 201 inputs. The displayed expected squared error is the sum of Monte Carlo squared bias, prediction variance and known fresh-noise variance 0.25 squared. These estimates are conditional on the fixed input design.

Notation for Figure 1.4

x	Dimensionless scalar input to this synthetic conditional-mean curve; the evaluation grid runs from minus 0.9 to 0.9.
sin	Sine function; its argument is measured in radians.
π	Circle constant, approximately 3.142, in the synthetic sine relationship; not a class prevalence.

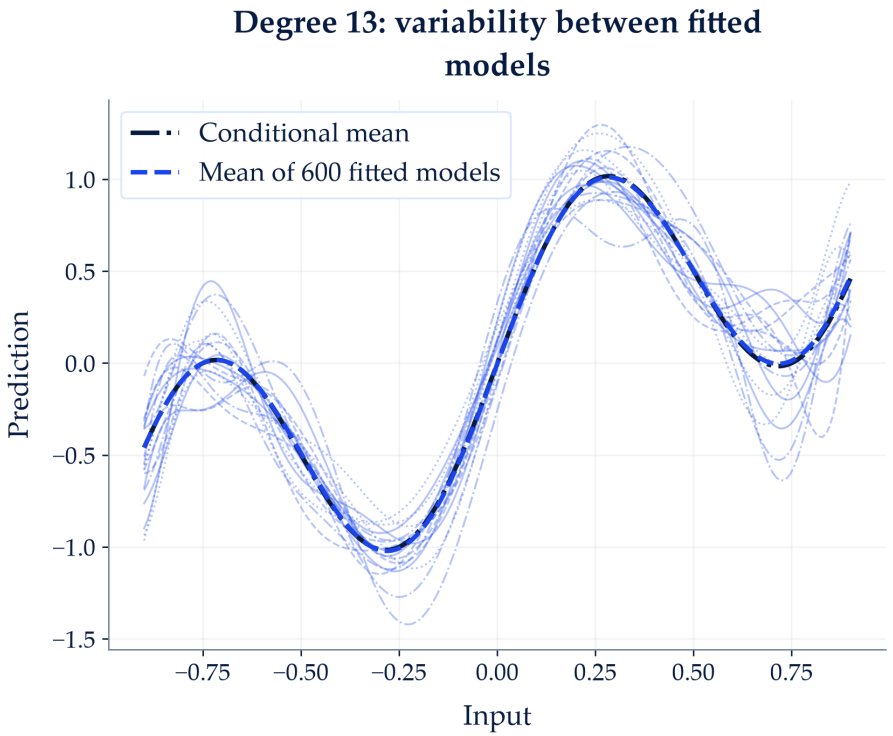


Figure 1.4. Eighteen degree-13 least-squares fits from the same 600-sample experiment are shown as pale curves. The dashed curve is the mean of all 600 fitted models; the dark curve is the known conditional mean x + 0.75 sin(2 pi x). Variation between fitted predictions is distinct from the bias of their 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. The finite-class theorem in Supplement S2 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 $\|X\|_2 \le B$ 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.

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

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 $\rho \le A$ 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.

The theorem is not a guarantee for raw squared error, log loss with unbounded logits, or discontinuous zero-one loss. Nor does it cover data-dependent feature construction merely by counting the final coefficients. Fix the representation independently or control the larger representation-and-predictor family. Smaller covers reduce sampling uncertainty but approximate the family less accurately; that trade-off is visible in (1.B5). A norm constraint can limit estimation error while excluding useful predictors, so approximation error remains. Regularized optimization also needs its own comparison argument: minimizing a penalized objective is not automatically the empirical-risk minimization assumed in (1.B9).

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

Notation for Figure 1.5

x	Scalar synthetic input on the horizontal axis, shown from approximately minus one to one; it has no physical unit here.

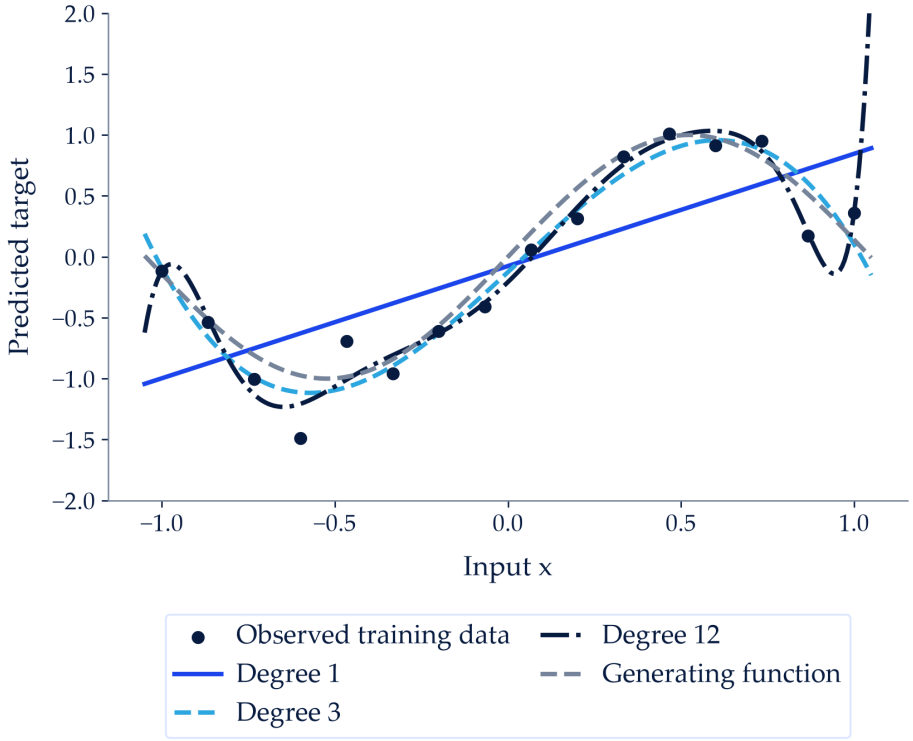


Figure 1.5. A reproducible synthetic sample with 16 observations. All fitted curves use the same training data; the dashed curve is known only because this is a simulation. Training fit alone cannot select the model.

Table 1.1. 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.

Problem	Observed evidence	Output and main limitation
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.

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?
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.

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 measurements improved substantially on the constant reference in this split. Greater capacity did improve validation performance here: the unrestricted tree beat, rather than merely tied, both shallower trees. Its final zero-one risk was 0.033, versus 0.667 for the reference.

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.

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

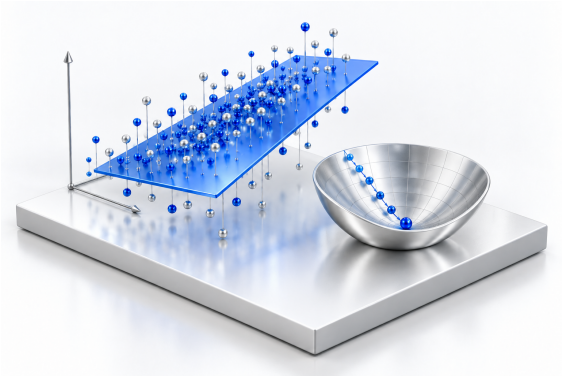
[1.7] Hoeffding, W. (1963). Probability Inequalities for Sums of Bounded Random Variables. Journal of the American Statistical Association 58(301), 13-30. Source / DOI

[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

CHAPTER 02

Regression, Optimization, And Regularization



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.

Core route: 2.1, 2.2, 2.3, 2.4, 2.5. Revisit the proofs after reproducing the Examples.

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 provides a compact setting in which the entire learning process can be made visible. We transform imperfect observations into a numerical representation, specify a predictive function, derive an objective, minimize it and evaluate the result. The same sequence later appears in neural-network training, although the function and optimization landscape become more complicated. Understanding one transparent model is therefore more valuable than treating its fitting method as an unexplained library call.

This chapter uses linear prediction as a foundation, not as a claim that every useful relationship is linear. Feature transformations can make a linear-in-parameters model nonlinear in its original inputs. Regularization changes the fitting objective, and preprocessing changes both the numerical geometry and the interpretation of that regularization. We will keep those choices explicit and use measured outputs rather than assuming that one model or penalty must win.

2.1 Preprocessing is part of the fitted model

A numerical table can contain missing measurements, differently scaled variables, categorical values and mistakes. Begin by distinguishing an unknown value from a meaningful zero, and a rare observation from an invalid one. Deleting every incomplete row can alter the represented population. Replacing missing values with a training median is a useful baseline, but it is not a universal statistical solution. Missingness may itself contain information, and the mechanism that produced it deserves investigation.

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.

Column standardization makes the Hessian diagonal equal to one, but the off-diagonal entries remain the empirical correlations. It produces a spherical quadratic objective only when the standardized columns are uncorrelated.

□ **Proof**

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

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

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

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.

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

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

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

The diagonal entries equal one. For two standardized features with correlation close to one, the eigenvalues below remain very different. Thus equal marginal variances do not remove an almost redundant direction. A whitening transform could make an invertible covariance identity, but that is a different fitted operation and requires handling small or zero eigenvalues.

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

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

□

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.

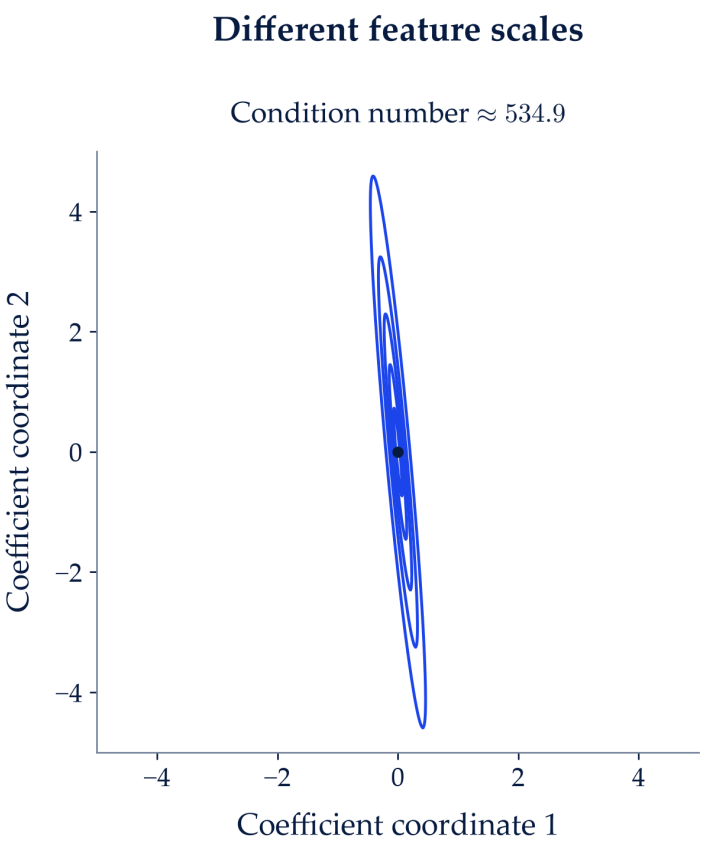


Figure 2.1. Level sets of a coefficient-space quadratic with Hessian diagonal entries 100 and 1 and off-diagonal entry 9. Both scale imbalance and correlation contribute to the elongated contours. The condition number is the ratio of the two positive Hessian eigenvalues.

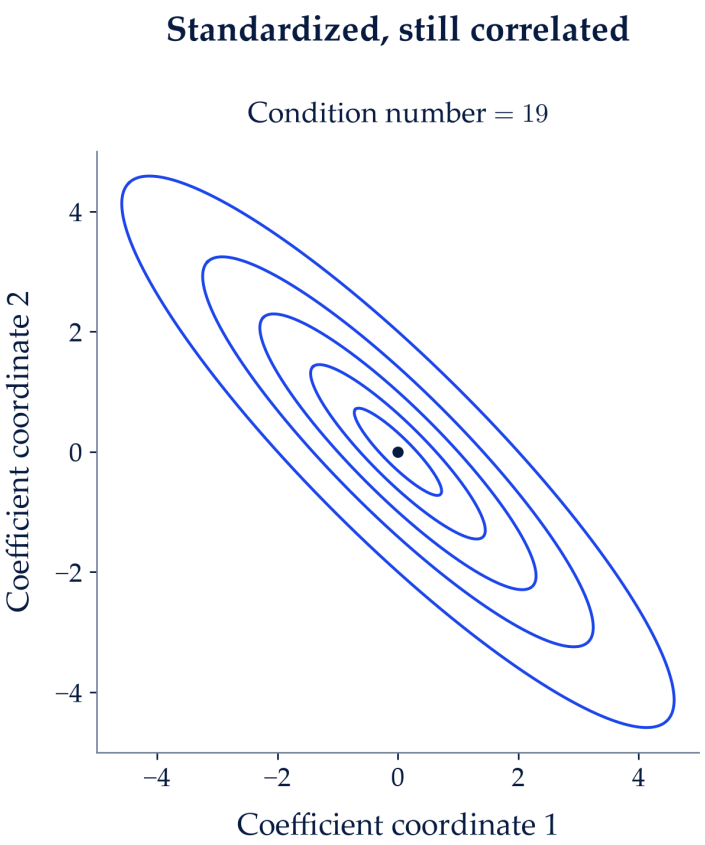


Figure 2.2. Standardizing the two feature scales makes the Hessian diagonal equal to one but leaves off-diagonal correlation 0.9. The resulting condition number is 19, not one. Unit marginal variance does not remove dependence between the coordinates.

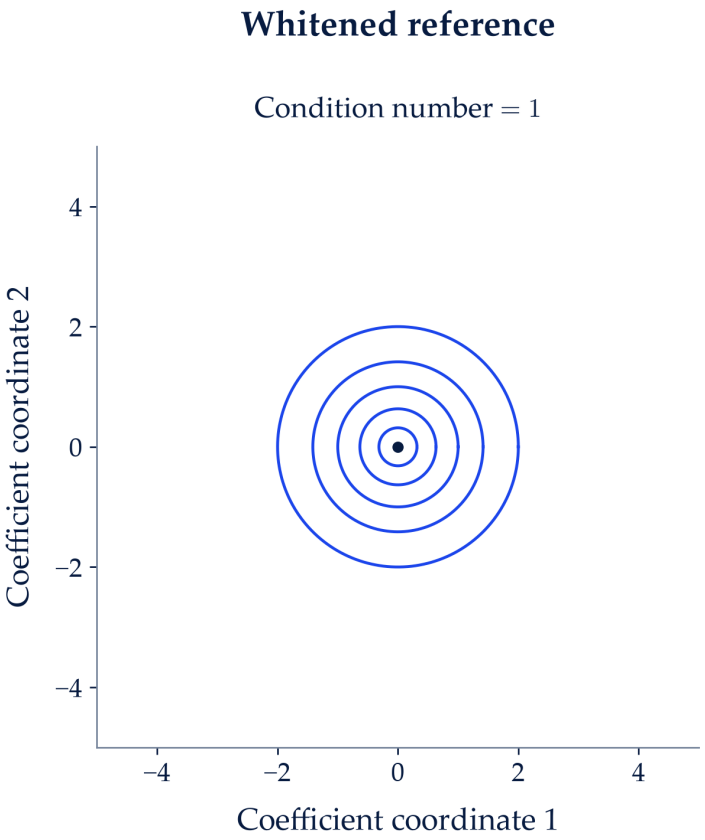


Figure 2.3. An identity Hessian produces circular level sets and condition number one. It is the whitened reference for this example, not the automatic result of feature-wise standardization. Coordinates belong to this transformed parameterization.

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.

A linear regression model predicts the inner product of the coefficient vector and the input, plus an intercept. The coefficients have one entry per represented feature, and the intercept shifts the prediction independently of those features. The word linear refers to the parameters. If a representation includes a squared input or an interaction, the prediction can be nonlinear in the original variables while remaining linear in the fitted coefficients.

Ordinary least squares (OLS) minimizes squared residuals. Multiplying its entire unregularized objective by a positive constant preserves the minimizers but rescales the gradient. Multiplying only the loss while leaving a penalty unchanged generally changes the fitted solution. Here the objective is half the mean squared error (MSE), so both gradient updates and penalty translations must use that normalization. Solve least squares by a stable factorization rather than explicitly inverting the normal-equation matrix.

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

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

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

Write the density for each observation, then multiply because the errors are conditionally independent. A density is not the probability of observing one exact real value; likelihood evaluates this density at the observed target vector while varying the parameters.

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

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

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.

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

If variance is also estimated and the minimum residual sum is positive, differentiating the negative log-likelihood in the variance gives the residual mean square at the least-squares fit. This is the likelihood estimate, not the degrees-of-freedom-corrected unbiased estimate. If the residual sum is zero, variance can approach zero and the likelihood is unbounded; there is no finite joint maximum 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.

A coefficient vector minimizes half-mean-squared error exactly when its residual is orthogonal to every design column. The resulting fitted vector is the orthogonal projection of the targets onto the design's column space.

□ Proof

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

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

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

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.

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

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

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

The cross term vanishes because the design transpose times the residual is zero. The added term cannot be negative, proving optimality. It is strictly positive for every nonzero perturbation exactly when the columns are independent. In that case the Gram matrix is invertible.

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

With rank deficiency, different coefficients can differ by a null-space vector and make identical predictions on the design. The fitted vector remains unique because the projection onto a linear subspace is unique. In numerical work solve the least-squares system with QR or singular-value methods; the inverse expression identifies the mathematical solution but is not a recommendation to form an inverse.

□

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.

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.

Stack the intercept and coefficients into one parameter vector and append a column of ones to the design. For half mean squared error, multiply the design transpose by predicted minus observed targets and divide by n. Check each gradient coordinate by perturbing that parameter equally forward and backward while holding the data fixed. Compare several small perturbation sizes: truncation dominates large steps, while floating-point cancellation can dominate extremely small ones.

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. Supplement S3 proves 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. Supplement S3 separates these conclusions, and S4 explains normalization and feature-scaling consequences. Neither result turns low training loss into evidence of low population risk.

Mathematical Supplement: S3

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**

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

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

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.

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

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

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

Diagonalize the symmetric Hessian in an orthonormal basis. Each coordinate evolves independently, so the entire stability question becomes a scalar recurrence for each curvature eigenvalue.

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

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

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

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

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

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

Inside this interval every positive-curvature coordinate decays geometrically. At the upper boundary the largest-curvature component oscillates with unchanged magnitude; beyond it, that component diverges unless initially zero. A zero eigenvalue gives a factor of one, leaving a null-space component unchanged. Objective scaling therefore changes the admissible step: residual sum of squares and half-mean-squared error cannot share stability constants without rescaling.

□

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 the objective one half of theta squared, the gradient is theta and each update multiplies it by one minus eta. Starting at one, eta equal to 0.5 gives one, 0.5, 0.25, 0.125: the multiplier has magnitude below one. At eta equal to 2.5, it gives one, negative 1.5, 2.25, negative 3.375: magnitudes grow. At the boundary eta equal to two, the iterates alternate between one and negative one without converging. The notebook checks the divergent case and a multivariable gradient before comparing predictions with least squares. The objective is unchanged; the update stability is not.

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

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.

$r = A\theta - y, \quad J = \frac{\|r\|_2^2}{2n}$ Prediction-minus-target residual and half-mean-squared objective; ||·||_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.

$\text{finite}, \text{Finish}$ 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 $\neg\,\mathrm{finite}(g)$

  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 $\neg\,\mathrm{finite}(\theta', r', J')$

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

Algorithm 2.1 (continued)

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

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.

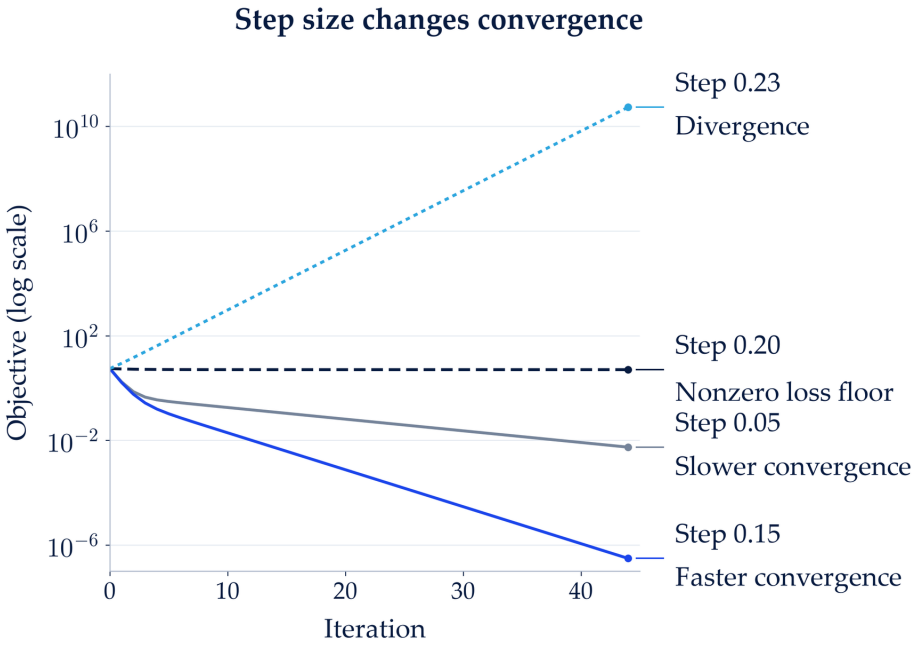


Figure 2.4. Exact gradient-descent iterates on a quadratic with Hessian eigenvalues 1 and 10, from error vector (1, 1). Constant steps strictly between zero and 0.2 converge. At 0.2 the high-curvature component oscillates without decaying; at 0.23 it diverges despite the continuous gradient flow being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

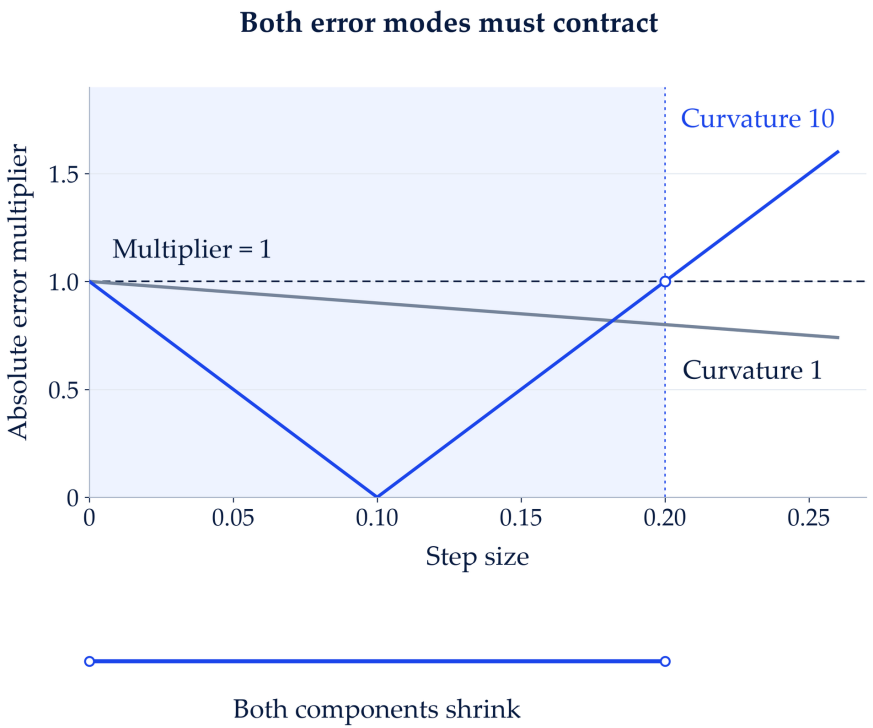


Figure 2.5. Absolute error multipliers for curvatures 1 and 10. Both are below one only for step sizes strictly between zero and 0.2 (open-ended bar). At 0.2 the curvature-10 component oscillates without decay; its zero at 0.1 cancels that component alone.

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 adds a preference to the data-fitting objective. Ridge penalizes the squared coefficient norm, discouraging large coefficients and often stabilizing estimates when features are correlated. Lasso penalizes the sum of absolute coefficients and can set some coefficients exactly to zero. Neither penalty guarantees a better evaluation score. Their value depends on signal, sample size, representation and the strength selected using validation rather than the final test set.

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.

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.

For scikit-learn Ridge's sum-loss convention, its alpha equals n times the lambda used here.

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.

Eliminating the intercept reduces Ridge to a centered linear system whose matrix is positive definite even if the feature columns are dependent. The fitted coefficients are unique for every strictly positive penalty strength.

□ Derivation

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

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.

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

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

The strict positivity proves invertibility and uniqueness. It comes from the penalty even when a coefficient direction is invisible to the centered data. A library minimizing residual sum of squares plus its own penalty coefficient represents the same optimum only when that coefficient equals the sample count times the penalty strength used here.

□

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

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. Fit scaling on training records only and retain it for inference.

Algorithm 2.2

Input: Training arrays and declared feature 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 $(w, \mathcal{D}) \leftarrow \mathrm{LS}(A, z)$

Algorithm 2.2 (continued)

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

09 **Return** Also return fitted feature 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

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 $t = 1, \dots, T$

  For $j = 1, \dots, d$

    $r_j \leftarrow r + x_j w_j$

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

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

    $r \leftarrow r_j - x_j w_j$

  End for

  $V \leftarrow \max_j v_j$

  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: for the centered column (-1, 1), target (-2, 2) and penalty 0.5, curvature is 1 and partial correlation is 2. Soft-thresholding gives slope 1.5. The objective decreases from 2 to 0.875; the KKT violation is zero. Correlated predictors may 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.

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.

Mean squared error is the average squared residual and emphasizes large numerical mistakes. Root mean squared error returns that quantity to the target's units. Mean absolute error averages residual magnitudes and responds linearly to unusually large errors. None describes the complete error distribution. A model with an acceptable average can still fail systematically on an important subgroup or produce unacceptable tail errors, so inspect residuals as well as a scalar summary.

The coefficient of determination compares a model's residual sum of squares with the sum of squared deviations from the evaluation targets' mean. On nonconstant evaluation targets, R squared equals one minus that ratio and can be negative. The evaluation mean is part of this metric's reference calculation,

not a deployable baseline fitted on test data. A separately fitted mean baseline must obtain its constant 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.
$\lvert\cdot\rvert$	Absolute value of a scalar residual; signs are removed before its contribution to MAE is averaged.

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

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

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-target evaluation requires a separate convention, and valid nonconstant-target scores can be negative.

▶ Example: One large residual changes the story

Targets one, two, three and four with predictions one, two, three and ten give residual magnitudes zero, zero, zero and six. Divide their sum by four: MAE is 1.5. Square first, average, then take the square root: RMSE is three. The target mean is 2.5 and its squared-deviation sum is five; R squared is one minus 36 divided by five, or negative 6.2. The negative value means these predictions lose to the evaluation-sample mean under this squared-error comparison. Inspect the residuals too: one failure drives the summaries, and 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.

Notation for Figure 2.6

w	Dimensionless scalar coefficient varied along this figure's horizontal axis; it is not the multi-feature coefficient vector.
λ	Dimensionless nonnegative weight of the squared scalar coefficient in this figure's unnormalized illustrative objective.

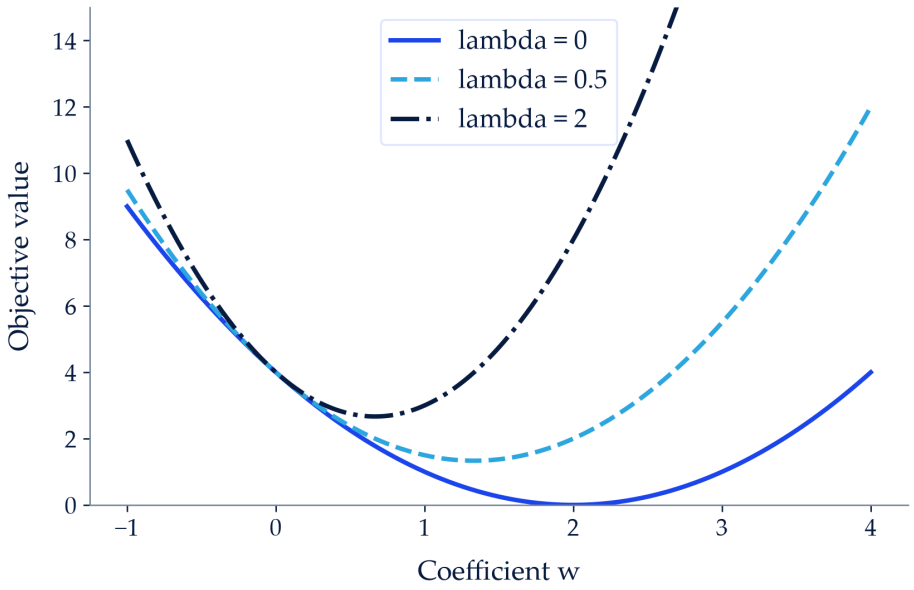


Figure 2.6. Analytical illustration: the data-fit term is (w - 2)^2 and the penalty is lambda w^2. The minimizing coefficient is 2 / (1 + lambda). This scalar example isolates shrinkage; it does not show that every feature or penalty behaves identically.

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.

Method	Objective effect	When the distinction matters
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.

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

Ridge only narrowly improved on least squares in this test. A separate alpha=10 algebra check reduced the coefficient norm from 77.414 to 43.323 while matching the library solution. That check establishes shrinkage under the stated objective, not a predictive advantage for alpha=10. The two R-squared values coincide at the displayed precision.

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

The first raw benchmark record has BMI 32.1 and progression target 151. The notebook computes its prediction only if it belongs to the final-test partition, then reports the actual residual, absolute residual and squared residual. BMI and progression are different quantities; subtracting 32.1 from 151 would not be a 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.

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

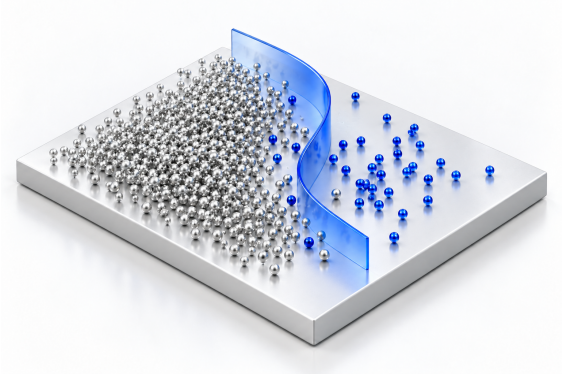
[2.8] Golub, G. H.; Reinsch, C. (1970). Singular value decomposition and least squares solutions. Numerische Mathematik 14, 403-420. Source / DOI

[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

CHAPTER 03

Classification And Imbalanced Data



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.

Core route: 3.1, 3.2, 3.3, 3.4, 3.5. Revisit the proofs after reproducing the Examples.

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.

Classification predicts categories, but a useful classifier often produces more than a category. Its score may support ranking, probability estimation or a decision under a cost constraint. Those uses require different checks. A fraud queue may care about precision among the highest-ranked cases, while an intervention system may require a threshold selected against a declared tradeoff. Neither overall accuracy nor the presence of a probability-shaped output answers every question.

We develop binary classification from a conditional Bernoulli model, then separate fitting, scoring and decision-making. Imbalanced data make this separation particularly important because a system can achieve high accuracy while ignoring the less frequent class. The chapter's synthetic laboratory compares real computed results under a locked test boundary. It does not promise that weighting, resampling or lowering a threshold will improve every application.

3.1 Logistic regression models a conditional probability

For a binary target, encode the two classes as zero and one and state which class is positive. Logistic regression first computes a real-valued score, called a logit, from a linear combination of features and an intercept. The sigmoid transforms that score into a number between zero and one. The model interprets the result as the conditional probability of the positive class, but the quality of that probability still needs evaluation.

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.

Probability estimation and classification are separate operations. A threshold converts a score or estimated probability into a label. The familiar threshold of one half is a convention with a decision-theoretic justification only under particular cost and calibration conditions. Some classification methods output scores that are not probabilities at all. Multiclass problems require a corresponding model and evaluation design; the binary formulas here should not be copied unchanged without deciding how classes are combined.

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.

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.

A Bernoulli variable takes value one with probability p and zero with probability one minus p. Its probability mass can be written compactly using the observed target as an exponent. Assuming examples are conditionally independent under the model, multiply those masses to obtain the conditional likelihood. Taking a negative logarithm turns the product into a sum. Dividing by n gives mean binary cross-entropy, also called 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

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 denominator is positive. Consequently the risk decreases below the true probability and increases above it, with its unique minimum at equality. If the true probability is zero or one, the optimum is the corresponding boundary limit. Averaging this conditional argument over inputs establishes the population target of log-loss fitting.

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.

Unregularized logistic cross-entropy is convex in its coefficients. It is strictly convex at finite coefficients when the augmented design has full column rank, yet strict convexity does not ensure that a finite minimizer exists.

□ Proof

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

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

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

Differentiating one softplus term gives the sigmoid; differentiating the sigmoid gives probability times one minus probability. The chain rule contributes the represented input vector once to the gradient and twice to the Hessian.

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

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

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

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

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

This proves convexity. With full column rank, every nonzero direction has at least one nonzero inner product, making the quadratic form strictly positive. A rank-deficient representation instead has flat directions. Even full rank does not make the objective coercive: its values need not increase when coefficient magnitude grows.

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

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

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

Every finite summand is positive, yet all approach zero along the separating direction. The infimum therefore is not attained by finite coefficients on strictly separable data. A positive squared penalty on all coefficients restores coercivity and strict convexity; a convention that excludes the intercept requires separate attention to existence, especially when the data contain only one class.

□

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.

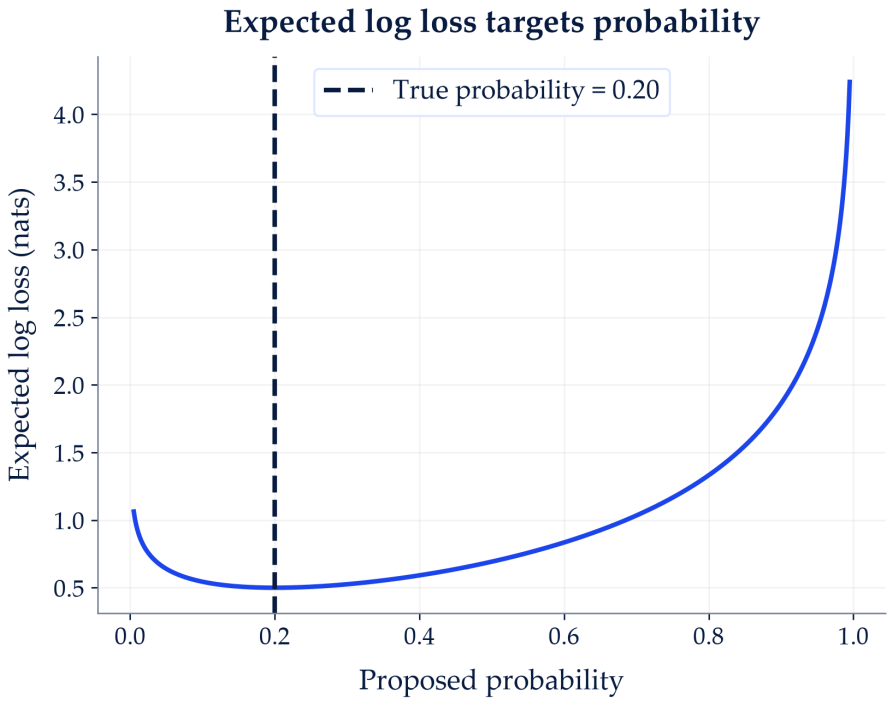


Figure 3.1. Exact conditional expected binary log loss when the positive-class probability is 0.20. Its minimum occurs at that true probability. This population calculation explains probability quality; it is not a measurement of finite-data classifier performance.

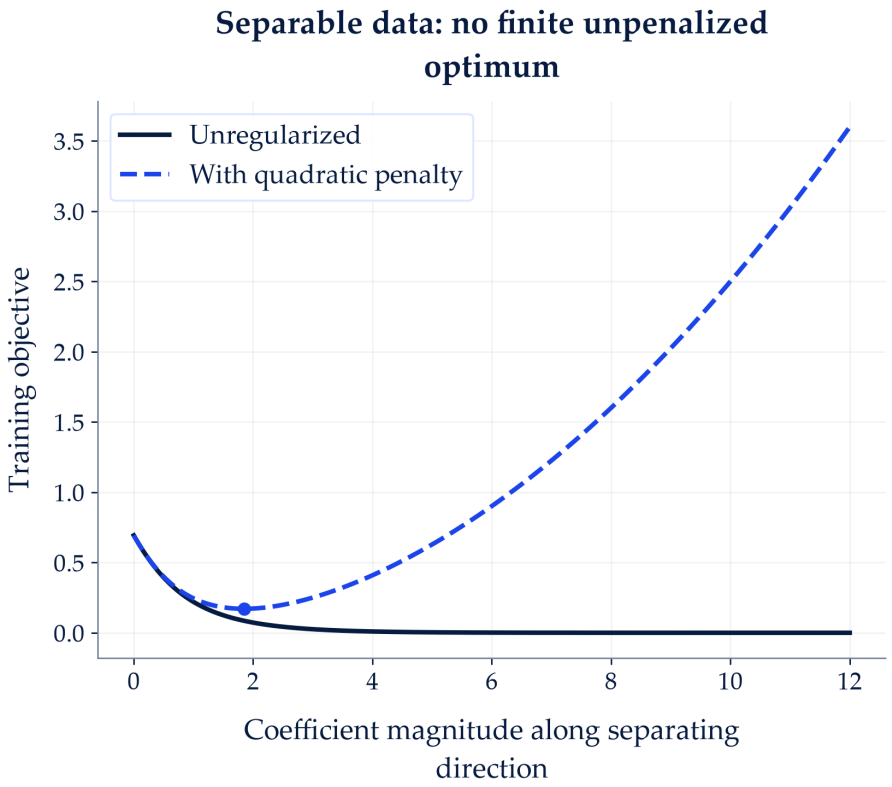


Figure 3.2. Mean logistic loss on separable observations with inputs (-2, -1, 1, 2), targets (0, 0, 1, 1), and one positive slope. Unregularized loss approaches zero only as the slope grows without bound. Adding 0.05 times half the squared slope creates a finite optimum, marked by the grid minimum.

3.3 Confusion matrices and ranking metrics answer different questions

For a specified threshold, a confusion matrix counts true negatives, false positives, false negatives and true positives. State the orientation of rows and columns rather than assuming a plot's convention. Precision is the fraction of predicted positives that are truly positive; recall is the fraction of actual positives detected. Their denominators differ, so neither is a replacement for the other. Undefined ratios need an explicit reporting policy when a denominator is zero.

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 combines high detection with few false alarms. The diagonal joins equal TPR and FPR and is the reference for an uninformative ranking in expectation. A useful ROC comparison considers the region the application can actually use: two curves may cross, and the larger full-area score need not be better at a stringent false-positive constraint. An empirical ROC curve need not be concave.

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 tied score block must enter together. Drawing a straight segment between its endpoint rates gives half-credit for tied positive-negative comparisons. It does not assert that a deterministic threshold can separate identical scores. Vertical segments have zero horizontal width and contribute no area; horizontal segments contribute their height times their 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 transformation preserves ROC AUC because every greater-than comparison and every tie remains unchanged. Such a transformation can nevertheless destroy probability calibration. AUC is therefore neither accuracy, confidence quality nor a prescription for a threshold. The proposition below proves the ranking identity directly from 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

Observation	True class	Score
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

Notation for Figure 3.3

t Decision threshold in this figure: a score at least t predicts positive. It is the same role as tau in the main threshold notation, not a time variable.

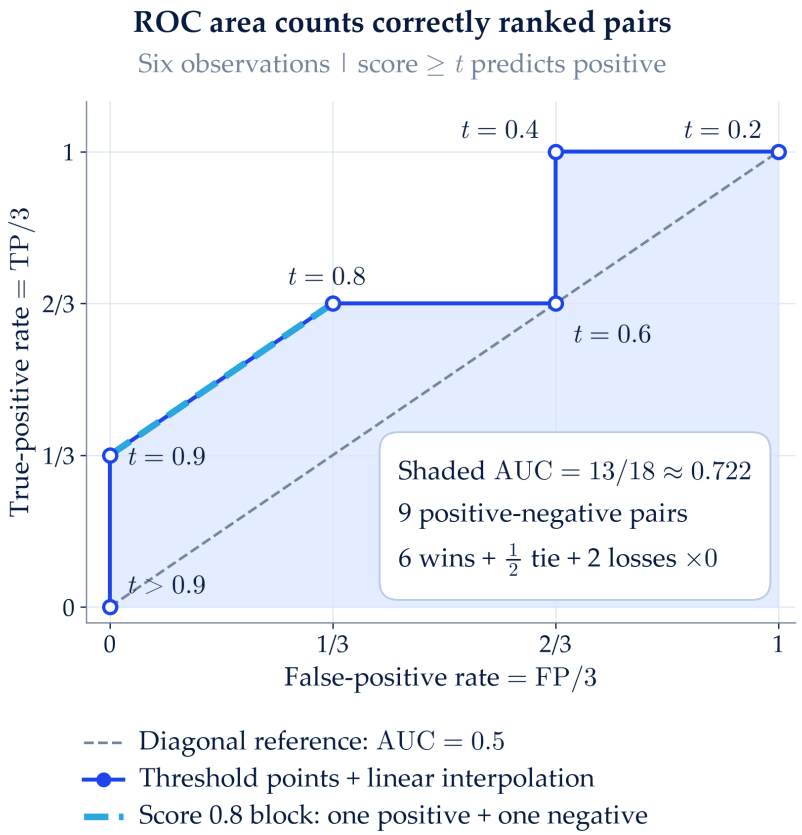


Figure 3.3. Exact ROC calculation for scores (0.9, 0.8, 0.8, 0.6, 0.4, 0.2) and labels (1, 0, 1, 0, 1, 0), using the decision score at least the threshold. Each marker is a realizable deterministic threshold; both observations tied at 0.8 enter together. The cyan segment linearly interpolates that tied block rather than breaking its tie by label. The shaded trapezoidal area is 13/18, approximately 0.722. Across the nine positive-negative pairs there are six strict wins, one tie and two losses, giving the same area with half-credit for the tie. The diagonal reference has area 0.5; no particular operating threshold is selected.

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.

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.
$\mathrm{precision}$	Positive predictive value, provided the predicted-positive probability is nonzero.

$$\mathrm{precision} = \frac{\pi\,\mathrm{TPR}}{\pi\,\mathrm{TPR} + (1-\pi)\,\mathrm{FPR}} \tag{3.19}$$

Bayes' rule exposes how precision depends on prevalence, even when both class-conditional rates remain fixed.

For example, TPR = 0.90 and FPR = 0.05 give precision about 0.947 when prevalence is 50%, but only about 0.154 when prevalence is 1%. Among 10,000 observations at 1% prevalence, the corresponding counts are 90 true positives, 10 false negatives, 495 false positives and 9,405 true negatives. A seemingly small FPR still produces many more false alerts than correct alerts. This comparison assumes unchanged class-conditional score behavior; changing the population can also alter that behavior and invalidate the assumption.

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.

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.

Complete supporting proof: Mathematical Supplement, A2 (in this book).

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

A high ROC AUC does not imply calibrated probabilities, adequate precision at the deployed threshold or a reliable estimate for an underrepresented subgroup.

Algorithm 3.1. Empirical ROC AUC with tied scores

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<=K<=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

01 $n_+ \leftarrow \sum_{i=1}^{n} y_i$

02 $n_- \leftarrow n - n_+$

03 **Require** $n_+ > 0,\quad n_- > 0$

04 **Group** $(\tau_k, G_k)_{k=1}^{K} \leftarrow \mathrm{Group}(s)$

05 $(A_0, B_0, F_0, T_0) \leftarrow (0,0,0,0)$

06 $U \leftarrow 0$

07 $Q \leftarrow [(+\infty, 0, 0)]$

08 **For** $k = 1, \ldots, K$

09 $\quad A_k \leftarrow A_{k-1} + \sum_{i \in G_k} y_i$

10 $\quad B_k \leftarrow B_{k-1} + \sum_{i \in G_k} (1 - y_i)$

Algorithm 3.1 (continued)

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.

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.

$$\text{cost}(\tau) = \frac{C_{FP}FP(\tau) + C_{FN}FN(\tau)}{n} \tag{3.21}$$

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

Condition on the available input before comparing actions. A positive action incurs a cost only when the target is negative, whereas a negative action incurs a cost only when the target is positive. Weight each possible cost by its conditional probability.

$$q = \Pr(Y = 1 \mid X = x) \tag{3.T29}$$

$$r_1(x) = C_{FP}(1 - q) \tag{3.T30}$$

$$r_0(x) = C_{FN}\, q \tag{3.T31}$$

$$r_1(x) < r_0(x) \iff C_{FP} < (C_{FP} + C_{FN})\, q \tag{3.T32}$$

$$q > \tau_* \tag{3.T33}$$

$$\tau_* = \frac{C_{FP}}{C_{FP} + C_{FN}} \tag{3.T34}$$

The denominator is positive by assumption, so the inequality direction is preserved. Selecting the smaller conditional risk at every input minimizes their average by the law of total expectation. This pointwise minimization is possible because there is no shared constraint such as a fixed daily inspection budget.

For example, a missed positive costing four times a false alarm gives a threshold of one fifth. An arbitrary ranking score cannot replace a probability. Agreement within broad score bins alone does not establish score calibration or justify the cost threshold. Exact score calibration means that, conditional on the reported probability score, the positive-class probability equals that score almost surely. Under the same cost assumptions, this supports the threshold among rules using only that score, not necessarily among rules using all available features.

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

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 tau_k.

$q(c) = \dfrac{C_{FP}FP + C_{FN}FN}{n}$ Mean empirical cost per validation observation; q_k stores this cost at threshold tau_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 $k = 1, \ldots, K$

  $a_k \leftarrow \sum_{i \in G_k} y_i$

  $b_k \leftarrow |G_k| - a_k$

  $c \leftarrow c + (a_k, b_k, -a_k, -b_k)$

  $Q \leftarrow \mathrm{Append}(Q, (\tau_k, c, q(c)))$

End for

Algorithm 3.2 (continued)

$k^* \leftarrow \min \arg \min_{0 \le k \le K} q_k$

$\tau^* \leftarrow \tau_{k^*}$

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.

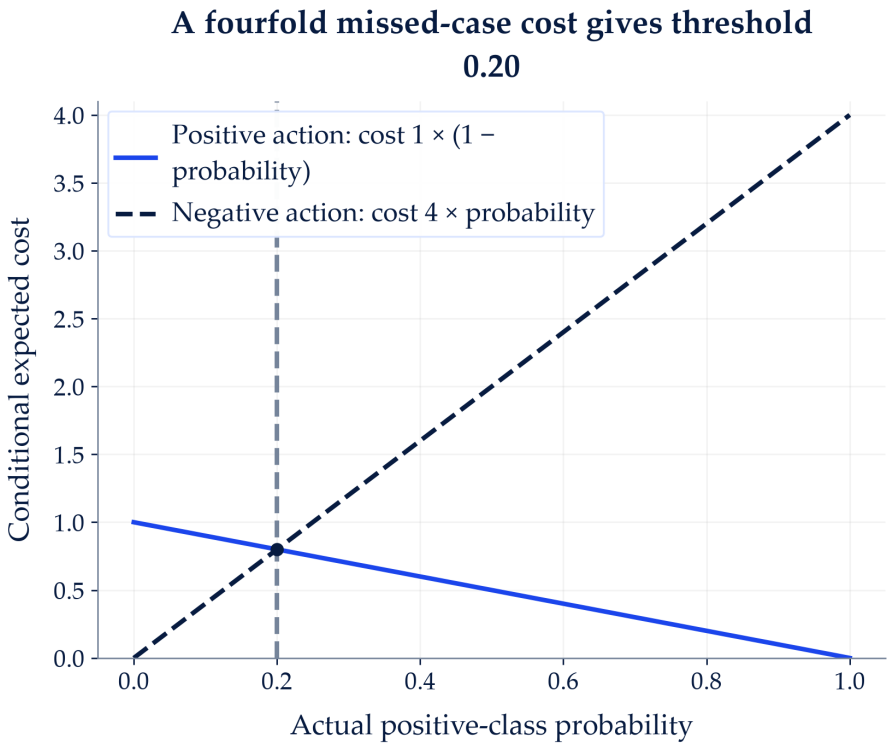


Figure 3.4. A positive action incurs false-positive cost 1; a negative action incurs false-negative cost 4; correct decisions cost zero. Conditional expected costs cross at positive-class probability 0.20. The threshold is optimal under these stated probabilities and constant costs, not under every deployment objective.

3.5 Handle imbalance without rewriting the evaluation population

Class imbalance describes unequal target frequencies, not automatically a data defect. Rare outcomes may be exactly what the application must predict. First establish a majority or class-frequency baseline, report prevalence and select metrics that expose the relevant errors. Also inspect absolute counts: one percent positives in a million examples is a different estimation problem from one percent positives in a hundred. Label quality and underrepresented subgroups can matter more than the overall ratio.

Class weighting changes the contribution of observations to the fitting objective. Oversampling repeats or generates minority-class training examples, while undersampling removes majority-class training examples. These interventions may improve a relevant decision metric, but they can increase variance, discard information or change probability interpretation. A weighted logistic model estimates an objective under altered class contributions; its raw probabilities should not automatically be read as probabilities under 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.

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**

$$r_w(p; q) = -a_1 q \log p - a_0(1 - q) \log(1 - p) \tag{3.T35}$$

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.

$$-\frac{a_1 q}{p} + \frac{a_0(1 - q)}{1 - p} = 0 \tag{3.T36}$$

$$p_w = \frac{a_1 q}{a_1 q + a_0(1 - q)} \tag{3.T37}$$

$$\frac{p_w}{1 - p_w} = \frac{a_1}{a_0} \frac{q}{1 - q} \tag{3.T38}$$

Both terms in the second derivative are positive, establishing the unique minimum. Solving the displayed odds relationship for the original probability gives the population inverse, useful for understanding what weighting changes.

$$q = \frac{a_0 p_w}{a_1(1 - p_w) + a_0 p_w} \tag{3.T39}$$

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.

$$\mathrm{PPV} = \frac{\pi t}{\pi t + (1-\pi) f} \tag{3.T40}$$

The numerator and denominator both depend on prevalence, so precision cannot be transported unchanged between differently sampled populations. This identity assumes that the class-conditional rates remain the same; a broader distribution shift can change those rates as well.

□

Weighting changes the fitted probability target; calibration checks forecasts; thresholds choose actions. Apply resampling only inside fitting folds and preserve evaluation prevalence. Verify probability corrections on representative independent data: finite-sample, regularization, model and optimization errors can prevent the population identity from holding exactly.

▶ 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.

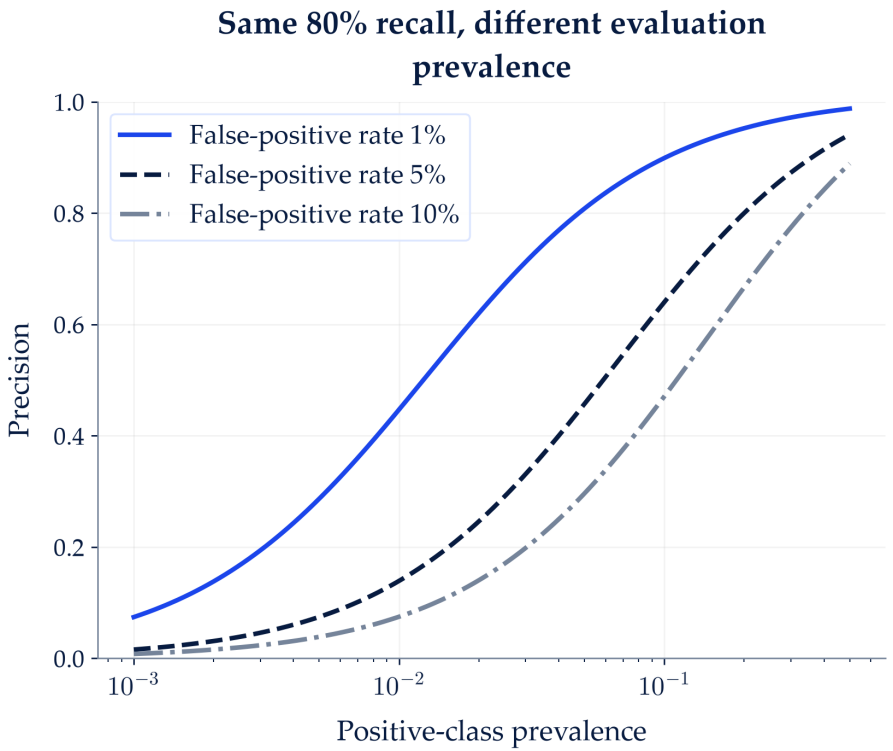


Figure 3.5. Bayes-rule precision as prevalence varies, with recall fixed at 0.80 and the indicated false-positive rate fixed for each curve. Precision decreases sharply for rare positives. Transporting these curves requires stable class-conditional detection and false-alarm rates.

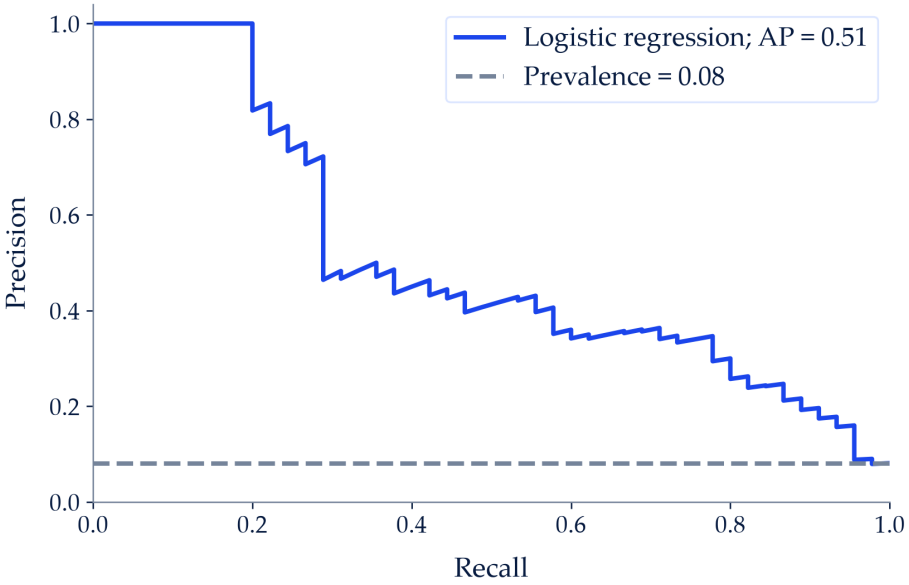


Figure 3.6. Actual output from a fixed synthetic, imbalanced classification experiment. Precision and recall change with the threshold. Average precision summarizes ranking but does not choose a threshold or encode the cost of a missed event.

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?	Values do not transfer unchanged after prevalence or population changes.
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.

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 traded additional false positives for fewer false negatives and lower declared cost. The ordinary model's final ROC AUC was 0.998 and log loss was 0.065; these summarize its scores, not a particular threshold.

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 original target value 0 means malignant; the case recodes it to positive label 1. Across the full collection, an always-benign rule would miss all 212 malignant records while correctly classifying 357 benign records. Its accuracy would be 357/569, about 0.627, despite detecting no malignant cases; this is descriptive arithmetic, 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.

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

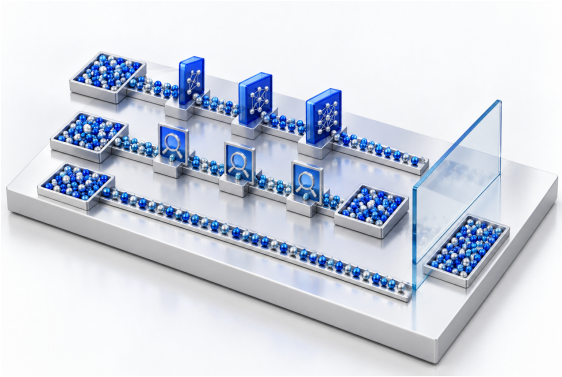
[3.7] Brier, G. W. (1950). Verification of Forecasts Expressed in Terms of Probability. Monthly Weather Review 78(1), 1-3. Source / DOI

[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

CHAPTER 04

Validation And Model Selection



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.

Core route: 4.1, 4.2, 4.3, 4.4, 4.5. Revisit the proofs after reproducing the Examples.

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 is a sequence of choices, not just a fitted estimator. Features, preprocessing, algorithms, regularization, thresholds and stopping rules can all adapt to observed results. Evaluation is credible only when its data boundaries account for that entire sequence. A carefully tuned model with an inappropriate split can produce less useful evidence than a simple baseline evaluated against the actual intended use.

This chapter organizes model development as an experiment that another person can inspect and repeat. We distinguish several cross-validation designs, show how leakage can create apparent signal from random labels, and separate inner tuning from outer evaluation. The goal is not to declare one split universally correct. It is to make explicit which population and which learning procedure each reported score estimates, and what uncertainty remains.

4.1 Holdout, K-fold, stratification and leave-one-out

A holdout split reserves observations for evaluation while fitting on the remainder. It is simple and often adequate when the evaluation sample is representative and sufficiently informative. Its result depends on the chosen split, and a small minority class can leave too few evaluation positives for a reliable metric. The appropriate split size depends on data availability, dependence, decision stakes and the precision required; fixed percentages are examples rather than 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 uses one evaluation observation per fold, creating n fits for n development examples. It can be computationally expensive and provides

very small per-fold evaluation sets; individual areas under the receiver operating characteristic curve (ROC AUC values) are not defined on one binary observation. Its heavily overlapping training sets also create dependent results. Leave-one-out is neither automatically best nor universally unbiased for every deployed procedure. Use it only when the estimator, metric, data size and dependence structure justify the design.

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.

$$\widehat{R}_{\mathrm{CV}} = \frac{1}{V} \sum_{v=1}^{V} \widehat{R}_{D_v}\left(\hat{f}_{D \setminus D_v}\right) \tag{4.1}$$

For mean per-observation loss, equal folds need equal weights. With fold sizes two and eight and mean losses zero and one, the unweighted mean is 0.5 but pooled loss is 0.8. Nonlinear metrics need not obey even this size-weighted identity.

This average weights folds equally. Unequal fold sizes or nonlinear metrics can differ from a pooled out-of-fold score.

◇ Theorem 4.1: What an independent holdout estimates

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 loss variance is needed 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.

The holdout mean loss is conditionally unbiased for the fitted predictor's population risk. With bounded losses, its deviation has an explicit finite-sample probability bound. Independence is an assumption about information flow, not merely different row identifiers.

□ **Proof**

$$Z_i = L(Y_i, f(X_i)) \tag{4.T1}$$

$$R(f) = \mathbb{E}[Z_i \mid f] \tag{4.T2}$$

$$\widehat{R} = \frac{1}{m} \sum_{i=1}^{m} Z_i \tag{4.T3}$$

$$\mathbb{E}[\widehat{R} \mid f] = R(f) \tag{4.T4}$$

$$\mathrm{Var}(\widehat{R} \mid f) = \frac{\mathrm{Var}(Z_1 \mid f)}{m} \tag{4.T5}$$

Linearity of expectation proves unbiasedness. Independence makes all covariance terms zero, proving the variance identity. To obtain a probability bound for bounded losses, first bound their centered moment-generating function. The second derivative of its logarithm is the variance under an exponentially tilted distribution. Tilting preserves the interval from zero to one, and any variable in that interval has variance at most one quarter.

$$g(s) = \log \mathbb{E}[e^{s(Z - \mathbb{E}Z)}] \tag{4.T6}$$

$$g(0) = g'(0) = 0 \tag{4.T7}$$

$$\mathbb{E}_s[h(Z)] = \frac{\mathbb{E}[h(Z)e^{sZ}]}{\mathbb{E}[e^{sZ}]} \tag{4.T8}$$

$$g''(s) = \mathbb{E}_s[Z^2] - (\mathbb{E}_s[Z])^2 \leq \tfrac{1}{4} \tag{4.T9}$$

The variance bound follows because variance is the minimum mean squared distance to a constant, whereas squared distance to one half is at most one quarter. Integrating the second-derivative bound twice from zero gives a quadratic upper bound for the log moment-generating function. Markov's inequality and independence then bound the upper tail of the sum.

$$g(s) \leq \frac{s^2}{8} \tag{4.T10}$$

$$\Pr(\widehat{R} - R(f) \geq \epsilon \mid f) \leq e^{-sm\epsilon + ms^2/8} \quad (s > 0) \tag{4.T11}$$

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.

$$\Pr(|\widehat{R} - R(f)| \geq \epsilon \mid f) \leq 2e^{-2m\epsilon^2} \tag{4.T12}$$

For a finite family fixed without these evaluation observations, apply the same bound to each candidate and use the union bound. Independence between

candidate scores is not required. A family repeatedly invented using the same holdout cannot automatically be treated as fixed.

$$\Pr\left(\max_{1\le j\le M} |\widehat{R}_j - R(f_j)| \ge \epsilon\right) \le 2M e^{-2m\epsilon^2} \tag{4.T13}$$

□

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.

Complete supporting proof: Mathematical Supplement, A3 (in this book).

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.

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.

Rows are often related through people, organizations, devices, experiments or source documents. If the use case involves new groups, validation must withhold complete groups rather than individual rows. Group-aware splitting prevents the same group identifier from appearing in fitting and evaluation roles, although other relationships may still require attention. A model evaluated on known users can be useful, but its score should not be advertised as performance on entirely new users.

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.

A gap can separate the last training observation from the first evaluation prediction, but its size needs a reason. For targets measured over a horizon, the information needed to construct every training label must exist before the evaluation prediction time. Overlapping windows, delayed reporting and multiple related entities can require combined restrictions. Evaluate the deployable procedure, including its data availability and update schedule, rather than assuming that calling a temporal splitter certifies the entire pipeline.

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.

4.3 The validation boundary surrounds the entire pipeline

Leakage occurs when information unavailable to a valid training procedure influences the fitted predictor or a development choice. It can enter without explicit access to a validation target during the estimator's fit call. Examples include scaling on all observations, selecting variables by their full-dataset association with the target, learning a vocabulary before splitting, or constructing an aggregate that uses future outcomes. A clean estimator interface does not guarantee a clean experiment.

Pass the complete pipeline to cross-validation or search, not a matrix transformed using all rows. Each training fold must fit its own imputer, scaler, dimensionality reduction and estimator. Feature selection and partial least squares (PLS) also use targets, making their fitting boundary especially important. Audit this with a held-out perturbation: changing only validation values or labels must not alter that fold's fitted preprocessing. The pipeline still cannot identify future-outcome columns or decide which related rows must be grouped.

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.

$$\hat{T}_k = \mathrm{fitPreprocessor}(D_{-k}) \tag{4.4}$$

$$\hat{f}_k = \mathrm{fitModel}(\hat{T}_k(X_{-k}), y_{-k}) \tag{4.5}$$

$$\hat{y}_k = \hat{f}_k(\hat{T}_k(X_k)) \tag{4.6}$$

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

Generate random features independent of a random binary target. Selecting the twenty features most associated with all labels lets evaluation labels choose the representation; cross-validation afterward cannot undo that access. Repeat selection using only each training fold to remove this shortcut. The notebook reports both scores and marks whole-dataset selection as an invalid negative control. A finite leakage-free score need not equal chance exactly, nor

must every leaky run score higher. Diagnose leakage from the information path, not a favorable numerical ordering. Feature selection remains legitimate when it is fitted inside the same boundary as the estimator.

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.

4.4 Search selects a model; nested validation evaluates that selection

Hyperparameters describe choices outside the estimator's ordinary parameter fit: regularization strength, neighborhood size, tree depth and many others. Grid search evaluates a declared Cartesian product of candidates; randomized search samples from specified distributions and can allocate a fixed computational budget more flexibly. The search space is itself a modeling decision. Choose sensible scales, document the budget and compare alternatives on consistent splits rather than interpreting unequal tuning effort as an 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**

$$\widehat{R}_j = R_j + E_j \tag{4.T20}$$

$$\mathbb{E}[E_j] = 0 \tag{4.T21}$$

For each fixed candidate, the minimum observed score is no greater than that candidate's score. Take expectations and then minimize over the fixed candidate index. This establishes optimism without assuming independent candidate errors.

$$\mathbb{E}[\min_j \widehat{R}_j] \leq \min_j \mathbb{E}[\widehat{R}_j] = \min_j R_j \tag{4.T22}$$

For two equally good candidates, express the minimum of their score errors using their sum and absolute difference. The sum has zero expectation. 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**

$$\widehat{h} = B(D_{\text{in}}) \tag{4.T27}$$

$$\hat{f} = A(D_{\text{in}}, \widehat{h}) \tag{4.T28}$$

Condition on all outer-training records and all randomness used by selection and refitting. The selected configuration and fitted predictor are now fixed. No outer evaluation labels have been consulted, so those records remain fresh samples for this fixed predictor. Apply the independent-holdout expectation argument.

$$\mathbb{E}[\widehat{R}_{D_{\text{out}}}(\hat{f}) \mid D_{\text{in}}, U] = R(\hat{f}) \tag{4.T29}$$

Averaging over outer training samples evaluates the selected-and-refitted procedure at that outer training size. If feature selection instead inspects outer labels, the fitted predictor is no longer fixed independently of the outer outcomes, and the conditional expectation step fails. A pipeline object enforces only the operations actually placed inside it; it cannot correct a target-derived raw feature prepared beforehand.

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

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 outer-training/evaluation index sets and finite outer-split count.
$(I_{kv}, V_{kv}), \quad v = 1, \dots, m_k$	Inner-training/validation indices contained in T_k, and finite inner-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 group/time restrictions and metric class support at both levels. Handle failed fits and undefined metrics by the predeclared policy; never silently omit them. 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.

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

For $k = 1, \ldots, K$

  Split $(I_{kv},\ V_{kv})_{v=1}^{m_k} \leftarrow \mathrm{InnerSplit}(T_k)$

  For $(h, v) \in \mathcal{H} \times \{1, \ldots, m_k\}$

    $f_{khv} \leftarrow \mathrm{Fit}_h(D_{I_{kv}})$

    $q_{khv} \leftarrow M(y_{V_{kv}}, f_{khv}(X_{V_{kv}}))$

  End for

  Aggregate For each candidate h. $\bar{q}_{kh} \leftarrow \mathrm{Agg}_v(q_{khv})$

  $\widehat{h}_k \leftarrow \mathrm{Select}_{h \in \mathcal{H}}\ \bar{q}_{kh}$

  $\hat{f}_k \leftarrow \mathrm{Refit}_{\widehat{h}_k}(D_{T_k})$

  $\widehat{y}_k \leftarrow \hat{f}_k(X_{E_k})$

  $s_k \leftarrow M(y_{E_k}, \widehat{y}_k)$

Algorithm 4.1 (continued)

14 $\quad \mathcal{R}_k \leftarrow (E_k, \widehat{h}_k, s_k, \widehat{y}_k)$

15 **End for**

16 **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.

Outer evaluation: hold out one complete fold



The dark block never enters the inner search.
Outer scores assess the complete procedure.
Do not reuse them to select a new procedure.

Figure 4.1. Five-fold outer evaluation: each row holds out one complete dark test fold and runs the entire model-selection procedure on the remaining light training folds. The outer scores assess the selection procedure. Reusing them to redesign that procedure loses the untouched-evaluation boundary.

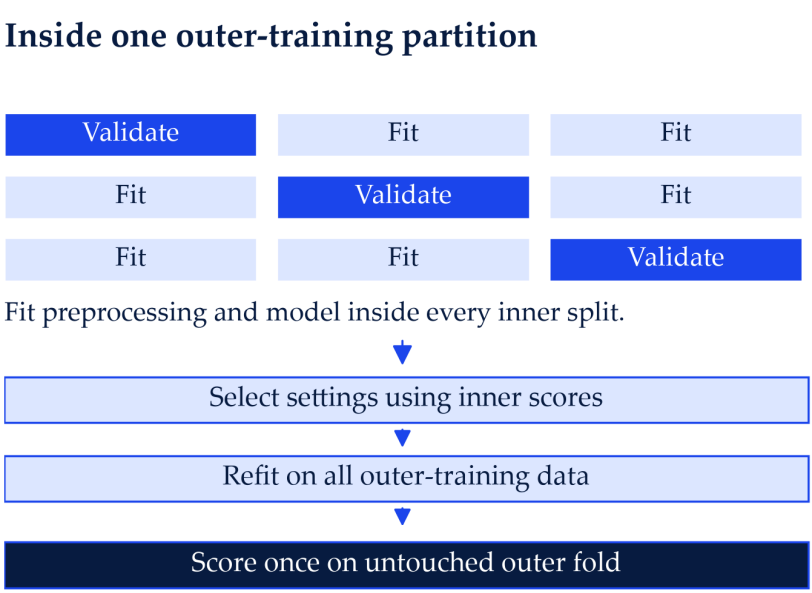


Figure 4.2. Inside one outer-training partition, three-fold inner validation selects settings. Every split fits preprocessing and the estimator using its own fitting rows only. The selected settings are refitted on all outer-training rows, then scored once on the untouched outer fold.

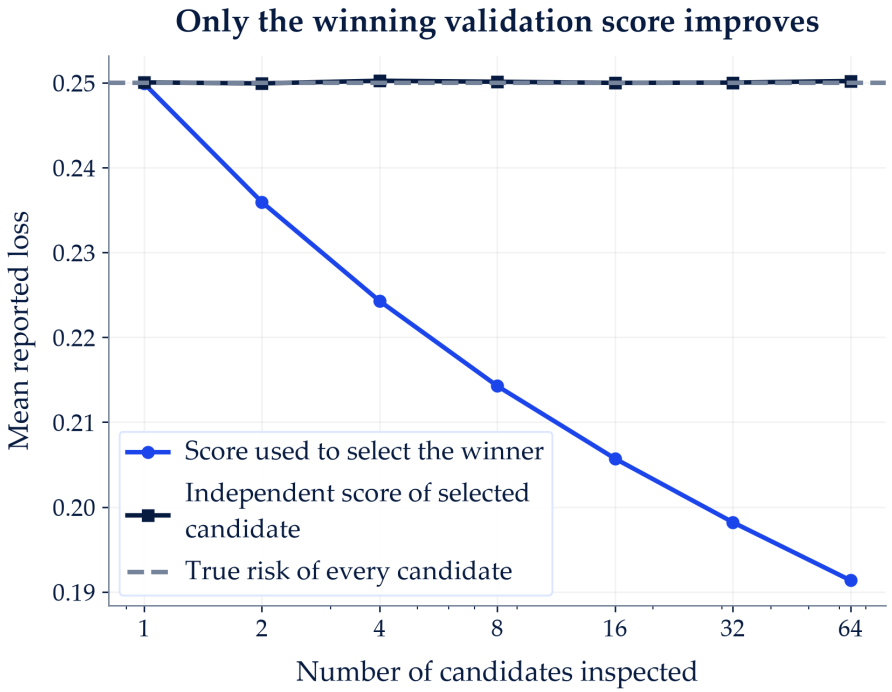


Figure 4.3. In 30,000 simulated experiments, every candidate has identical true loss 0.25 and independent Gaussian score noise with standard deviation 0.025. Searching more candidates lowers the winning validation score but not its independent evaluation score. Selection rewards favorable noise even when underlying quality is unchanged.

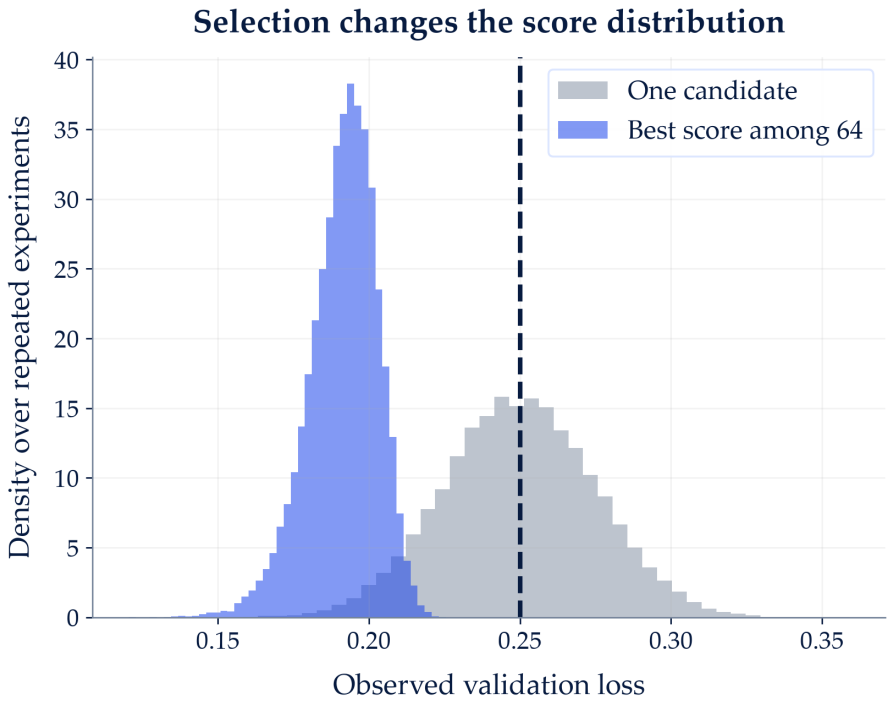


Figure 4.4. Score distributions for one candidate and the best validation score among 64 equally good candidates. Both come from the same 30,000-experiment score-noise simulation. The minimum is systematically optimistic. This controlled illustration does not claim that real candidate scores are independent.

4.5 Uncertainty, reproducibility and the final test lock

Distinguish uncertainty from finite evaluation samples, different training samples and stochastic fitting. Cross-validation folds share training observations, so their scores are generally dependent. Their standard deviation describes observed spread; dividing by the square root of the fold count ignores covariance unless independence is justified. Even identical fold scores need not mean zero uncertainty about future performance. Name the randomness being estimated and the assumptions connecting the reported spread to that uncertainty.

For a fixed model evaluated on independent test examples, resampling paired targets and predictions can provide a bootstrap interval for a chosen metric. This describes evaluation-sample variation conditional on that fitted model and observed population; it does not capture the full uncertainty from training, selection or future distribution changes. Grouped or temporal observations require resampling schemes that preserve the relevant dependence. Sparse outcomes can make intervals unstable or metrics undefined in some resamples.

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.

$$i_j \overset{\text{iid}}{\sim} \mathrm{Uniform}\{1,\dots,n\}, \qquad j = 1,\dots,n \tag{4.9}$$

$$\hat{s}^{(b)} = M\big((y_{i_j}, \hat{y}_{i_j})_{j=1}^{n}\big) \tag{4.10}$$

This paired bootstrap resamples independent evaluation examples for a fixed predictor. Percentiles of the resulting metric distribution are approximate intervals under stated assumptions, not guarantees under grouping, temporal dependence or shift.

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: What a bootstrap interval does and does not say

Freeze the classifier and development-selected settings, then obtain probabilities on an independent held-out test sample. Draw observation indices with replacement and use each selected index for both its label and probability; repeated indices remain repeated. Calculate ROC AUC only when both classes occur. The resulting bootstrap distribution approximates test-sample variability conditional on this fixed model, not variability from retraining, a different search or distribution shift. The notebook labels that scope and never selects a model or threshold from the interval. Count one-class resamples: their AUC is undefined. Dropping them conditions the retained distribution on both classes being present; if many are dropped, an ordinary unconditional percentile-interval interpretation is unjustified. A class-stratified bootstrap fixes the class counts and targets a different conditional sampling design. Keep the pairing, resampling unit and conditioning explicit before interpreting the 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.

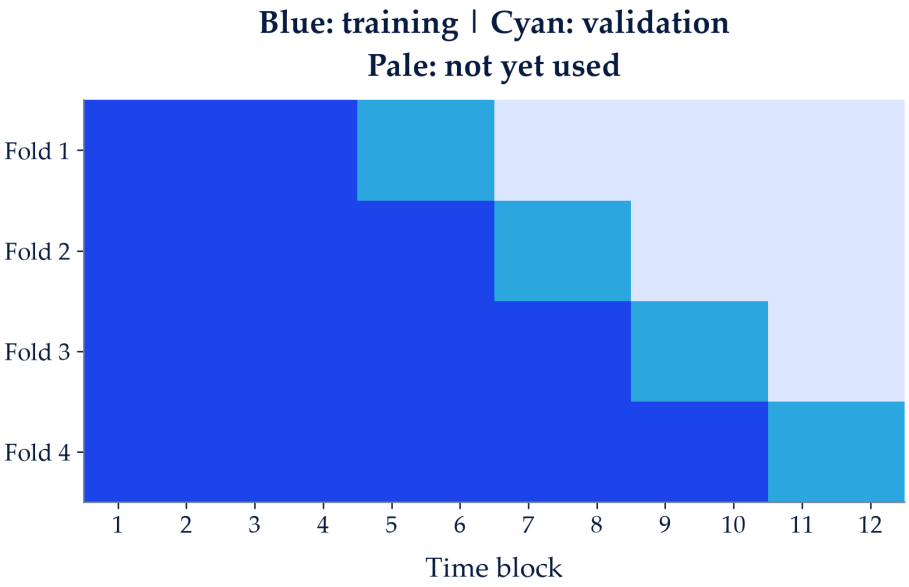


Figure 4.5. A schematic expanding-window split. Each validation block occurs after its training blocks. Real applications may also need a gap, grouped entities, and a separate final test period. Random row splitting answers a different question.

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.

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, but its northern test error was higher than its random-row error. The baseline RMSE moved in the opposite direction, underscoring that the test populations changed.

Failure Analysis And Limits

A random-row score understated the selected procedure's error on this northern holdout. This contrast is not a causal estimate of leakage: the protocols overlap in source records and differ in development geography and target distribution. Spatial-cell separation also does not make neighboring cells independent. The stored target is capped near $500,000, so the dollar errors describe neither uncapped prices nor a future housing 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.

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

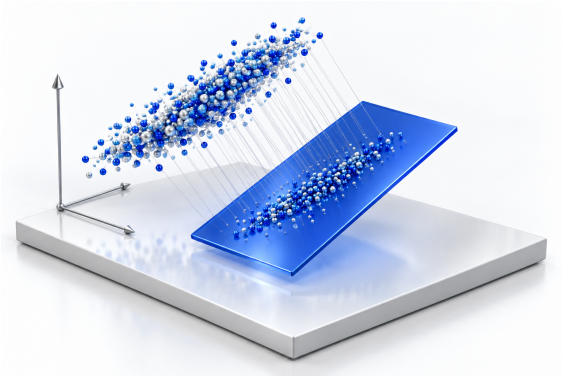
[4.7] Ambroise, C.; McLachlan, G. J. (2002). Selection bias in gene extraction on the basis of microarray gene-expression data. Proceedings of the National Academy of Sciences 99(10), 6562-6566. Source / DOI

[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

CHAPTER 05

Representations, PCA, And PLS



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.

Core route: 5.1, 5.2, 5.3, 5.4. Deepening route: 5.5.

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.

A predictive system rarely works directly with reality. It works with a representation: numerical coordinates constructed from measurements,

categories, text, images, or earlier models. Changing those coordinates can simplify optimization, expose structure, or remove information the target requires. Representation learning is therefore part of modeling, not a cosmetic preprocessing step. This chapter studies linear projections first because their assumptions and information losses can be written down explicitly.

We will connect correlated features to numerical conditioning, derive principal component analysis, and distinguish useful compression from useful prediction. Partial least squares adds target information to the representation; t-SNE serves a different purpose by making local neighborhoods visible. Throughout, learned transformations belong inside the training procedure. A representation fitted using held-out observations can leak their distribution, even when their target values are never consulted.

5.1 Representations, scale, and numerical conditioning

Let X contain n observations in rows and d measured features in columns. A representation maps an input vector to another coordinate system, sometimes with fewer dimensions. Standardization changes feature units; a rotation changes axes; a projection also discards directions. These operations answer different questions. Standardization is appropriate when relative deviations matter, but it may amplify an unreliable, low-variance measurement. The metric implied by preprocessing should match the task rather than follow a universal 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 is a mathematical summary, not automatically a hidden causal variable. Shared variation may reflect a common signal, a batch effect, measurement drift, or an unrecorded nuisance. Scores along covariance eigenvectors, as in PCA, are uncorrelated on the centered training sample; an arbitrary orthogonal change of axes need not decorrelate the data. Even uncorrelated scores need not be statistically independent, and the same covariance need not hold after deployment. Interpreting a coordinate requires

external evidence, repeatability checks, and comparison with known quantities rather than a compelling label attached after viewing 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 matrix has full column rank. Features and target use a fixed scaling convention; the intercept has been handled by centering. Only the centered target vector is perturbed by a zero-mean vector; the design and intercept convention are held fixed. The result concerns coefficient sensitivity, not a universal claim about prediction error.

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**

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.

$$X_c = U\Sigma V^\top \tag{5.T1}$$

$$\widehat{\beta} = V\Sigma^{-1}U^\top y_c \tag{5.T2}$$

The difference between solutions is a linear image of the perturbation. The largest singular value of the inverse map is the reciprocal of the smallest design singular value, giving the bound below.

$$\Delta\widehat{\beta} = V\Sigma^{-1}U^\top \delta \tag{5.T3}$$

$$\|\Delta\widehat{\beta}\|_2 \le \frac{\|\delta\|_2}{\sigma_d} \tag{5.T4}$$

Equality occurs when the perturbation follows the weakest left singular direction. Multiplying back by the design cancels the inverse singular values, so the fitted-value change is instead a projection. Thus unstable coefficients and stable in-sample fitted values can coexist.

$$\Delta\widehat{y} = X_c\Delta\widehat{\beta} = UU^\top\delta \tag{5.T5}$$

$$\|\Delta\widehat{y}\|_2 \le \|\delta\|_2 \tag{5.T6}$$

□

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.

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.

Principal component analysis chooses orthogonal directions that explain the greatest sample variance after centering. For a unit direction v, the projected variance is v transpose S v, where S is the sample covariance matrix. Introducing a multiplier for the unit-length constraint and setting the derivative to zero gives an eigenvector equation. The leading eigenvector maximizes variance; subsequent directions solve the same problem subject to orthogonality with earlier directions. Eigenvalues quantify variance, not semantic importance.

The same solution minimizes squared reconstruction error among rank-r orthogonal linear projections. Scores Z equal the centered observations multiplied by the retained loading directions. Multiplying the scores by the loading transpose maps back to the original feature space; adding the training mean completes reconstruction. This dual interpretation is useful: PCA both describes dominant variation and provides an optimal linear compression under a particular squared-error metric. It does not optimize classification accuracy, fairness, or target prediction.

In practice, compute PCA using an SVD of the centered matrix, avoiding an unnecessary covariance-matrix inverse. Component signs are arbitrary: negating a loading and its scores changes nothing about reconstruction. When eigenvalues are equal or nearly equal, individual axes can rotate substantially across samples even if their joint subspace remains stable. Compare subspaces or reconstruction quality before interpreting small changes in a loading plot as a real change in the underlying system.

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 Figure 5.1

$\mathrm{PC1}, \mathrm{PC2}, \mathrm{PC3}$	First, second, and third principal-component directions, ordered by decreasing sample variance; only the first two are retained.
A, B, C	Identifiers of three fixed observations tracked between panels; these labels do not denote matrices or random variables.

3D, 2D — Three-dimensional input coordinates and two-dimensional principal-score coordinates, respectively, in this synthetic projection example.

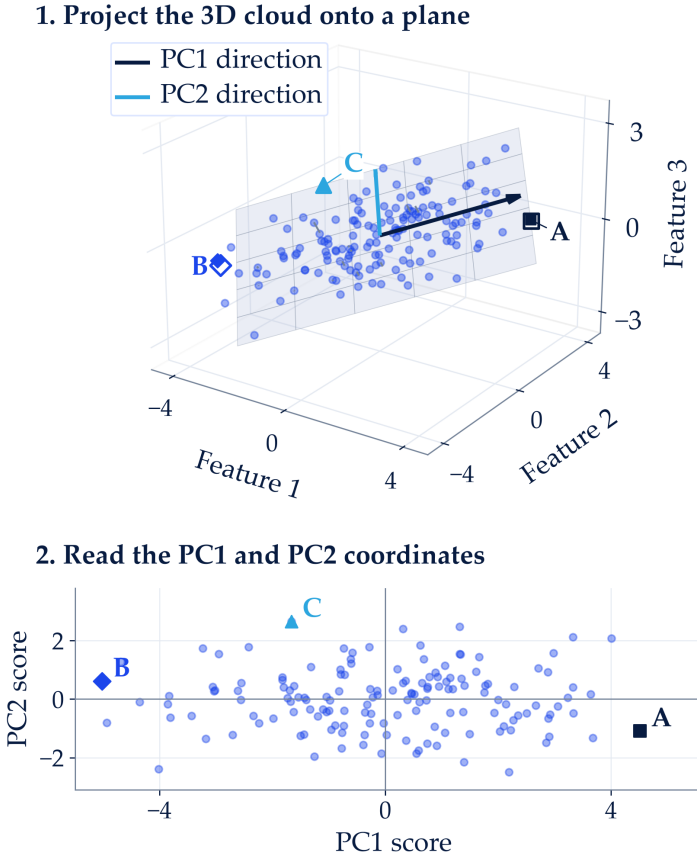


Figure 5.1. PCA fitted to 150 centered synthetic 3D observations (seed 202705). The pale plane is the two-dimensional principal subspace; short segments are orthogonal residuals in the discarded PC3 direction. For A, B and C, dashed segments connect filled observations to hollow projections on the plane. The navy and cyan arrows show the orthonormal PC1 and PC2 directions. The lower panel contains the same observations in PC-score coordinates; IDs, colors and marker shapes remain unchanged. The two scores retain 99.4% of sample variance. Squared reconstruction error equals the third squared singular value, checked numerically. No target is used.

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.

$$S = \frac{X_c^\top X_c}{n-1} \tag{5.3}$$

$$\max_{v^\top v=1} v^\top S v \quad \Longrightarrow \quad Sv = \ell v \tag{5.4}$$

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

$$S = \frac{X_c^\top X_c}{n-1} \tag{5.T7}$$

$$\mathrm{Var}(X_c v) = v^\top S v \tag{5.T8}$$

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.

$$\mathcal{L}(v, \ell) = v^\top S v - \ell(v^\top v - 1) \tag{5.T9}$$

$$\nabla_v \mathcal{L} = 2Sv - 2\ell v = 0, \qquad Sv = \ell v \tag{5.T10}$$

To distinguish maxima from the other stationary points, expand any unit direction in a complete orthonormal eigenbasis. Its variance is a weighted average of eigenvalues, with nonnegative weights summing to one. No weighted average can exceed the largest entry.

$$v = \sum_{j=1}^{d} a_j v_j, \qquad \sum_{j=1}^{d} a_j^2 = 1 \tag{5.T11}$$

$$v^\top S v = \sum_{j=1}^{d} \ell_j a_j^2 \le \ell_1 \tag{5.T12}$$

The leading eigenvector attains equality. Restricting the candidate to be orthogonal to previously selected eigenvectors repeats the argument on the remaining subspace. When eigenvalues tie, any orthonormal basis of the tied eigenspace is equally optimal; a particular axis is not identified.

□

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**

$$P = W W^\top, \qquad W^\top W = I_r \tag{5.T13}$$

$$\begin{aligned} \|X_c - X_c P\|_F^2 = \mathrm{tr}(X_c^\top X_c) \\ - \mathrm{tr}(W^\top X_c^\top X_c W) \end{aligned} \tag{5.T14}$$

The identity follows by expanding the squared norm and using symmetry and idempotence of the projector. Reconstruction minimization is therefore equivalent to maximizing captured sum of squares.

$$\mathrm{tr}(W^\top X_c^\top X_c W) = \sum_{j=1}^{d} \sigma_j^2 h_j, \quad 0 \le h_j \le 1, \quad \sum_{j=1}^{d} h_j = r \tag{5.T15}$$

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.

$$\widehat{X}_c = X_c V_r V_r^\top \tag{5.T16}$$

$$\|X_c - \widehat{X}_c\|_F^2 = \sum_{j=r+1}^{d} \sigma_j^2 \tag{5.T17}$$

For the stronger claim, take any candidate matrix of rank at most the retained dimension and let the next projector span its row space. Orthogonal row-space components give a Pythagorean decomposition. The first term cannot beat the best projection, while the second is nonnegative.

$$\|X_c - A\|_F^2 = \|X_c(I - P_A)\|_F^2 + \|X_c P_A - A\|_F^2 \tag{5.T18}$$

□

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

Take a centered sample covariance with diagonal entries 4 and off-diagonal entries 3. Multiplying by (1, 1) gives (7, 7); multiplying by (1, -1) gives (1, -1). Thus the eigenvalues are 7 and 1; divide each direction by the square root of two for unit loadings. Total variance is 8. Keeping the first component retains 7/8, or 87.5 percent, and discards 12.5 percent. This is optimal rank-one reconstruction in these units, not evidence that the discarded difference is irrelevant to the target.

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

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, d-by-r loading matrix, 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.

$$(\mathcal{T}, \mu, V_r, \Sigma, Z)$$

01 **Require** Valid dimensions, finite entries and integer retained count.

02 Training rows only. $\mathcal{T} \leftarrow \mathrm{FitScale}(X)$

03 $\widetilde{X} \leftarrow \mathcal{T}(X)$

04 $\mu \leftarrow n^{-1} \widetilde{X}^\top \mathbf{1}_n$

05 $X_c \leftarrow \widetilde{X} - \mathbf{1}_n \mu^\top$

06 **Factor** Sorted thin singular-value decomposition. $X_c = U \Sigma V^\top$

07 $V_r \leftarrow V_{[:,1:r]}$

08 $Z \leftarrow X_c V_r$

09 **Define** Frozen new-input map. $z(x) = V_r^\top (\mathcal{T}(x) - \mu)$

In scaled coordinates, for the residual check below.

10 $\widehat{X} \leftarrow Z V_r^\top + \mathbf{1}_n \mu^\top$

11 **Check** Within declared numerical tolerance. $V_r^\top V_r = I_r$

Algorithm 5.1 (continued)

12 **Check** $\|\widetilde{X} - \widehat{X}\|_F^2 = \sum_{j=r+1}^{q} \sigma_j^2$

13 **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.

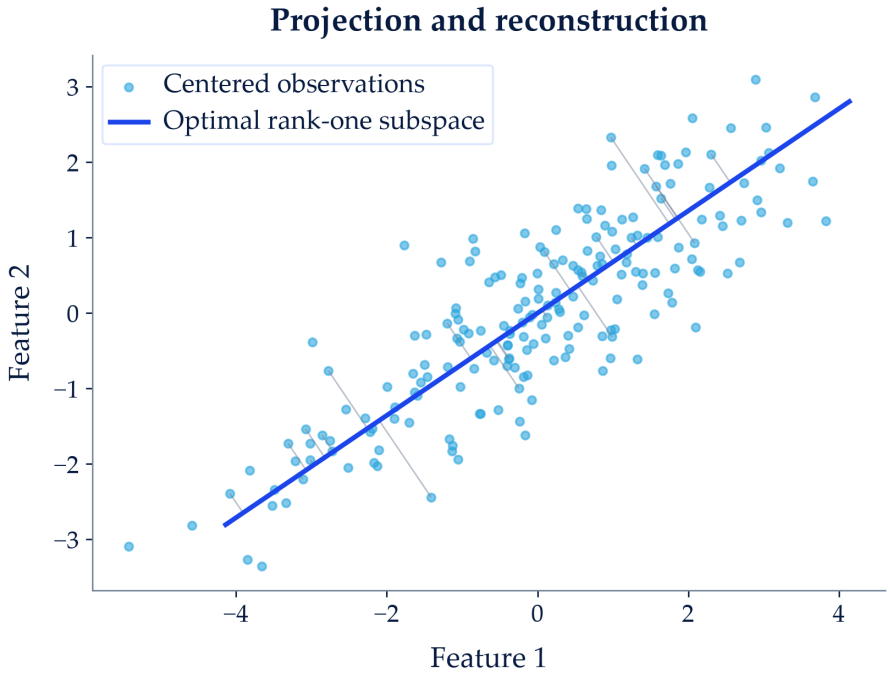


Figure 5.2. Rank-one PCA on 200 centered two-dimensional observations. The line is the first principal subspace; thin segments show discarded orthogonal residuals. The squared reconstruction residual equals the discarded squared singular value, checked numerically. This small example isolates the reconstruction geometry.

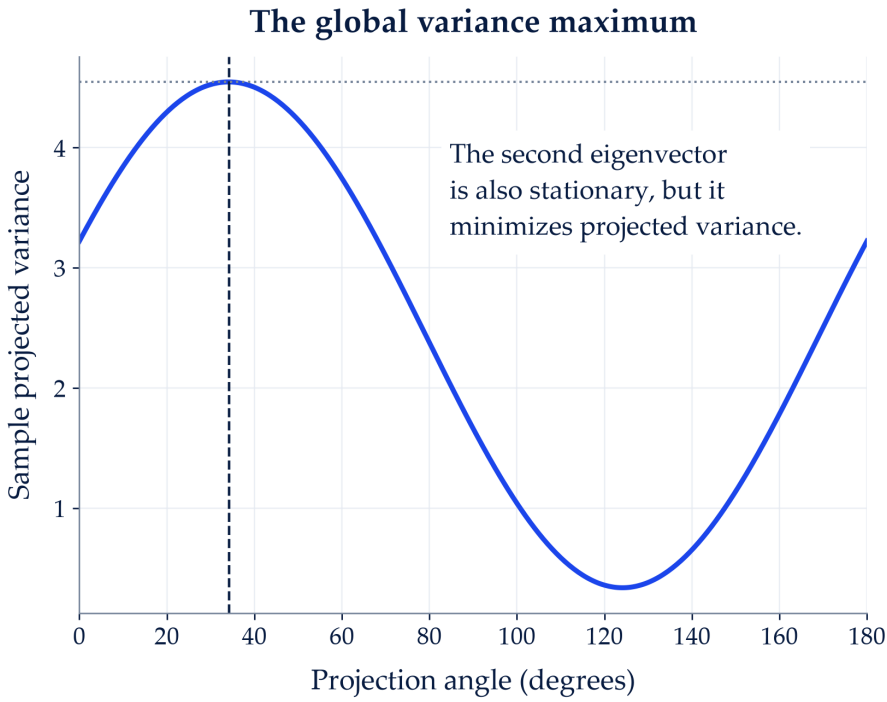


Figure 5.3. Projected sample variance is evaluated over a dense grid of direction angles for the same centered observations. Analytically, its global maximum is the largest covariance eigenvalue and occurs at the principal direction. The orthogonal second eigenvector is also stationary but minimizes, rather than maximizes, projected variance.

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 dimension against the intended loss. For compression, a scree plot or variance threshold proposes a rank; held-out reconstruction error, storage cost, and feature-specific distortion assess it. For prediction, select rank using the downstream validation metric. The same rank need not serve both purposes: a small discarded direction may contain the target signal.

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.

Low-variance signal is a particularly instructive negative case. A feature may vary very little yet track the target accurately, while a high-variance feature is irrelevant. Without scaling, PCA may discard the useful feature; after scaling, dominant correlated nuisance groups may still occupy many leading components. Representation selection therefore requires a baseline that retains all inputs. Ridge regression is often a strong comparator because it continuously shrinks uncertain directions rather than deleting every direction after a hard cutoff.

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.

5.4 Partial least squares uses target information

Partial least squares constructs latent directions using both inputs and targets. In a common first-component formulation, it seeks normalized input and output directions whose projected scores have large covariance. For a single target, the output direction is scalar; with multiple targets, it combines outputs as well. Covariance depends on units, so centering and scaling choices are part of the model. Unlike PCA, this representation is explicitly supervised and may prioritize a lower-variance direction connected to the target.

PLS extracts a component and deflates residual matrices before the next extraction. Name the implementation, normalization, and deflation settings:

variants differ. Distinguish scores (projected observations), weights (projection directions), loadings (reconstruction coefficients), and final regression coefficients. Maximizing first-component covariance explains an initial direction; repeating that calculation on unchanged matrices is not the complete PLS algorithm.

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.
$\|\cdot\|$	Euclidean norm; both weight vectors have unit length.
$\max$	Joint optimization of input/output weights under the displayed normalizations.

$$\max_{\|w\|=\|c\|=1} \operatorname{Cov}(X_c w,\, Y_c c)^2 \tag{5.9}$$

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.

Complete supporting proof: Mathematical Supplement, A4 (in this book).

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.

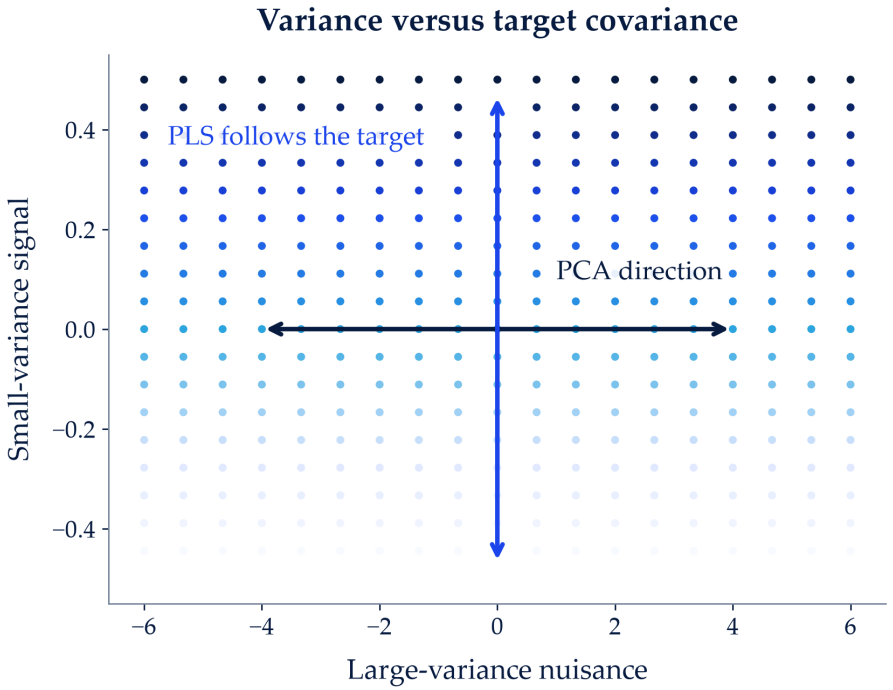


Figure 5.4. A balanced input grid makes the large-variance nuisance coordinate orthogonal to the small-variance target-bearing coordinate. PCA follows the nuisance axis; the normalized input-target cross-product identifies the first PLS direction. Colors encode the constructed target. This is an algebraic counterexample, not a benchmark.

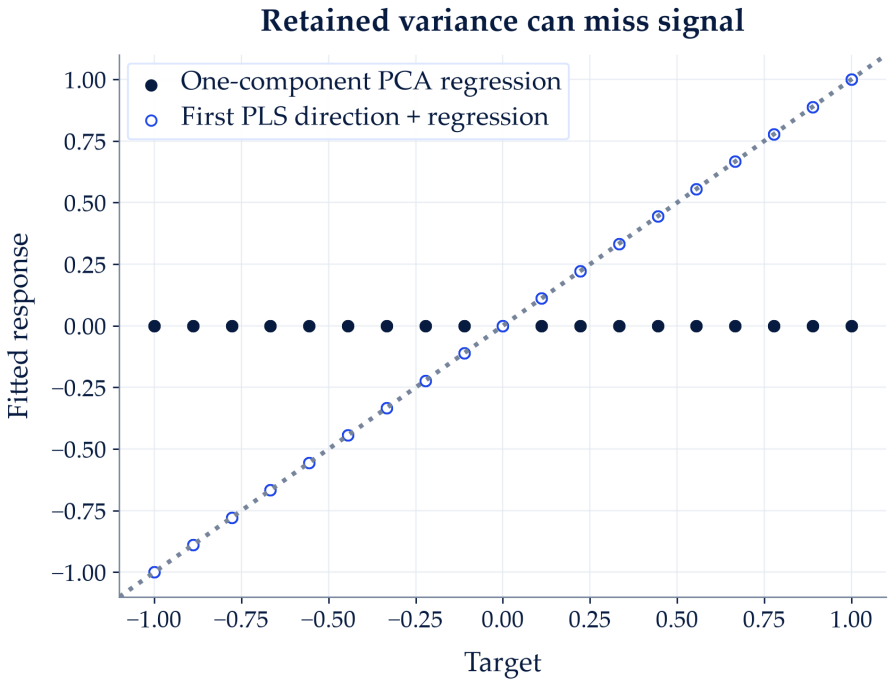


Figure 5.5. One-component PCA retains 99.3% of input variance in the orthogonal-grid example yet discards its target signal. Regression on the first PLS direction recovers the target exactly on this constructed grid. Fitted responses illustrate the different objectives; they do not estimate generalization performance.

PLS1 From Residual Directions To New Predictions

The first covariance direction is not yet a multi-component regressor. We now specify one complete variant: single-response PLS with unit input weights and regression deflation, the PLS1 specialization of NIPALS. NIPALS means nonlinear iterative partial least squares; with one response, each residual cross-covariance is a vector, so its direction is obtained directly rather than by an iterative singular-vector calculation.

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.

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.

$$q_h = \frac{t_h^T f_{h-1}}{t_h^T t_h} \tag{5.B4}$$

Subtract both fitted rank-one contributions before extracting the next component. These are regression-deflation updates: the target is deflated using the input score, not a separate target score.

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 $T = [t_1, \dots, t_r]$, 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 deflation makes the current score orthogonal to the remaining input and target residuals. Later scores are therefore orthogonal to earlier scores. It also removes the residual projection on the current weight; this remains zero after later deflations. Thus the final residual obeys $E_r W = 0$. Multiplying the decomposition $E_0 = TP^T + E_r$ 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

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.

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

5.5 t-SNE is a neighborhood visualization

t-distributed stochastic neighbor embedding turns high-dimensional similarities into probabilities and searches for a low-dimensional arrangement with compatible neighborhoods. A heavy-tailed distribution in the embedding helps separate nearby groups without forcing every high-dimensional relationship into a crowded center. Unlike PCA, t-SNE is nonlinear and does not provide an ordered sequence of variance-explaining components. Its objective emphasizes local relationships, so distances between remote islands and the sizes of visual gaps are not generally faithful measurements of original-space separation.

Perplexity controls neighborhood scale, not cluster count. Record initialization, optimization settings, seed, and sample composition; vary defensible settings and compare preserved neighbors rather than visual attractiveness. Trustworthiness measures one aspect of local preservation, not scientific validity. Keep coloring labels out of unsupervised fitting and disclose any selection based on those labels.

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.

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

Suppose a t-SNE display shows three islands. For one observation, six of its ten original-space nearest neighbors remain among its ten display-space neighbors: the overlap is 6/10, or 60 percent, not a trustworthiness score. Repeat this check across observations, seeds, and neighborhood scales. Inspect whether acquisition batches explain the islands and whether the original coordinates support them. A continuous gradient can fragment, and sampling can hide a rare group. Report settings and use the display to propose a test, such as a subgroup error comparison, rather than count confirmed populations.

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
PCR, six components	0.545	0.697
PLS, two components	0.537	0.687

At equal component count, PLS predicted better than PCR in this run. All-feature ridge retained the lowest test RMSE. The independently checked PCA reconstruction identity concerns input geometry, not this supervised 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

The first red-wine record has alcohol 9.4, pH 3.51 and observed quality 5. Under an explicitly hypothetical prediction of 6, the absolute score error is 1 and squared score error is 1. This illustrates the chosen loss on a real observation; it is not a measured output of any fitted model.

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.

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

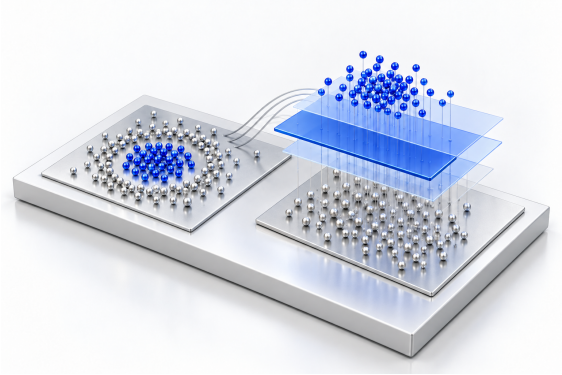
[5.7] Tipping, M. E.; Bishop, C. M. (1999). Probabilistic Principal Component Analysis. Journal of the Royal Statistical Society: Series B (Statistical Methodology) 61(3), 611-622. Source / DOI

[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

CHAPTER 06

Distance, Margins, And Kernels



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.

Core route: 6.1, 6.2, 6.4, 6.5. Deepening route: 6.3.

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.

Many learning methods begin with a notion of similarity. Nearest neighbors predicts from nearby observations; support vector machines learn a boundary with a controlled margin; kernels replace ordinary inner products with a richer similarity. These ideas connect geometry to prediction, but their

behavior depends on representation, scale, and the data available near each query. A geometrically elegant model can still fail under distribution shift or a poorly chosen metric.

This chapter develops both the mathematical structure and the practical workflow. We move from local averaging to maximum-margin classification, derive the role of the dual variables, and then introduce nonlinear kernels and support-vector regression. Model selection always includes preprocessing and hyperparameters inside validation. A margin is a distance in model coordinates, not a physical safety certificate, and a classifier score is not automatically a calibrated probability.

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.

The k-nearest-neighbors rule stores training examples and predicts from the k examples closest to a query. Regression averages their targets; classification uses their class frequencies or a weighted vote. Small k permits highly local behavior but is sensitive to noise. Larger k smooths predictions and can cross meaningful boundaries. Choose k using validation rather than intuition alone, and consider distance weighting when near and far members of a neighborhood should contribute differently.

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

The query-to-A coordinate differences are (1, 0.1); the query-to-B differences are (0.2, 0.8). Squared distances are 1 + 0.01 = 1.01 and 0.04 + 0.64 = 0.68, so B is

nearer. Multiplying the second coordinate by ten changes these sums to 2 and 64.04. Their square roots are approximately 1.414 and 8.002, making A nearer. The observations have not changed, but the metric has. Fit a domain-appropriate scaling rule within training data and validate the complete neighbor procedure.

Interpretation and limits

Nearest-neighbor probabilities are local class frequencies, not guaranteed calibrated probabilities. Sparse neighborhoods, weighting, and class imbalance can make them unreliable.

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 binary labels encoded as minus one and plus one, a linear classifier uses the sign of w transpose x plus b. Multiplying w and b by a positive constant leaves its decision boundary unchanged. For nonzero w, the canonical score planes have scores minus one and plus one, separated by distance two divided by the norm of w. A hard-margin SVM minimizes the squared coefficient norm while requiring every labeled observation to lie on or beyond its corresponding 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.

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i \\ \text{subject to} \quad & y_i(w^\top x_i + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0 \quad (i = 1, \dots, n). \end{aligned} \tag{6.3}$$

With C strictly positive, optimal slack equals the smallest required margin violation: zero on or beyond the correct margin, between zero and one for a correctly classified point inside the margin, one on the decision boundary, and greater than one for a misclassified point. Arbitrary feasible slack can be larger. C multiplies summed violations in this convention.

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.
$\lVert \cdot \rVert_2$	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

A displacement along the unit normal changes a hyperplane score by its length times the coefficient norm. Consequently, moving from the negative unit-score plane to the positive unit-score plane requires the following distance.

$$f(x) = w^\top x + b \tag{6.T1}$$

$$\text{width} = \frac{2}{\|w\|_2} \tag{6.T2}$$

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.

$$\min_{w,b,\xi} \ \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{n} \xi_i \tag{6.T3}$$

$$y_i f(x_i) \geq 1 - \xi_i, \qquad \xi_i \geq 0 \tag{6.T4}$$

For fixed coefficients and intercept, feasible slack must exceed both zero and the signed-score shortfall. Because its objective coefficient is positive, the minimizing slack is the smaller feasible choice. Substitution proves equivalence to summed hinge loss.

$$\xi_i^* = \max\{0, 1 - y_i f(x_i)\} \tag{6.T5}$$

$$\min_{w,b} \ \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^{n} \max\{0, 1 - y_i f(x_i)\} \tag{6.T6}$$

□

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.

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.

Introduce multipliers alpha for the margin constraints and beta for nonnegative slack. Stationarity of the Lagrangian gives w as a weighted sum of labeled training vectors, the equality constraint that the signed alphas sum to zero, and alpha plus beta equal to C. Eliminating primal variables produces a concave quadratic maximization with box constraints on alpha. The kernel matrix is positive semidefinite, so the dual is concave but not necessarily strictly concave; dual coefficients need not be unique.

The Karush-Kuhn-Tucker conditions combine stationarity, primal feasibility, dual feasibility, and complementary slackness. For this convex problem, with the usual feasible conditions, they characterize optimality. An observation strictly beyond the correct margin has alpha zero. A support vector has nonzero alpha, but in a soft-margin model it need not lie exactly on a margin plane: it may sit inside the margin or be misclassified. Only a free support vector, with alpha strictly between zero and C, is guaranteed to lie on its canonical margin.

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.

Complete supporting proof: Mathematical Supplement, A5 (in this book).

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

$$\alpha_i[y_i f(x_i) - 1 + \xi_i] = 0 \tag{6.T13}$$

$$(C - \alpha_i)\xi_i = 0 \tag{6.T14}$$

For a strictly interior multiplier, the second product forces zero slack and the first forces unit signed score. At a zero multiplier, the second still forces zero slack, but feasibility permits a score equal to or larger than one. Thus a point on the margin may have a zero multiplier in a degenerate solution; the converse implication is not automatic.

$$0 < \alpha_i < C \implies \xi_i = 0, \quad y_i f(x_i) = 1 \tag{6.T15}$$

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.

$$b = y_i - w^\top x_i \qquad \text{for } 0 < \alpha_i < C \tag{6.T18}$$

□

These products are only the complementary-slackness part of KKT; they do not replace feasibility and stationarity. Recovering an intercept from every nonzero multiplier is justified for the separable hard-margin case, but is generally wrong for soft-margin fitting.

▶ 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.

Notation for Figure 6.1

C	Positive upper bound on each margin multiplier; its value is 0.8 in this fitted synthetic example.

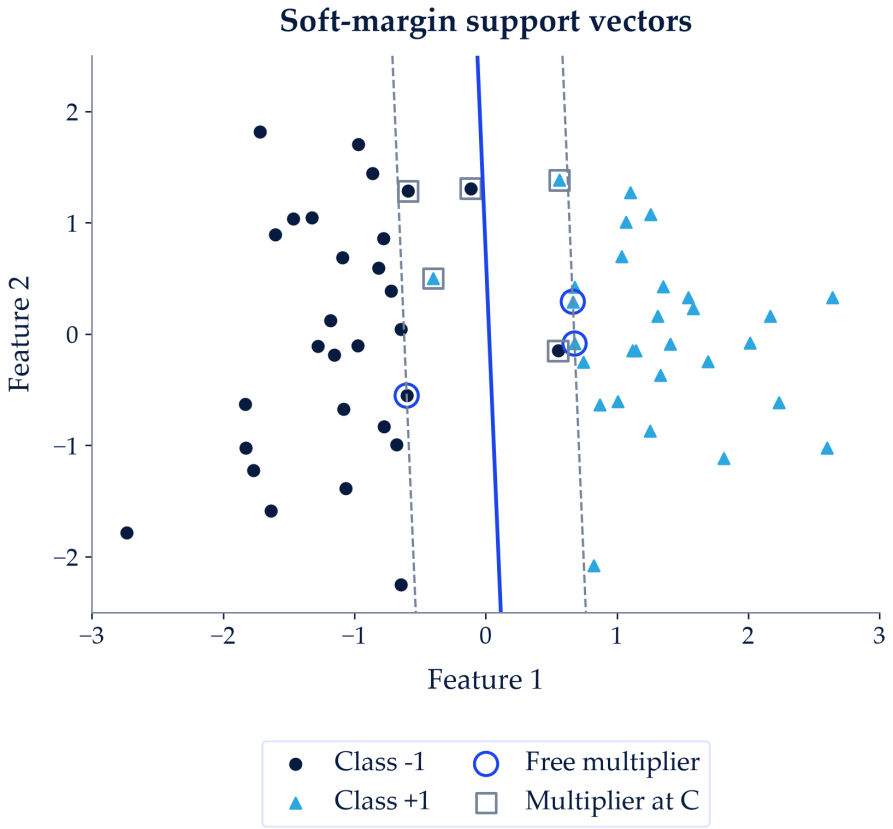


Figure 6.1. A fitted linear soft-margin SVM on two declared classes. Outlined circles identify multipliers strictly between zero and C; outlined squares identify multipliers at C. The former lie on unit margin, while the latter may violate it. C is 0.8 in this example.

Notation for Figure 6.2

C	Common positive multiplier upper bound, equal to the summed-slack penalty of 0.8 for these fitted observations.

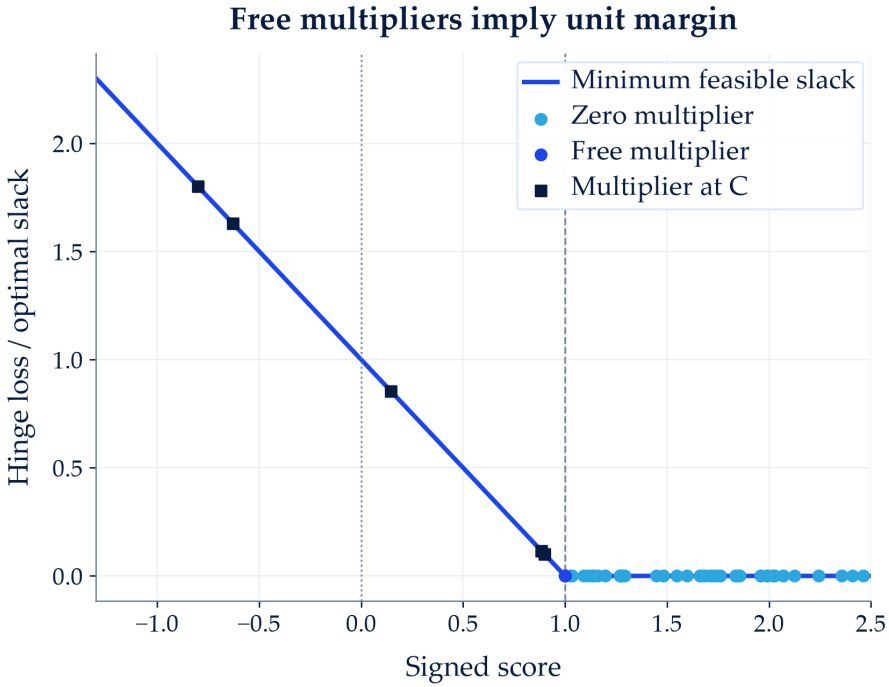


Figure 6.2. Every fitted observation is placed at its signed decision score and exact optimal slack, max(0, 1 minus signed score). Free multipliers occur at unit margin. Bounded and zero multipliers have different KKT roles. Numerical checks verify the free-support margin relation.

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 represents an inner product in a possibly high-dimensional feature space without requiring explicit construction of those features. Replacing the training inner products in the dual produces nonlinear decision boundaries in the original input coordinates. Not every similarity is a valid kernel: positive semidefinite Gram matrices are required for the usual convex SVM formulation. Common choices include linear, polynomial, and Gaussian radial basis function kernels. Kernel selection is a modeling assumption about what kinds of input relationships should count as similar.

The RBF kernel decays exponentially with squared distance. Its gamma parameter controls locality: small gamma produces broad influence, while large gamma makes influence concentrate near individual training observations. C and gamma interact. Large values of both can make highly detailed boundaries that fit noise; values that are too small can underfit. Search a logarithmic grid within a pipeline containing suitable scaling. Compare a linear baseline, because a flexible kernel adds computational expense without guaranteeing 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

$$K_{ij} = k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle \tag{6.T19}$$

$$a^\top K a = \left\| \sum_i a_i \phi(x_i) \right\|^2 \geq 0 \tag{6.T20}$$

This identity proves necessity. Conversely, a positive-semidefinite matrix on one sample admits an eigenvalue factorization and hence finite feature coordinates on that sample. A valid kernel requires consistency over every finite input set; one successful matrix test alone does not prove that global property.

$$e^{-\gamma\|x-z\|_2^2} = e^{-\gamma\|x\|_2^2} e^{-\gamma\|z\|_2^2} \cdot \sum_{m=0}^{\infty} \frac{(2\gamma)^m}{m!} (x^\top z)^m \tag{6.T21}$$

Each integer-power dot product is an inner product of tensor-product features and is positive semidefinite. Multiplying a kernel by the same scalar feature factor on each argument preserves this property. Nonnegative finite sums therefore remain positive semidefinite; entrywise convergence of the exponential series preserves every finite quadratic-form inequality in the limit. □

A familiar-looking similarity is not automatically a valid SVM kernel. Positive semidefiniteness secures the convex optimization structure; it does not establish that the implied geometry is useful for a particular prediction problem.

▶ 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

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<=tau and alpha>=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 $K_{ij} \leftarrow \kappa(x_i, x_j)$

04 $\alpha \leftarrow \mathrm{DualQP}(K, y, C; \varepsilon, T)$

05 **Require** Solver success, finite multipliers and feasible box/equality constraints.

06 Using the full multiplier vector. $g_i \leftarrow \sum_{j=1}^{n} \alpha_j y_j K_{ij}$

07 $I \leftarrow \{i : \tau < \alpha_i < C - \tau\}$

Algorithm 6.1 (continued)

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

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 $1, \ldots, K$. 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 lies strictly between minus one third and one third. Even two opposite corrections differ by less than two thirds, so they cannot reverse a one-vote advantage. They can resolve a vote tie. These scores need not sum to one and are not calibrated probabilities.

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

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 replaces the classification margin with an epsilon-insensitive tube around the predicted response. Errors whose absolute magnitude is at most epsilon receive zero loss; larger errors incur a linear penalty for the excess. The model balances this loss against the squared coefficient norm in its feature space. Epsilon is measured in target units. Rescaling the target without adjusting epsilon and other parameters changes the effective problem, so any target transformation must be documented and inverted for reporting.

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.

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

$$\min_{w,b,\xi,\xi^*} \frac{1}{2}\|w\|^2 + C\sum_i(\xi_i + \xi_i^*) \tag{6.T22}$$

$$y_i - f_i \le \epsilon + \xi_i, \qquad f_i - y_i \le \epsilon + \xi_i^* \tag{6.T23}$$

$$f_i = \langle w, \phi(x_i)\rangle + b \tag{6.T24}$$

Minimizing slacks at a fixed prediction gives the positive excess of absolute residual over the half-width. Assign separate multipliers to the two residual inequalities. Stationarity in the coefficient vector uses their difference; stationarity in the intercept forces those differences to sum to zero. Slack stationarity bounds both multiplier families by the penalty.

$$d_i = \alpha_i - \alpha_i^* \tag{6.T25}$$

$$w = \sum_i d_i \phi(x_i) \tag{6.T26}$$

$$\sum_i d_i = 0, \qquad 0 \le \alpha_i, \alpha_i^* \le C \tag{6.T27}$$

Eliminating primal variables leaves a concave quadratic objective with a target-weighted linear term and a tube-width penalty. This is structurally different from the binary-classification dual.

$$\max_{\alpha,\alpha^*} \sum_i y_i d_i - \epsilon \sum_i (\alpha_i + \alpha_i^*) - \frac{1}{2} \sum_{i,j} d_i d_j k(x_i, x_j) \tag{6.T28}$$

□

The tube is a no-penalty region of a training objective, not a confidence or prediction interval. Its width does not guarantee a future coverage fraction. The signed regression coefficients must not be mistaken for 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.

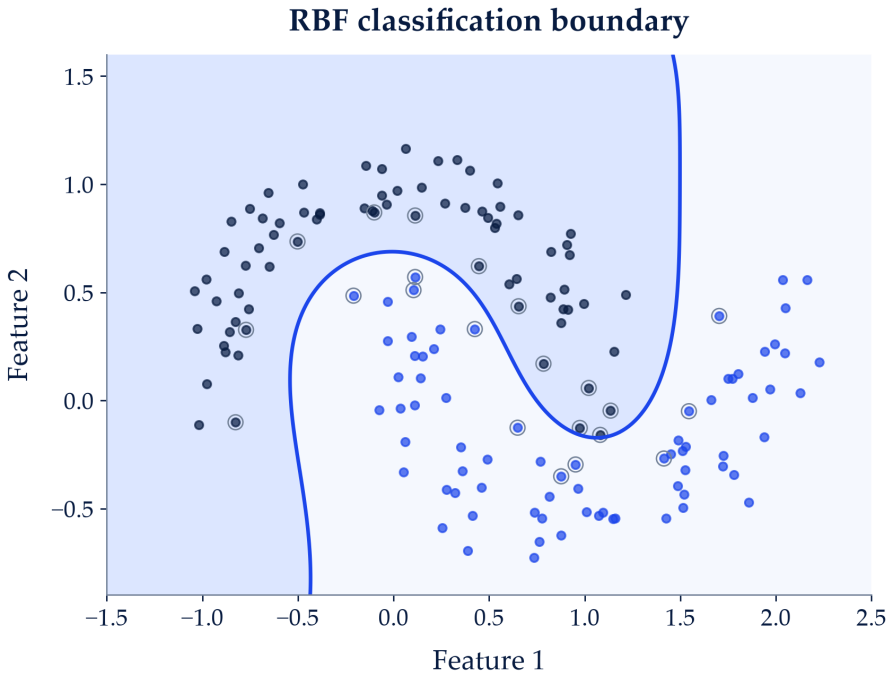


Figure 6.3. An RBF support-vector classifier fitted to two curved synthetic groups. The blue contour is decision score zero; outlined observations are support vectors. Kernel bandwidth and penalty determine the fitted geometry. The boundary is not a calibrated probability contour.

Notation for Figure 6.4

ϵ	No-penalty regression-tube half-width, 0.22 target units in this figure; it does not specify coverage probability.

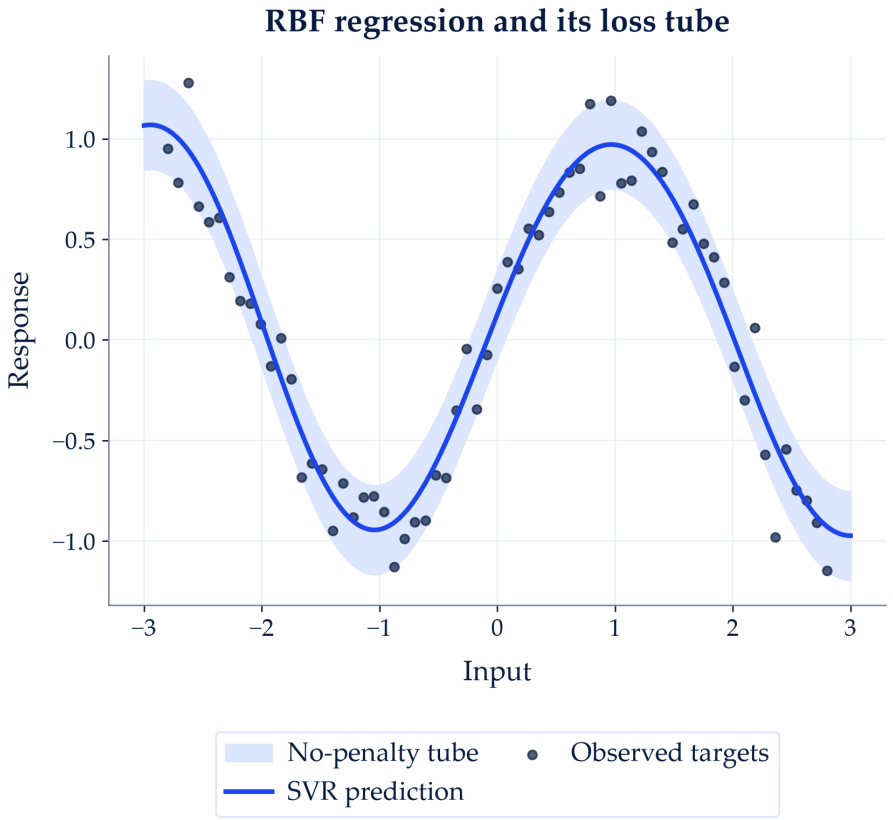


Figure 6.4. A fitted RBF support-vector regressor with epsilon-insensitive half-width 0.22 target units. The pale tube is the no-penalty region of the training loss around the prediction. It is not a prediction interval and carries no probabilistic coverage guarantee.

Notation for Figure 6.5

C, γ	Positive summed-slack penalty and positive inverse squared-distance kernel scale varied in the accompanying figure question.

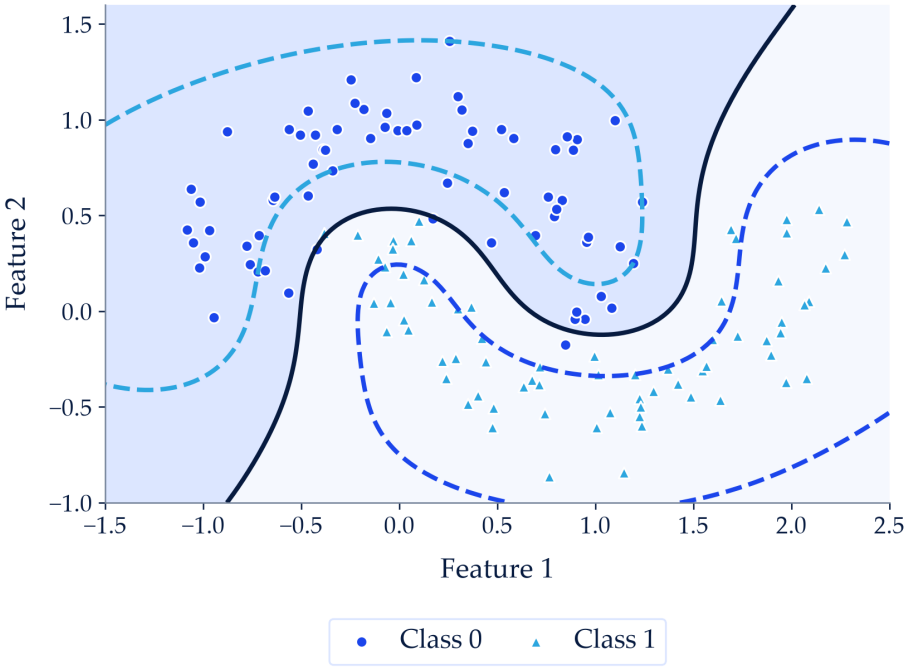


Figure 6.5. Actual RBF SVC on a synthetic two-moon dataset. The solid contour is decision score zero; dashed contours show scores -1 and +1. These are model margins, not calibrated probabilities or physical safety envelopes.

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.

Method	How prediction is constructed	Critical design choice
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 had the fewest observed errors. A separate binary development audit reconstructed linear-SVM scores to machine precision; it checked the correct free-support-vector margin condition rather than imposing equality on all coefficients.

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

The first image's top row includes observed intensities 5 and 13. Their difference is 8 intensity counts; dividing all pixels by the fixed range 16 would make that difference 0.5. The notebook instead learns coordinate-wise standardization on training data, illustrating that two legitimate transformations need not define the same distance.

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.

From Error Counts To Paired Evidence

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

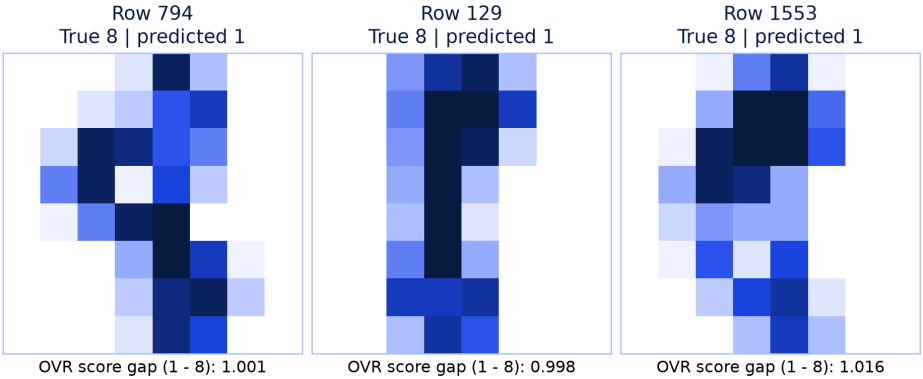


Figure 6.B1. The three Digits errors identified in the preceding case study, from four test eights predicted as ones. These are selected failure examples, not a representative sample. The score gap compares one-versus-rest presentation scores, not calibrated probabilities; its scale must not be interpreted 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.

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

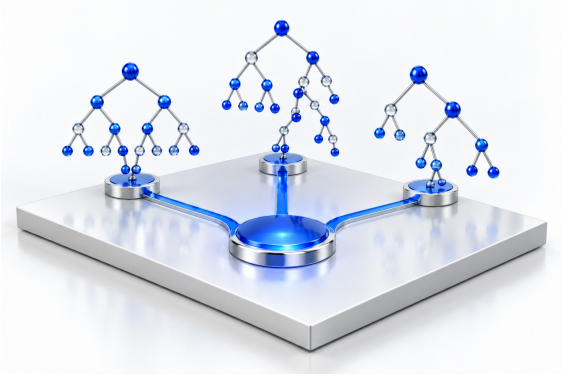
[6.7] Weinberger, K. Q.; Saul, L. K. (2009). Distance Metric Learning for Large Margin Nearest Neighbor Classification. Journal of Machine Learning Research 10, 207-244. Source / DOI

[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

CHAPTER 07

Trees And Ensemble Learning



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.

Core route: 7.1, 7.2, 7.3, 7.4. Deepening route: 7.5.

Chapter notation

CART	Classification and regression trees: recursive fitted partitions with declared node losses and split rules.
MSE, SSE	Mean squared error and sum of squared errors, respectively; the normalization distinguishes them.

Decision trees divide an input space into regions and assign a simple prediction within each region. This makes interactions and threshold-like behavior natural to represent. Trees are useful on many tabular problems, but their partitions are learned from finite samples, not discovered as exact laws. A split near a known threshold is evidence to investigate, not proof of a causal mechanism. Smooth models can also approximate sharp changes, and should not be dismissed without evaluation.

Single trees are often unstable, so this chapter develops several ways to combine them. Bagging averages models fitted to resampled data; random forests add feature randomization; gradient boosting sequentially improves a loss; voting and stacking combine predictions from potentially different model families. These procedures have different data-flow requirements. In particular, a stacking model must learn from predictions that were produced without fitting on the corresponding target values.

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 repeatedly asks an axis-aligned question such as whether feature j is at most a threshold s. Following the answers sends an observation to a leaf, where ordinary squared-error trees predict the mean training target. The resulting predictor is piecewise constant and can express interactions: a split on one feature may matter only after another split has selected a region. Trees usually do not require feature standardization because positive affine changes of feature units preserve candidate partitions, apart from 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

Add and subtract the sample mean inside each residual. The cross term vanishes because deviations from a sample mean sum to zero. The remaining extra term is a nonnegative quadratic, uniquely minimized at the mean.

$$\sum_{i \in I}(y_i - c)^2 = \sum_{i \in I}(y_i - \bar{y}_I)^2 + n_I(c - \bar{y}_I)^2 \tag{7.T1}$$

Apply the same decomposition inside each child, measuring its observations around the parent mean. Adding the two child identities separates within-child error from between-child error.

$$\Delta = n_L(\bar{y}_L - \bar{y})^2 + n_R(\bar{y}_R - \bar{y})^2 \tag{7.T2}$$

$$\bar{y} = \frac{n_L\bar{y}_L + n_R\bar{y}_R}{n_L + n_R} \tag{7.T3}$$

$$\Delta = \frac{n_L n_R}{n_L + n_R}(\bar{y}_L - \bar{y}_R)^2 \tag{7.T4}$$

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.

□

A large difference between child means is useful only together with child sample sizes. An isolated extreme target can create a large training gain, so minimum leaf size and validation remain necessary. Greedy optimality for one split is not global optimality of the completed tree.

▶ 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.

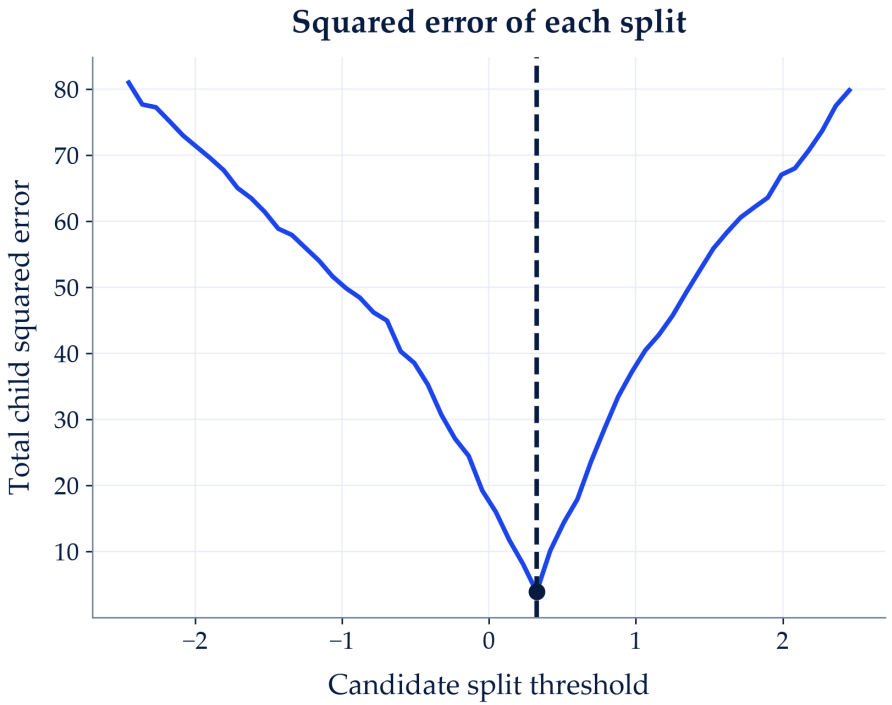


Figure 7.1. Every midpoint between ordered observations is evaluated using the sum of within-child squared errors and independently fitted child means. The minimum agrees numerically with a depth-one CART model. The selected threshold is a local split optimum, not a globally optimal tree.

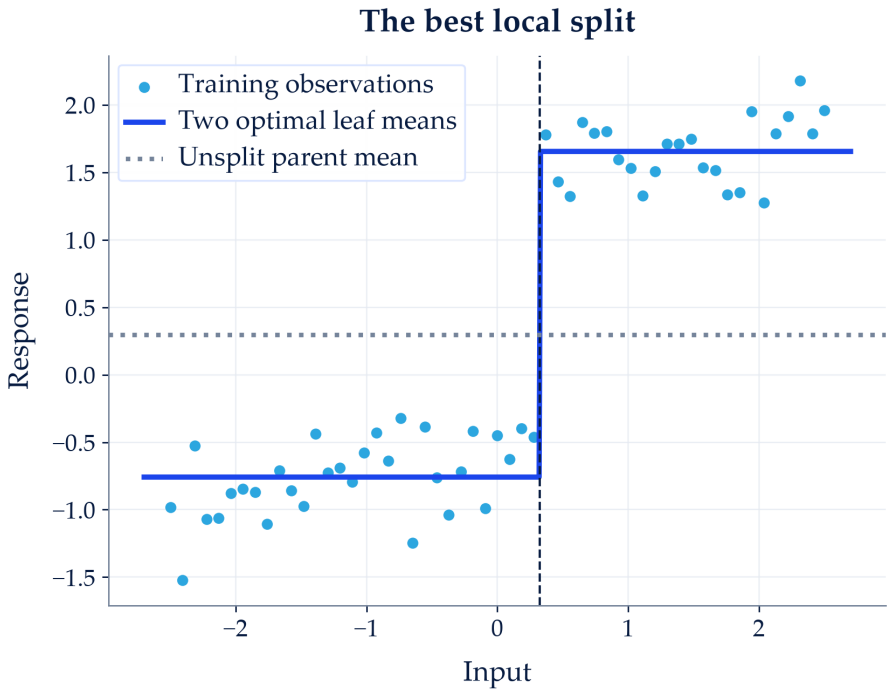


Figure 7.2. The fitted depth-one regression tree uses the optimal mean in each child. The dashed vertical line is the selected split; the dotted horizontal line is the unsplit parent mean. Its error reduction equals the count-weighted squared difference between child means, checked numerically.

7.2 Classification impurity and complexity control

Classification trees measure how mixed the classes are within a node. Gini impurity is one minus the sum of squared class proportions; entropy is the negative sum of proportion times log proportion, treating zero-probability terms as zero. Both vanish for a pure node. A candidate split is evaluated by parent impurity minus a sample-size-weighted average of child impurities. Weighting is essential: an unweighted average can overvalue a tiny pure child while ignoring a large mixed remainder.

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

$$\frac{1}{n_R}\sum_{i\in I_R}\sum_{k=1}^{K}(\mathbf{1}[y_i=k]-q_k)^2 = 1-\sum_{k=1}^{K}p_k^2 + \sum_{k=1}^{K}(q_k-p_k)^2 \tag{7.T5}$$

Expansion uses the fact that each observation has exactly one active class indicator. The last sum is nonnegative and vanishes only at the empirical frequency vector. The remaining constant is Gini impurity. Thus squared error on class indicator vectors is mathematically valid; squaring arbitrary numeric class labels would impose an unrelated ordering.

$$-\sum_{k=1}^{K} p_k \log q_k = -\sum_{k=1}^{K} p_k \log p_k + \sum_{k=1}^{K} p_k \log \frac{p_k}{q_k} \tag{7.T6}$$

The last term is the relative entropy between the empirical and predicted class distributions. Its nonnegativity follows from the logarithm inequality or Jensen's inequality; equality holds at the empirical distribution. A predicted zero for an observed class incurs infinite log loss. Therefore the optimized log loss is entropy.

Total empirical loss sums observation-wise contributions. Consequently, comparing a split must weight each child's optimized node loss by its fraction of parent observations. An unweighted average would evaluate a different problem in which a tiny child and a large child have equal influence.

□

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.

7.3 Bagging and random forests

Bagging draws bootstrap samples with replacement and combines the fitted predictors: average regression outputs, or average probabilities or votes for classification. It helps when the base learner is sensitive to sample perturbations, as deep trees often are. Independent bootstrap draws conditional on the observed dataset do not create independent estimates of the population relationship. The original sample and systematic errors remain shared, limiting how much averaging can remove.

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 observation omitted from a tree's bootstrap sample can serve as an out-of-bag evaluation case for that tree. Aggregating these omitted-tree predictions provides a useful internal estimate under an appropriate independent-observation setting. It is not a replacement for group-aware or temporal evaluation when observations are dependent. Tune leaf size, feature subsampling, and other complexity settings on development data, and reserve final evaluation for an untouched test set. Record memory and prediction cost because hundreds of deep trees can become operationally substantial.

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.

$$\bar{f}(x) = \frac{1}{B}\sum_{b=1}^{B} f_b(x) \tag{7.8}$$

$$\mathrm{Var}(\bar{f}) = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2 \tag{7.9}$$

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.
$f_b, \bar{f}$	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

$$\bar{f} = \frac{1}{B}\sum_{b=1}^{B} f_b \tag{7.T7}$$

$$\mathrm{Var}(\bar{f}) = \frac{1}{B^2}\sum_{b=1}^{B}\sum_{c=1}^{B}\mathrm{Cov}(f_b, f_c) \tag{7.T8}$$

There are as many diagonal covariance terms as base estimators; each is the common variance. Every ordered pair of distinct estimators contributes the common correlation times that variance.

$$\mathrm{Var}(\bar{f}) = \frac{B\sigma^2 + B(B-1)\rho\sigma^2}{B^2} = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2 \tag{7.T9}$$

For a fixed ensemble size greater than one, positive semidefiniteness requires the common correlation to be no smaller than minus the reciprocal of one less than the ensemble size. A fixed negative correlation cannot persist for arbitrarily many estimators. The familiar infinite-ensemble floor assumes a feasible fixed nonnegative correlation.

Bootstrap draws may be independent after conditioning on the observed training sample, yet the trained predictors share the information and errors in that original sample. Unconditional covariance therefore need not vanish. Random feature subsets can reduce that covariance, but can also increase the weakness or bias of individual trees; neither effect has a universal 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.

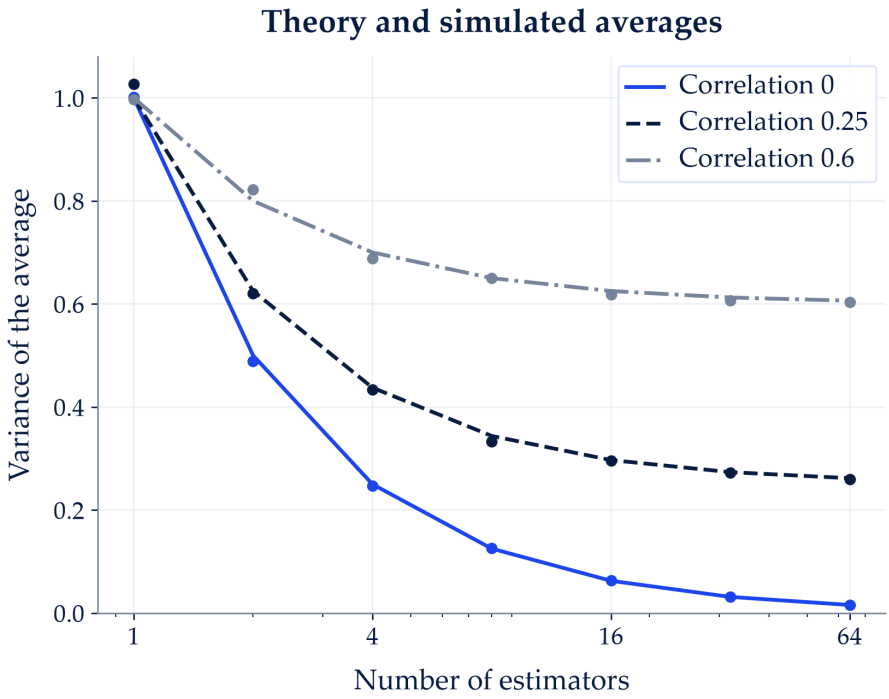


Figure 7.3. Variance of an equal-weight average under equal individual variances and fixed common correlation. Lines show the covariance identity; markers show 12,000-draw simulations for each ensemble size. The nonzero correlation component remains as the number of averaged estimators increases.

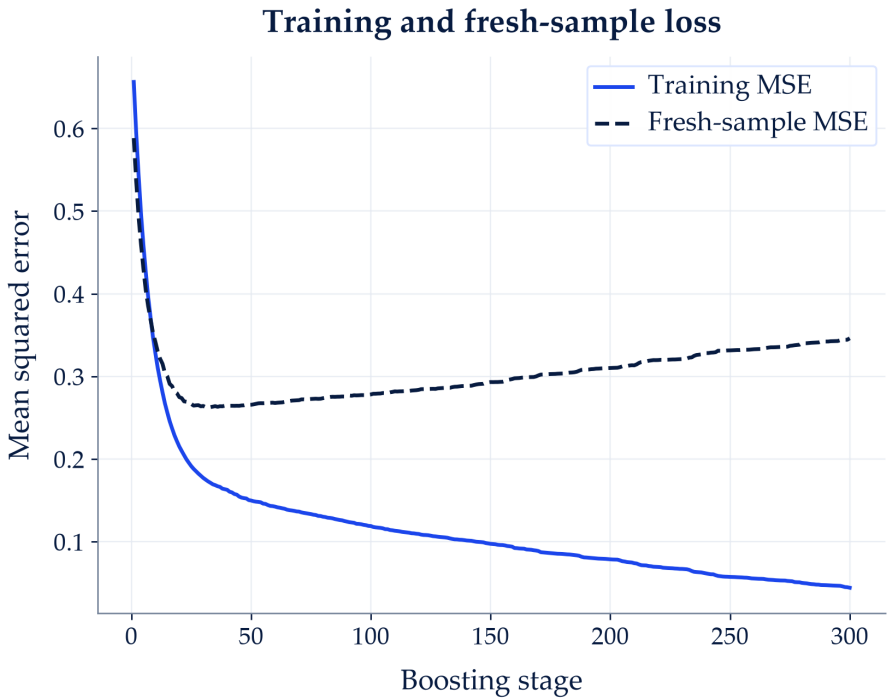


Figure 7.4. Actual squared-error gradient boosting on noisy synthetic regression. Training loss continues falling while a separate fresh-sample loss can flatten or worsen. This fresh sample diagnoses behavior and is not used to select a boosting stage. The curves do not claim monotonic improvement in generalization.

7.4 Gradient boosting follows the loss gradient

Boosting updates a predictor rather than replacing it. Start with a constant minimizing the chosen training loss. At each stage, differentiate that loss with respect to each current prediction, negate the derivatives, and fit a weak learner to those pseudo-residuals. Add its predictions multiplied by the learning rate. For half-squared error, the pseudo-residuals are target minus prediction; other losses require their own derivatives.

For half-squared error, the negative gradient is the ordinary residual, so fitting residuals is an exact special case. For other losses, including logistic loss, the pseudo-residual has a different form. Some implementations further optimize

leaf values or use second-order information. Calling all boosting residual fitting without specifying the loss hides this distinction. Histogram gradient boosting groups feature values into bins to accelerate split search and is a genuine gradient-boosted tree method, not simply another name for 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.

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

$$\mathcal{J}(F) = \sum_{i=1}^{n} \ell(y_i, F(x_i)) \tag{7.T10}$$

$$r_i = -\frac{\partial \ell(y_i, F(x_i))}{\partial F(x_i)} \tag{7.T11}$$

$$\frac{d}{dt}\mathcal{J}(F + th)\Big|_{t=0} = -\sum_{i=1}^{n} r_i h(x_i) \tag{7.T12}$$

The chain rule proves the directional derivative. A positive gradient-direction inner product makes this derivative negative, ensuring improvement for sufficiently small steps by differentiability. Fitting a tree by squared error to the negative gradients attempts to approximate that direction within the restricted class of tree functions; it does not automatically guarantee improvement for an arbitrarily large update.

$$\ell(y, F) = \tfrac{1}{2}(y - F)^2 \Rightarrow r = y - F \tag{7.T13}$$

$$\ell(y, F) = \log(1 + e^F) - yF \Rightarrow r = y - \sigma(F) \tag{7.T14}$$

For squared error, the direction is the familiar target residual. For binary log loss, the function being updated is a log-odds score, while its negative gradient subtracts the current probability from the label. A line search or fitted leaf values may further scale a direction; shrinkage reduces its contribution. Early stopping selects stage count on development evidence.

□

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) and predictions (2, 2, 4) give half-squared-error pseudo-residuals (-1, 0, 2). A weak tree returns (-0.5, -0.5, 2). Multiplying by learning rate 0.1 and adding gives (1.95, 1.95, 4.2). The summed half-squared loss falls from 2.5 to approximately 2.073. This verifies a training improvement for this step, not improved future prediction. The tree supplies a correction rather than replacing the ensemble; use validation to stop. For logistic loss, differentiate with respect to its score instead of blindly 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

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.

$\mathcal{A}$, FirstDecrease — Ordered steps eta,eta/2,...,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 on the pseudo-residuals under the declared tree rule. 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.

Require Valid data, fixed controls and untouched final test.

Finite constant minimizer. $F_0 \leftarrow \arg\min_{c \in \mathbb{R}} \sum_i \ell(y_i, c)$

Require Finite initializer, training loss and validation score.

$\mathcal{R} \leftarrow [(0, L(F_0), V(F_0))]$

$m_* \leftarrow 0$

$s \leftarrow$ budget exhausted

For $m = 1, \ldots, M$

  $r_i \leftarrow -\partial_z \ell(y_i, z)\big|_{z=F_{m-1}(x_i)}$

  Require Finite pseudo-residuals.

  Squared-error regression tree. $h_m \leftarrow \mathrm{FitTree}(X, r; \mathcal{H})$

  Stop if No verified descent direction. $\sum_i r_i h_m(x_i) \leq \delta$

  Log every attempted step.

  $a_m \leftarrow \mathrm{FirstDecrease}(F_{m-1}, h_m, \mathcal{A})$

  Stop if Search fails; do not add this tree.

Algorithm 7.1 (continued)

$F_m \leftarrow F_{m-1} + a_m h_m$

Accepted training and validation scores.

$\mathcal{R} \leftarrow \mathcal{R} \,\|\, [(m, L(F_m), V(F_m))]$

If better Retain earlier stages on ties.

$V(F_m) < V(F_{m_*}) \;\Rightarrow\; m_* \leftarrow m$

End for

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. Binary logistic initialization needs both classes unless a finite smoothing rule is declared. Training descent is not a generalization guarantee; final test data never select a stage.

7.5 Voting and leakage-free stacking

Voting combines models using a fixed rule. Hard voting aggregates class labels; soft voting averages class probabilities, optionally with nonnegative weights summing to one. Align class order across models before averaging. A weighted mean can balance strengths, but weights chosen to maximize a test score leak test information. Poorly calibrated probabilities can also make one model dominate the average inappropriately. Use development validation for weights and compare the ensemble with its constituents to justify added complexity.

A stacking meta-model learns from base-model predictions. Generate each development row's meta-features with base procedures trained on other folds, including their preprocessing and tuning. Fitting a base model on that row's target first produces optimistic meta-features unavailable at deployment. After filling the out-of-fold matrix, fit the meta-model. These held-out base predictions are training inputs for the meta-model, not an independent evaluation of the complete stack.

Fit the meta-model on the completed out-of-fold prediction matrix. Then refit each base procedure on all development data and feed its predictions on new inputs into the fitted meta-model. Hyperparameter selection needs its own

disciplined separation; extensive tuning around the same out-of-fold matrix can still overfit the development process. Grouped or temporal tasks require compatible folds, and some temporal rows may have no valid earlier training history. Finally evaluate the whole stack once on the 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

$$Z_{ib} = f_b^{(-v(i))}(x_i) \tag{7.T15}$$

Under independent sampling, training pairs outside a fold are independent of the held-out pair. Conditional on its input, the base procedure therefore cannot adapt its fitted parameters to that row's particular target. By contrast, an interpolating base model trained on all rows can place the target itself into a meta-feature, a relationship that need not hold on future data.

The meta-model then deliberately fits the complete meta-feature matrix and development targets. Its training residuals are in-sample residuals for that second-stage fit; exclusion at the first stage does not protect this score from second-stage overfitting. After generating meta-features, refit bases on development data, freeze the meta-model, and evaluate the whole stack on untouched observations. Group or time dependence requires compatible 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.

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.

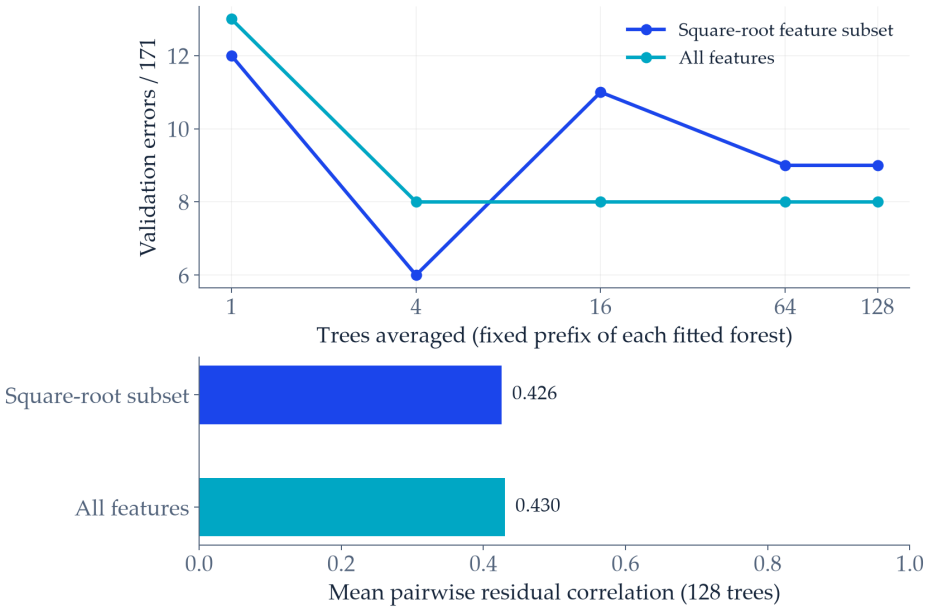


Figure 7.5. Measured forest prefixes on a fixed WDBC development split: 398 fitting and 171 validation rows. Two 128-tree forests use depth six and either square-root feature subsampling or all features. The upper panel evaluates fixed prefixes, not validation-selected tree counts; the lower panel measures 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.

Method	Combination mechanism	Key limitation
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.

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 achieved validation macro F1 of 0.798, versus 0.748 for the forest and 0.741 for the constrained tree. Its final improvement over the majority rule was visible in both accuracy and macro F1, not accuracy alone.

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 notebook prints an actual retained training row and its encoded dimension. Identify one observed categorical value, locate its one-hot coordinate, and verify that its active indicator is one. Compare the final confusion-matrix sum with the number of selected original-test source indices; the two counts must agree.

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.

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

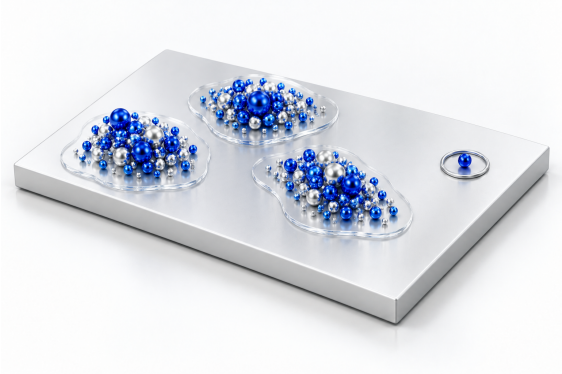
[7.8] Ke, G.; Meng, Q.; Finley, T.; Wang, T.; Chen, W.; Ma, W.; Ye, Q.; Liu, T.-Y. (2017). LightGBM: A Highly Efficient Gradient Boosting Decision Tree. Advances in Neural Information Processing Systems 30. Source / DOI

[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

CHAPTER 08

Clustering And Anomaly Detection



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.

Core route: 8.1, 8.2, 8.3, 8.4. Deepening route: 8.5.

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.

When target labels are unavailable, learning can still reveal useful structure. Clustering groups observations under an explicit similarity rule; anomaly detection highlights observations that differ from a reference. Neither procedure knows the meaning of a group or the consequence of an unusual event. A cluster may represent a real population, an acquisition batch, a measurement artifact, or an arbitrary partition of a continuum. An anomaly may be a fault, a valuable discovery, or an ordinary example underrepresented in the reference data.

The chapter develops k-means, hierarchical clustering, and DBSCAN, then connects their limitations to anomaly detection and drift monitoring. We will distinguish mathematical objectives from substantive conclusions and use controlled synthetic examples to make failures visible. The main discipline is to define what data establishes the reference, what question the geometry answers, and what additional evidence is required before a geometric signal becomes an operational decision.

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 minimizes summed squared Euclidean distances to K centroids. Lloyd's iteration has two conditional optimizations: assign each observation to its nearest current centroid, then replace each occupied centroid by its assigned mean. Repeat under a declared stopping rule. Each exact step cannot increase the objective, but neither step searches all simultaneous assignments and centers; monotone progress is not a global-optimum certificate.

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**

$$J(c, \mu) = \sum_{i=1}^{n} \|x_i - \mu_{c_i}\|_2^2 \tag{8.T1}$$

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.

$$c_i^{t+1} \in \arg\min_{1 \le k \le K} \|x_i - \mu_k^t\|_2^2 \tag{8.T2}$$

$$J(c^{t+1}, \mu^t) \le J(c^t, \mu^t) \tag{8.T3}$$

For fixed assignments, add and subtract the assigned cluster mean. Its centered deviations sum to zero, eliminating the cross term. The remaining quadratic shows that the mean is the unique minimizer for a nonempty cluster.

$$\sum_{i: c_i = k} \|x_i - u\|_2^2 = \sum_{i: c_i = k} \|x_i - \bar{x}_k\|_2^2 + n_k \|u - \bar{x}_k\|_2^2 \tag{8.T4}$$

$$J(c^{t+1}, \mu^{t+1}) \le J(c^{t+1}, \mu^t) \le J(c^t, \mu^t) \tag{8.T5}$$

The objective is nonnegative, so its values converge. With nonempty clusters, strict decrease whenever assignments change, and a fixed tie rule, finitely many possible assignments also give finite termination. Degenerate ties require care; monotone objective convergence alone is the more general claim. A stable solution minimizes each coordinate block while holding the other fixed, not every simultaneous change.

□

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.

A centroid's closed nearest-center region, allowing shared boundaries at ties, is an intersection of half-spaces and is therefore convex. Standardization replaces raw Euclidean distance by a diagonal quadratic metric rather than leaving the problem unchanged.

□ **Proof**

$$\begin{aligned} &\|x-\mu_k\|_2^2 \le \|x-\mu_j\|_2^2 \\ \iff \quad &2(\mu_j-\mu_k)^\top x \le \|\mu_j\|_2^2 - \|\mu_k\|_2^2. \end{aligned} \tag{8.T6}$$

Expansion cancels the squared query norm on both sides. Each pairwise preference is therefore a linear inequality. Requiring it against every other centroid intersects half-spaces. If two queries satisfy all the inequalities, every convex combination of them does too, proving convexity of the cell.

$$D = \operatorname{diag}(s_1, \dots, s_d) \tag{8.T7}$$

$$\|D^{-1}(x-z)\|_2^2 = \sum_{j=1}^{d} \frac{(x_j - z_j)^2}{s_j^2} \tag{8.T8}$$

Translation by a shared training mean cancels in differences; division by feature scales remains. Large scales reduce a coordinate's distance contribution. A change of measurement units accompanied by the same change in its scale preserves standardized distances, whereas changing raw units alone does not.

□

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

Notation and assumptions

$X = (x_i)_{i=1}^{n}, d, K$	Finite n-by-d input matrix and requested centroid count; 1<=K<=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; $\sum_i$ ||x_i -mu_(c_i)||^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.
$c_{\text{old}}, s, \mathcal{B}, J_*$	Pre-update labels, run status, saved best state, and its objective.
$\text{Init}, K_{\text{occ}}$	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. 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.

01 **Require** Finite data, valid K, seeds and budgets.

02 $J_* \leftarrow +\infty$

03 **For** $r = 1, \dots, R$

04 $\quad \mu \leftarrow \text{Init}(X, K, r)$

Algorithm 8.1 (continued)

$c \leftarrow A(\mu)$

$J \leftarrow \mathcal{J}(c, \mu)$

$s \leftarrow \text{budget exhausted}$

For $t = 1, \dots, T$

  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.

  $c = c_{\text{old}} \;\Rightarrow\; s \leftarrow \text{converged}$

  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 failure occurs before any valid state, return failure without a fitted result.

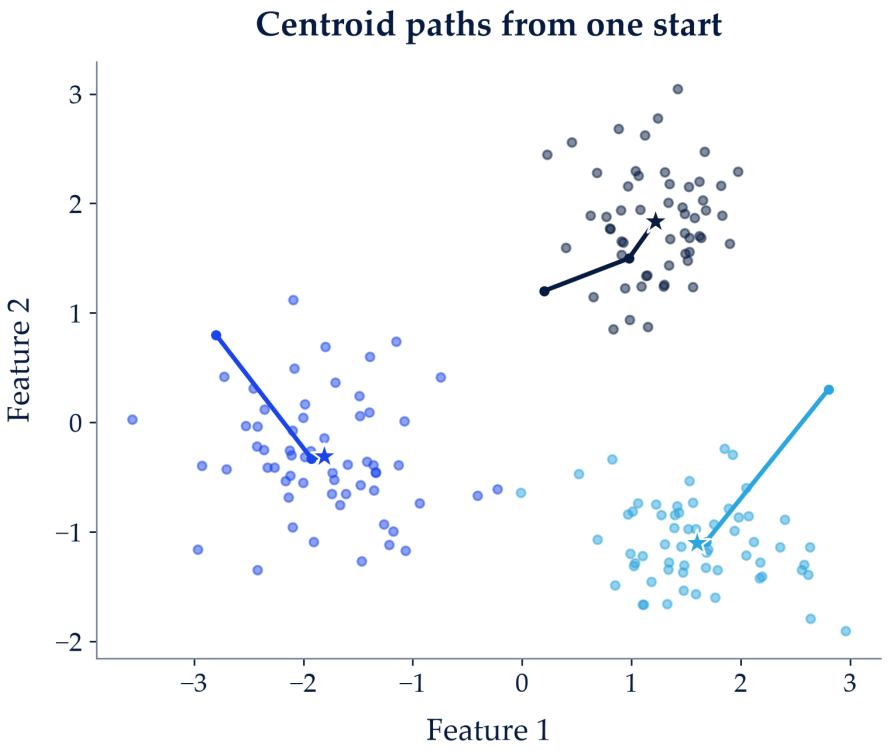


Figure 8.1. Explicit Lloyd iterations alternate nearest-centroid assignment and cluster-mean updates from one initialization. Paths show centroid movement; stars show final centers. The final grouping is one local result and can depend on initialization.

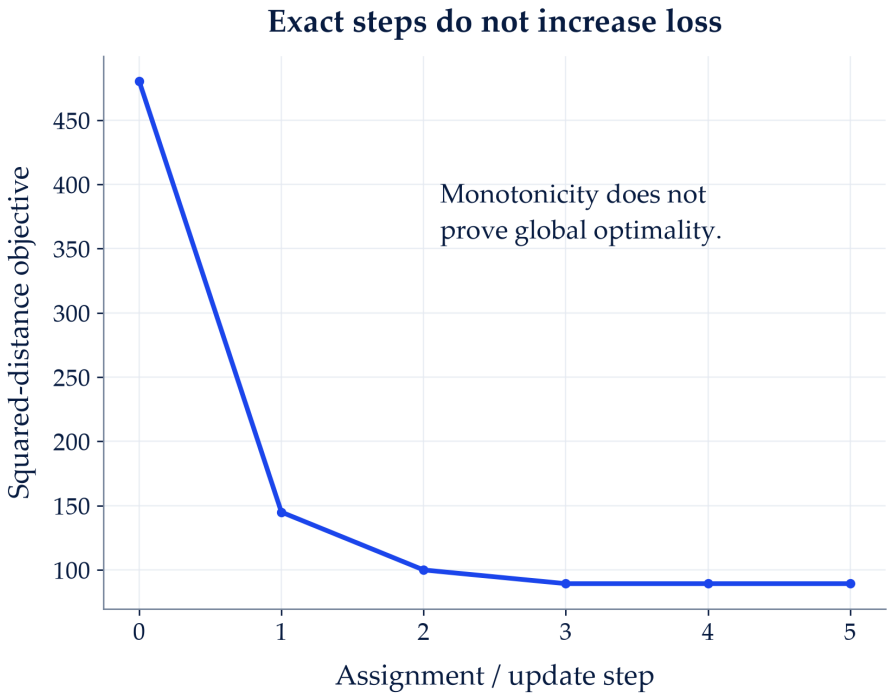


Figure 8.2. The squared-distance objective is recorded after every assignment and every mean update in the same Lloyd run. Non-increase is asserted numerically. Monotonic improvement of this objective does not prove convergence to a global optimum or establish application meaning for the clusters.

8.2 Evaluating cluster structure without inventing labels

The globally optimal k-means objective is non-increasing as K grows, reaching zero when every distinct point can receive its own center. Independently fitted local optima need not follow that trend. Raw inertia therefore cannot select K by unconstrained minimization. An elbow plot highlights diminishing returns but often lacks a unique elbow. Silhouette compares an observation's average within-cluster distance with its nearest alternative-cluster distance. This internal diagnostic favors certain geometries and does not establish that the clustering matches the application's intended categories.

Test stability by changing seeds, samples, or defensible preprocessing. Compare partitions of the same indexed observations using a label-permutation-invariant measure such as adjusted Rand index. For resampled fits, use shared observations or a common probe set with a defined assignment rule. Stability can reflect a persistent acquisition artifact; instability can also occur for a real but sparsely sampled subgroup. Interpret agreement alongside provenance and the intended downstream use, not as semantic ground 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] introduced the silhouette to compare within-cluster cohesion with separation from alternative clusters. The resulting score evaluates a partition relative to a chosen dissimilarity; it does not establish that the clusters correspond to 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 mean within-group distance one and mean alternative-group distance 8.5. Its silhouette is (8.5 minus 1)/8.5 = 15/17, approximately 0.882. Observation 1 gives 13/15, approximately 0.867; symmetry gives an overall mean approximately 0.875. This quantifies separation under this metric, not the existence of two natural categories. Compare candidate group counts, scaling, and resampling stability, then inspect acquisition batches and task relevance. Better coverage of relevant conditions may matter more than the highest internal score.

Interpretation and limits

A high silhouette or stable partition does not provide missing semantic labels. Internal geometry and downstream usefulness are separate evaluation dimensions.

8.3 Hierarchical clustering and linkage choices

Agglomerative hierarchical clustering starts with individual observations and repeatedly merges the closest pair of groups. The sequence forms a dendrogram, showing how groups combine as the resolution becomes coarser. Cutting the hierarchy at a chosen height or group count yields a flat partition. This can help explore nested structure and avoid committing to one K before fitting, but the hierarchy still depends strongly on the metric, linkage rule, and sampled observations. A tree-shaped display does not imply a true evolutionary or causal hierarchy.

Linkage determines which merge is cheapest. Single linkage takes the nearest cross-group pair and can follow thin chains; complete linkage takes the farthest pair and favors bounded diameters. Average linkage averages all cross-group distances. Ward instead minimizes the increase in within-cluster squared error, requiring Euclidean geometry. Changing linkage changes the optimization question, not merely the dendrogram's appearance.

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.

$$\Delta_{Ward}(A, B) = \frac{|A||B|}{|A| + |B|} \|\mu_A - \mu_B\|^2 \tag{8.7}$$

Ward's merge cost is the increase in within-group squared error for Euclidean coordinates. A software dendrogram may display a transformed linkage distance rather than this raw increase.

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.

Complete supporting proof: Mathematical Supplement, A6 (in this book).

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. Treat (0, 1) and (4, 5) as endpoint clouds, with 2 and 3 forming a bridge. Single linkage can connect the whole chain at distance one, since each neighboring gap is one. Complete linkage's final merge has height five, the distance between the extremes. Neither number determines whether the bridge is a meaningful continuum or contamination.

Remove or investigate the bridge points only with provenance-based justification, and report the metric and linkage used.

Interpretation and limits

A dendrogram is an algorithm-dependent hierarchy, not automatic evidence of causal ancestry or a universally correct grouping resolution.

8.4 DBSCAN and density connectivity

DBSCAN counts indexed observations within radius epsilon, including the query observation itself. A point meeting the minimum count is core; a noncore point neighboring a core point is border; the remainder is noise. Clusters join connected core neighborhoods and attach border points. Border points cannot extend a connectivity path to merge otherwise separate core components. Duplicate coordinates count as separate observations, 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.

Noise is a geometric label under the fitted settings, not a verified fault. A legitimate rare mode can be marked noise, while an emerging abnormal regime can form a dense cluster and receive an ordinary group label. Dense gradual drift may remain connected to the original cluster. Standard DBSCAN fitting is transductive over the supplied observations and does not provide a general fitted predict interface for new points. Re-running it on a new batch changes the reference and can change the interpretation of both existing and new observations.

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 Figure 8.3

K	Requested centroid count; the displayed nearest-centroid fit uses two centroids on the same data as the density method.

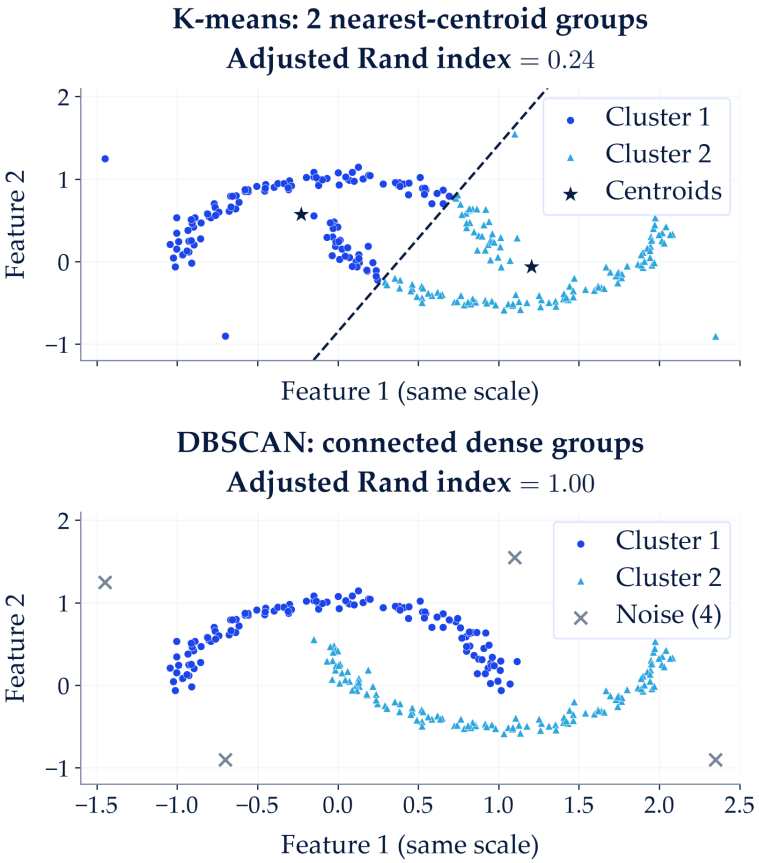


Figure 8.3. Both panels use the identical 250 noisy moon observations (seed 31) plus four declared distant observations, in the same Euclidean coordinates and axis limits. K-means (K=2, 10 initializations) cuts across the curves because nearest-centroid regions are convex. DBSCAN (radius 0.18, minimum 5 neighbors including self) follows the curved dense groups and marks 4 points as noise. Adjusted Rand index uses the known synthetic moon labels on the original 250 points only: K-means 0.24, DBSCAN 1.00. These teaching labels are unavailable in ordinary clustering; the score is not evidence of application meaning.

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.
$N_\epsilon(x)$	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.

$$N_\epsilon(x) = \{i \in \{1, \dots, n\} : d(x, x_i) \leq \epsilon\} \tag{8.8}$$

$$x \text{ core} \iff |N_\epsilon(x)| \geq m \tag{8.9}$$

The neighborhood contains observation indices, so repeated observations count separately. The query x is itself a fitted observation, and its own index is counted. A point is core when this count reaches min_samples, denoted m. Epsilon controls the distance 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**

$$N_\epsilon(i) = \{j \in \{1, \dots, n\} : d(x_i, x_j) \leq \epsilon\} \tag{8.T12}$$

$$V = \{i : |N_\epsilon(i)| \geq m\} \tag{8.T13}$$

$$E = \{\{i, j\} : i, j \in V,\ i \neq j,\ d(x_i, x_j) \leq \epsilon\} \tag{8.T14}$$

Define direct density reachability from a core observation to every observation in its neighborhood. Along a multi-step density-reachability path, every observation from which another step departs must be core. Hence all intermediate vertices form a path in the core graph; only the last vertex may be a noncore border observation.

Conversely, every path in the core graph is a valid sequence of direct density-reachability steps because each departing vertex is core. The two constructions therefore identify the same connected core components. A noncore observation neighboring a component attaches as a border point. If it neighbors two components, it does not create an edge between them; an implementation's traversal can determine its final cluster assignment.

An observation outside the core set with no core neighbor is noise. It may still have nearby noncore observations: noise does not mean literal isolation. Core memberships are determined by the graph, but ambiguous border memberships need not be uniquely determined by the geometric definitions.

□

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

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; labels0 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|>=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.

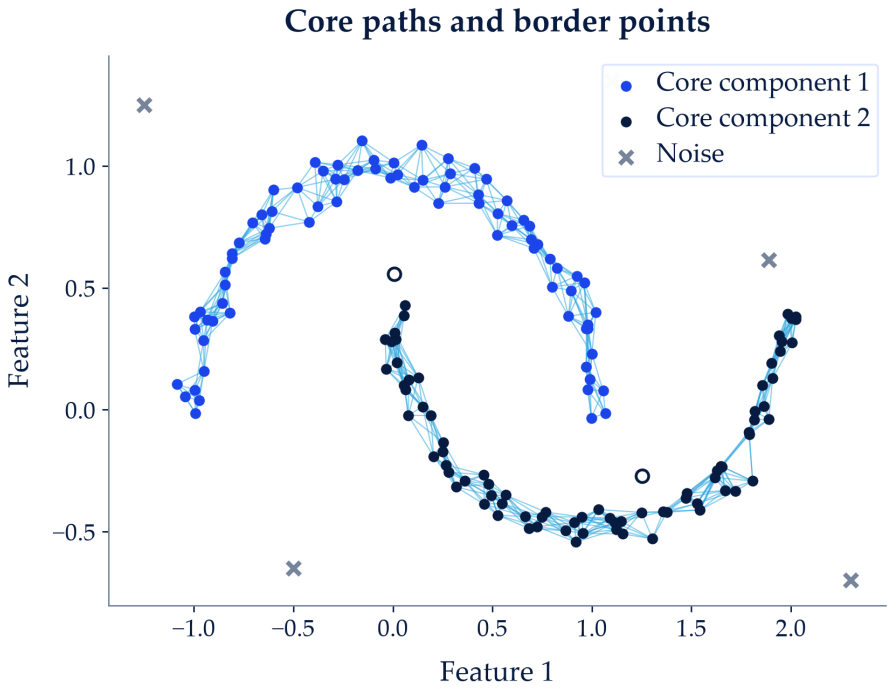


Figure 8.4. DBSCAN neighborhoods use radius 0.22 and minimum count six, including the observation itself. Edges connect core observations only. Hollow markers are border points; slate crosses are noise. Core connectivity can follow curved groups without identifying a meaningful physical category.

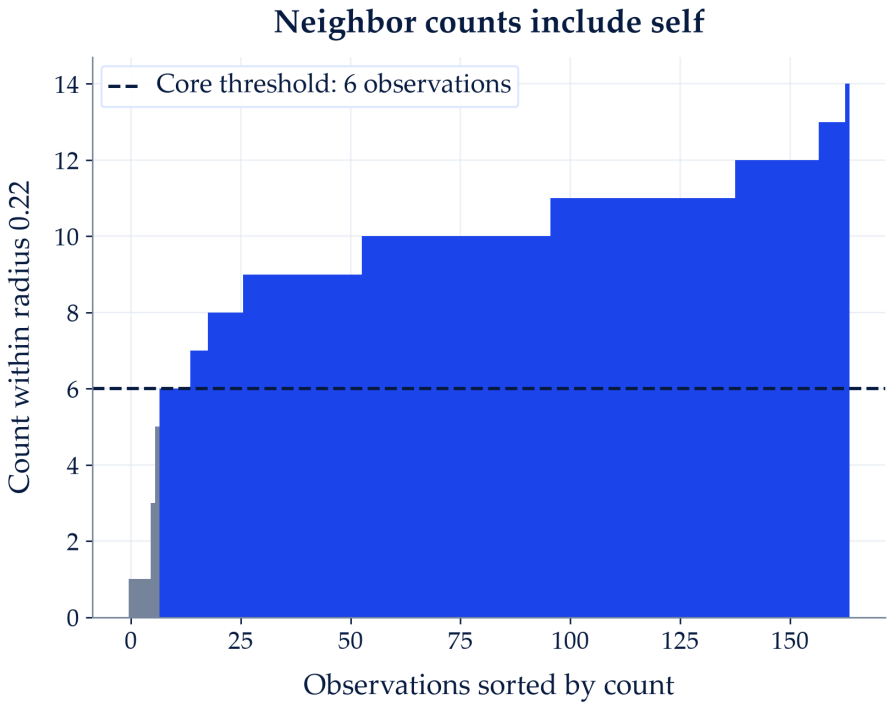


Figure 8.5. Direct counts within radius 0.22, sorted by size. A point is core exactly when at least six observations including itself lie in its neighborhood. The manually computed condition is checked against the fitted DBSCAN core indices. Low-count border and noise points are not both automatically outliers.

8.5 Anomaly scores, thresholds, and drift

An anomaly detector assigns a score relative to a reference distribution or learned notion of normality. Methods include distance to known data, density estimation, reconstruction error, and isolation-based scores. Novelty detection trains primarily on a reference believed to represent acceptable behavior and scores future observations. Outlier detection instead searches for unusual observations within a dataset that may already contain contamination. These distinctions affect training data, score interpretation, and whether the model should be refitted when new observations arrive.

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.

Drift is a distribution change over time, not necessarily an extreme point. Input drift changes the input distribution; covariate shift additionally assumes the conditional outcome law is unchanged. Other shifts can alter prevalence or that conditional law. Monitor missingness, units, ranges, and score distributions against a saved reference; use delayed labels to assess performance. Assign an investigation owner and a baseline-update rule. Automatically fitting all alerted data can absorb the change under investigation rather than resolve it.

Liu, Ting, and Zhou [8.5] introduced isolation-based anomaly scoring, and Chandola, Banerjee, and Kumar [8.6] distinguish several kinds of anomaly-detection problems. A high score signals unusualness under a model; validating an operational alert still requires context, costs, and representative examples. Breunig and colleagues [8.9] define local outlier factor, comparing an observation's local density with that of its neighbors. This makes unusualness relative to a local reference, while leaving the distinction between an unusual observation and a confirmed harmful event unresolved.

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.
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.

$$\tau = \text{Quantile}_{1-a}\{s(x_i) : x_i \in D_{\text{baseline,val}}\} \tag{8.10}$$

$$\text{alert}(x) = \mathbf{1}[s(x) > \tau] \tag{8.11}$$

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.

Method	Structure favored	Parameters and caveats
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.
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

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

Validation silhouette was 0.322 for two groups, above the alternatives. Three additional seeds reproduced its validation partition exactly. Nevertheless, the selected grouping only partially agreed with the external cultivar labels, and changing scale changed the partition.

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 notebook prints the first actual fitting wine's alcohol and proline values and their standardized coordinates. Reproduce one coordinate by subtracting that training feature's mean and dividing by its fitted population standard deviation. Explain why the two raw numerical magnitudes should not automatically determine their scientific importance.

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.

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

CHAPTER 09

Probability Calibration And Model Diagnosis



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.

Core route: 9.1, 9.2, 9.3, 9.5. Deepening route: 9.4.

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 established how classification scores, probabilities, and decisions differ; Chapter 4 established how to evaluate a selected procedure without reusing its test evidence. We now assume a fitted predictor and ask a different question: what does its numerical confidence mean, and what evidence can explain an apparent failure? This question matters when a probability is used to prioritize cases, quantify uncertainty, or compare risks rather than merely rank observations.

The chapter develops probability diagnostics in a deliberate sequence. Proper scores establish what is being rewarded; calibration compares predictions with outcome frequencies; slices reveal where aggregate behavior fails; feature perturbations investigate dependence; and distribution-shift analysis separates warnings from demonstrated degradation. A conformal prediction result then makes the assumptions behind a finite-sample uncertainty guarantee explicit. Artifact packaging and training-loop implementation are left to Chapter 12, rather than repeated here.

9.1 Proper scores and probability information

A probability prediction makes a numerical claim about uncertain outcomes. When a system assigns 0.8 to many comparable cases, its claim is stronger than placing those cases near the top of a ranking. The ranking and threshold machinery of Chapter 3 remains useful, but it cannot test this frequency interpretation. We therefore evaluate the probability values themselves, using scores whose population optimum is truthful probability reporting.

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.

A proper score rewards truthful probabilities, not calibration alone. A constant prevalence forecast can be calibrated yet ignore useful risk differences. The next section decomposes score into uncertainty, resolution, and calibration error. Evaluate on the intended population, with any weighting declared: a changed prevalence also changes the reference forecast and score baseline.

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.

$$\text{Brier} = \frac{1}{n}\sum_{i=1}^{n}(p_i - y_i)^2 \tag{9.1}$$

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**

$$q = \Pr(Y = 1 \mid X = x) \tag{9.T1}$$

For a binary outcome, both its first and second conditional moments equal its positive-class probability. Expanding the squared residual therefore gives a quadratic in the reported probability.

$$\mathbb{E}[(p - Y)^2 \mid X = x] = p^2 - 2pq + q \tag{9.T2}$$

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

Take two equally common groups with outcome rates 0.2 and 0.8. Predicting 0.5 always gives squared loss 0.25, whichever binary outcome occurs. Reporting each group's true probability instead gives expected loss 0.2 times 0.8 = 0.16 in either group. Both predictors are calibrated, but the second improves Brier risk by 0.25 minus 0.16 = 0.09 because it retains risk information. This population calculation shows why a diagonal reliability curve alone cannot establish useful discrimination.

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.

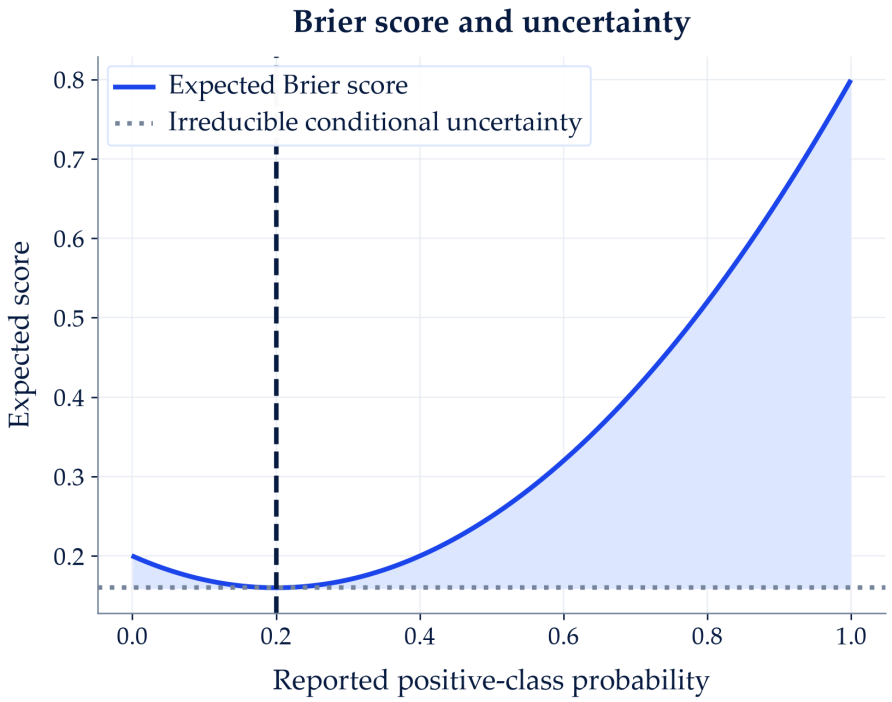


Figure 9.1. Exact expected Brier score for a Bernoulli outcome with probability 0.20. Risk separates into fixed conditional uncertainty and the shaded squared discrepancy between reported and true probability. The optimum is the true probability, independently of a later classification threshold.

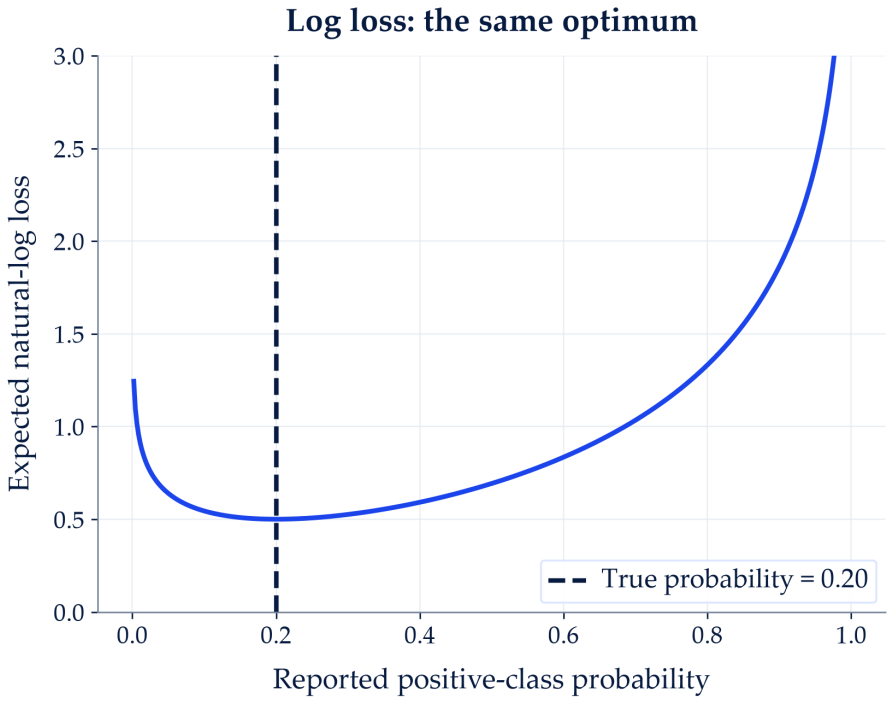


Figure 9.2. Exact expected natural-log loss under the same Bernoulli probability 0.20. Its unique interior optimum is the true probability, and confident errors are heavily penalized. This proper scoring objective assesses probability quality rather than only ranking.

9.2 Estimating and correcting calibration

Write the fitted predictor's output as a random probability on a fresh case. Population calibration means that the conditional outcome frequency, given that reported probability, equals the probability itself. A reliability diagram estimates a coarser version of this relationship by grouping predictions into bins. For each nonempty bin, compare the average prediction with the observed positive fraction and show the number of supporting observations.

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. A single expected calibration error depends on binning and can hide opposite deviations; selecting bins for visual agreement adds another selection effect.

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. If score direction is known and ranking preservation is required, constrain the sigmoid slope to be nonnegative. 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)

i	Observation index restricted to the specified bin.
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.
$p_{\mathrm{cal}}(x)$	Positive-class probability after the separately fitted calibration map.

$$p_{\mathrm{cal}}(x) = \sigma(a\, s(x) + b) \tag{9.5}$$

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

$$P = \widehat{p}(X) \tag{9.T4}$$

$$m(P) = \mathbb{E}[\,Y \mid P\,] \tag{9.T5}$$

Add and subtract the conditional mean in the prediction residual. Conditional on the original probability, the outcome minus its conditional mean has expectation zero. This makes the cross product vanish after taking expectations.

$$\begin{aligned}\mathbb{E}[(Y-P)^2] = \mathbb{E}[(Y-m(P))^2] \\ +\mathbb{E}[(m(P)-P)^2].\end{aligned} \tag{9.T6}$$

The first term is the Brier risk after ideal recalibration; the second is nonnegative. Moreover, the recalibrated value is itself calibrated by the tower property, even if several original probabilities map to the same output.

$$\mathbb{E}[Y \mid m(P)] = m(P) \tag{9.T7}$$

For binary outcomes, conditional variance equals conditional frequency times its complement. The law of total variance then separates the Brier risk into outcome uncertainty, resolution, and calibration error.

$$\begin{aligned}\mathbb{E}[(Y-P)^2] = \bar{q}(1-\bar{q}) - \mathrm{Var}(m(P)) \\ +\mathbb{E}[(m(P)-P)^2].\end{aligned} \tag{9.T8}$$

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

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.

Constrain sigmoid slope nonnegative before fitting if order preservation is required; nondecreasing isotonic maps also qualify. 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.

    $B_j \leftarrow \{i \in D_e : q_i \in I_j\}$

    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 $\bar{y}_j \leftarrow n_j^{-1} \sum_{i \in B_j} y_i$

Algorithm 9.1 (continued)

18 $\mathcal{E} \leftarrow \mathcal{E} \parallel [(n_j, \bar{q}_j, \bar{y}_j)]$

19 **End for**

20 **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. If unavailable, mark the raw comparison unavailable. 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.

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.

An aggregate diagnostic averages over a population mixture. It can therefore conceal an important error in one acquisition source, time period, or input-quality range. Define an error slice as a subset with an interpretable membership rule, then compute the same declared loss and calibration quantities within it. Report the number of observations, the number of positives, and the prediction distribution; these determine what a slice estimate can support.

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)

i	Index of an observation in the displayed slice.
y_i, p_i	Observed binary outcome and predicted positive-class probability.
$\hat{r}_g$	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

$$m(P) = \mathbb{E}[\, Y \mid P \,] \tag{9.T9}$$

$$m_G(P) = \mathbb{E}[Y \mid P, G] \tag{9.T10}$$

Condition first on the score and slice, then average over slice membership at the same score. The tower property recovers the coarser conditional frequency.

$$\mathbb{E}[m_G(P) \mid P] = m(P) \tag{9.T11}$$

Thus global calibration requires only that slice frequencies average to the reported probability; it does not require each frequency to equal it. For the risk comparison, add and subtract the finer conditional mean inside the residual. The cross term has zero expectation because the outcome minus its finer conditional mean is orthogonal to any square-integrable function of score and slice.

$$\begin{aligned}\mathbb{E}[(Y - m(P))^2] = \mathbb{E}[(Y - m_G(P))^2] \\ +\mathbb{E}[(m_G(P) - m(P))^2].\end{aligned} \tag{9.T12}$$

The last term is nonnegative, proving the claim. In the equal-mixture example with constant reported probability one-half and slice frequencies one-fifth and four-fifths, that term is nine hundredths. A diagnostic estimate of these means may not achieve the population improvement: small samples, adaptive slice search, and a changed population introduce estimation and selection error.

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.

9.4 Interpreting predictive dependence

Once a failure pattern is localized, ask what information the fitted function uses. Permutation importance evaluates a controlled input perturbation: shuffle one feature across reference observations, keep the predictor fixed, and measure the change in a chosen loss. A positive increase means that this perturbation damaged predictive performance. Repeat the shuffles to describe their variability; this is not the same uncertainty as refitting on new training samples.

Feature dependence makes the question delicate. A near-duplicate predictor may substitute for the shuffled feature, hiding reliance on their shared signal. Alternatively, independent shuffling may create impossible combinations and exaggerate loss. Grouped permutation applies the same row permutation to a declared group of columns, preserving dependence within that group while disrupting its relationship with the outcome and other columns; conditional perturbation can aim to preserve dependencies but requires a defensible conditional model. Neither method turns a predictive perturbation into the causal effect of intervening in the physical system.

Partial dependence examines a different object: the fitted response while a coordinate is set to a common value and the remaining coordinates come from a reference population. Individual conditional expectation curves retain one response curve per reference case rather than averaging them immediately. Opposing responses can cancel in the average, just as subgroup calibration errors can cancel. Restrict interpretation to supported input combinations and explicitly identify the output being 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 as a way to inspect a fitted predictor. Strobl and colleagues [9.5] show why predictor dependence matters for permutation importance; these analyses describe predictive relationships and should not be interpreted as causal intervention effects 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.

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.

Probability diagnostics are conditional on a population. Compare a reference distribution with a later one by separating the input marginal from the conditional outcome relationship. Covariate shift changes the input distribution while preserving the outcome distribution conditional on the input. Label shift changes class prevalence while preserving the input distribution within each class. A change in the outcome relationship at a fixed input is often called concept shift. These are assumptions about distributions, not labels that an unlabeled dashboard can establish automatically.

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. The artifact and training procedures in Chapter 12 support these checks without replacing the reference silently.

Under pure covariate shift, a target-population risk can sometimes be expressed as a weighted reference risk. The identity below requires the same conditional outcome law and target support contained within reference support. 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] studies importance weighting under covariate shift, grounding the density-ratio calculation below. Angelopoulos and Bates [9.6] explain conformal prediction's finite-sample marginal coverage: exchangeability and the specified construction matter, and arbitrary distribution shift is not covered by the ordinary guarantee. Tibshirani and colleagues [9.10] extend conformal prediction under covariate shift using likelihood-ratio weighting. Their exact guarantee requires the stated shift assumptions and known ratios; estimating those ratios or allowing conditional outcome drift introduces additional uncertainty rather than preserving the ordinary guarantee automatically.

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 new-population support to lie within reference support; the ratio is used only 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

$$w(x) = \frac{p_{\text{new}}(x)}{p_{\text{ref}}(x)} \tag{9.T13}$$

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.

$$r(x) = \mathbb{E}[\ell(Y, f(x)) \mid X = x] \tag{9.T14}$$

Now average that conditional risk over new inputs and substitute the input-density ratio. Absolute continuity is what allows this replacement without discarding any new-population mass.

$$\mathbb{E}_{\text{new}}[\ell(Y, f(X))] = \int r(x) p_{\text{new}}(x)\, dx \tag{9.T15}$$

$$\int r(x) p_{\text{new}}(x)\, dx = \int r(x) w(x) p_{\text{ref}}(x)\, dx \tag{9.T16}$$

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**

$$R_i = s(X_i, Y_i), \quad i = 1, \dots, m \tag{9.T18}$$

$$k = \lceil (m+1)(1-\alpha) \rceil \tag{9.T19}$$

$$\widehat{q} = \begin{cases} R_{(k)}, & k \le m, \\ +\infty, & k = m+1. \end{cases} \tag{9.T20}$$

$$\mathcal{C}(x) = \{ y : s(x, y) \le \widehat{q} \} \tag{9.T21}$$

First suppose all scores are distinct. Exchangeability makes the rank of the future score uniform among the calibration-count-plus-one positions. Membership occurs exactly when this rank does not exceed the selected index. The ceiling therefore yields the claimed lower bound. With tied scores, augment scores by independent continuous tie-breaking variables. Their ranks remain uniform; the unrandomized weak-inequality set contains the randomized accepted event, so ties can only make this construction more conservative.

$$\Pr\{\, Y_{m+1} \in \mathcal{C}(X_{m+1})\} \geq \frac{k}{m+1} \geq 1 - \alpha \tag{9.T22}$$

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.

$$s(x, y) = |y - \hat{f}(x)| \tag{9.T23}$$

$$\mathcal{C}(x) = [\hat{f}(x) - \widehat{q}, \hat{f}(x) + \widehat{q}] \tag{9.T24}$$

□

Coverage is marginal over the next pair, not a promise for each input, each subgroup, or each realized calibration set. Reusing calibration outcomes to tune the model or score invalidates this proof. Ordinary interpolated quantiles need not equal the required order statistic; very small calibration samples may require an infinite interval.

▶ 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

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** $i = 1, \ldots, m$

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.

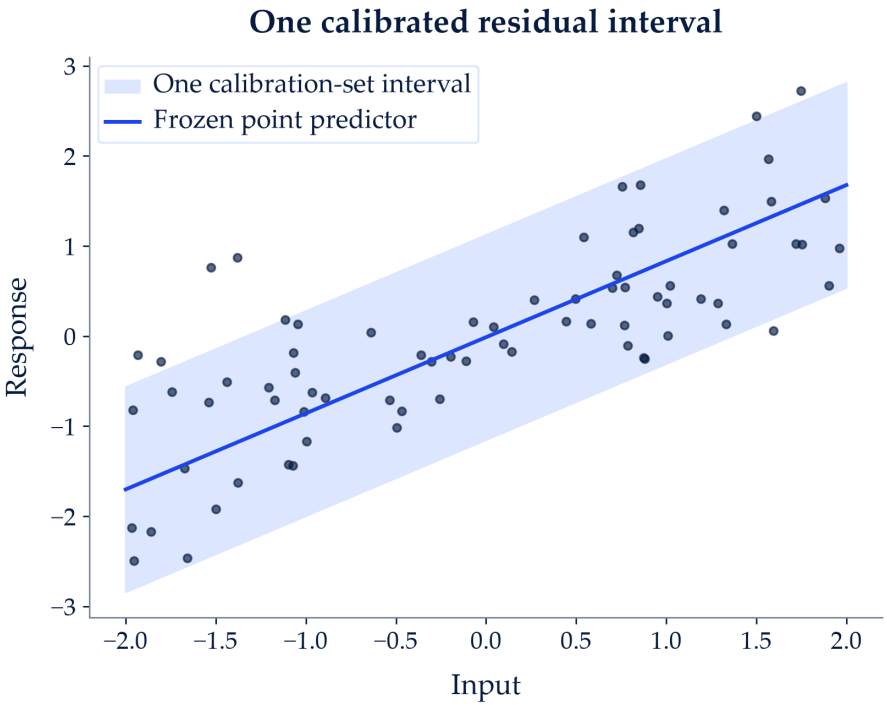


Figure 9.3. A point predictor is frozen before a separate set of 99 residuals calibrates its interval. For 90% target marginal coverage, the residual threshold is the 90th order statistic, not an interpolated percentile. This panel displays one calibrated interval and new noisy observations from a heteroscedastic population.

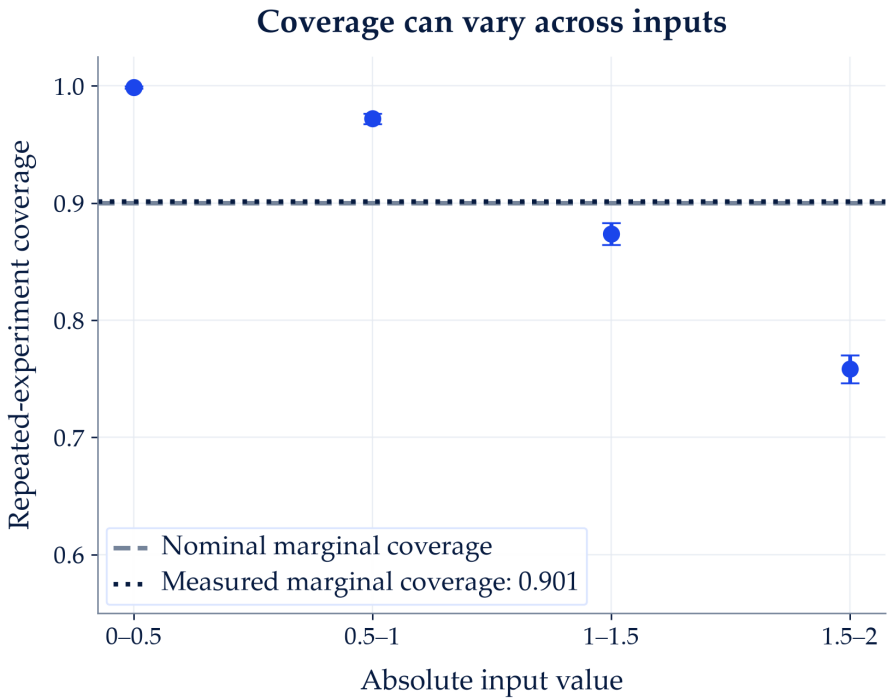


Figure 9.4. Twenty thousand independent calibration-and-test experiments under one heteroscedastic population yield about 90% marginal coverage while coverage varies with input magnitude. Error bars are approximate 95% Wilson intervals for the plotted group estimates. Split conformal supplies a marginal rank guarantee, not identical conditional coverage everywhere.

Notation for Figure 9.5

ROC, AUC	Receiver operating characteristic and area under that curve, respectively; the accompanying question compares ranking with probability calibration.

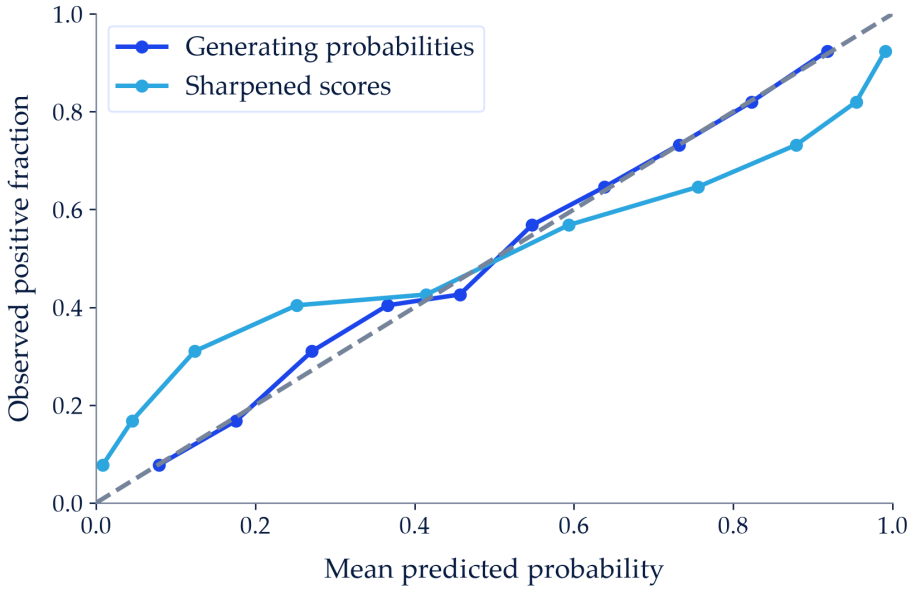


Figure 9.5. Synthetic outcomes are sampled from the generating probabilities. A monotonic sharpening preserves ranking but makes scores more extreme. The reliability plot reveals the resulting calibration distortion. Bin estimates remain noisy even when the generating probabilities are known.

Table 9.1. 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.

Tool	What it measures or changes	Interpretation limit
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.

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.
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

On decision validation, recalibration reduced Brier score from 0.025 to 0.022. Its final Brier score was higher than that selection-stage value, while still much lower than the prevalence reference. Independent final evaluation therefore added information beyond selecting a favorable calibration map.

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 notebook prints an actual fitting observation's original label and its recoded malignant indicator. For a final observation with reported probability p, compute its Brier contribution as (p minus the displayed zero-or-one indicator) squared, then verify that the mean of all recorded contributions equals the library Brier score.

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.

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

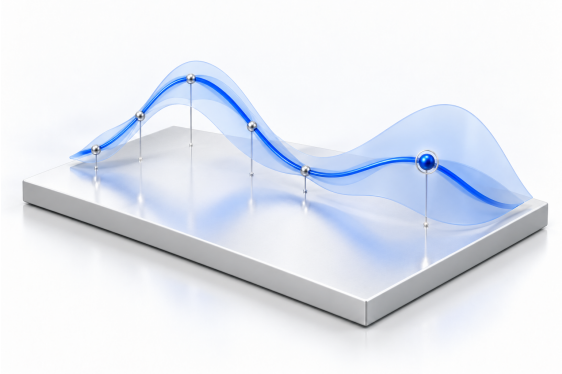
[9.8] Kumar, A.; Liang, P.; Ma, T. (2019). Verified Uncertainty Calibration. Advances in Neural Information Processing Systems 32. Source / DOI

[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

CHAPTER 10

Gaussian Processes And Experiment Selection



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.

Core route: 10.1, 10.2, 10.3, 10.4. Deepening route: 10.5.

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.

A point prediction answers one question: what response should we expect at this input? Choosing the next measurement requires another: how much could that answer change when evidence arrives? A Gaussian process, or GP, combines a model of relationships between inputs with a probability distribution over function values. Its uncertainty is conditional on modeling assumptions, not a certificate that those assumptions describe the world.

This chapter develops that model and then uses it for two different decisions. Active learning selects labels to improve a predictive model. Bayesian optimization selects evaluations to find a good objective value. Both can use a GP, but they need different acquisition rules and different evaluation protocols. The synthetic laboratory makes their distinction visible without suggesting that either method always beats random sampling.

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 assigns a multivariate normal distribution to function values at every finite collection of inputs. Its mean function gives prior expected responses; its covariance function, or kernel, links deviations around those means. For a chosen input list, evaluate the mean function into a vector and the kernel into a covariance matrix. These consistent finite-dimensional distributions define the function prior. The assumption concerns function values, not normally distributed input data.

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

Take two standardized inputs with zero prior mean, unit latent variances, and covariance 0.8. First, subtracting the latent values gives variance 1 plus 1 minus twice 0.8, or 0.4. Next, add independent observation noise of variance 0.09 to each value. Each observed variance becomes 1.09, while their covariance remains 0.8. The observed difference therefore has variance 0.58: both independent noise terms contribute. Positive covariance favors similar latent deviations, but independent measurement errors do not move together. These are specified prior assumptions, not statistics inferred 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.

10.2 RBF and Matérn kernels encode smoothness

The radial basis function kernel, often called the squared-exponential kernel, makes correlation decay with squared distance. Its amplitude sets the scale of latent variation. Its length scale determines how far information travels: a large value links distant inputs and favors slowly varying functions; a small value permits rapid changes. These statements depend on feature units. Standardizing a feature changes the numerical meaning of its length scale, so preprocessing belongs 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.

Kernel hyperparameters may be fitted by maximizing log marginal likelihood, which balances target fit with the covariance structure's complexity. This optimization can have local solutions and poorly identified combinations of length scale, amplitude, and noise. Bounded ranges, multiple starting points, and prior knowledge can help. A very short fitted length scale might reflect

genuine variation, unmodeled noise, or an inappropriate representation. It is evidence about a model fit, not a direct discovery of 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.

Complete supporting proof: Mathematical Supplement, A10 (in this book).

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.

10.3 Conditioning and interpreting posterior uncertainty

Conditioning a joint Gaussian distribution produces another Gaussian distribution. For a zero-mean GP, the posterior mean at a query is a weighted combination of training targets, with weights determined by covariance and noise. The posterior latent variance equals prior variance minus the part explained by the training observations. A nonzero mean requires subtracting training mean values before the calculation and adding the query mean back afterward; simply using the zero-mean expression would change 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.

Latent variance describes uncertainty about f at a query. For a future independent measurement with the same noise model, add the observation-noise variance. Intervals formed from these quantities are conditional Bayesian intervals. Their empirical coverage can be poor when the kernel, mean, noise model, or deployment distribution is wrong. Evaluate interval coverage and width on suitable held-out data, including relevant subgroups. Hyperparameters estimated as fixed values also omit uncertainty about those estimates.

Rasmussen and Williams [10.1] derive posterior prediction through conditioning a joint Gaussian distribution. The resulting uncertainty is conditional on the kernel, noise model, and fitted hyperparameters; it is not automatically calibrated for a misspecified model or an unfamiliar population. Titsias [10.7] develops an inducing-variable approximation by optimizing a variational bound on the GP marginal likelihood. This offers a principled route beyond dense exact inference, but approximation error must be distinguished from the observation-noise uncertainty used here.

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**

Write the finite joint distribution in block form. Independent training noise belongs only in the observation block. The cross-covariance still comes from the latent kernel.

$$\begin{bmatrix} y \\ f_* \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m_X \\ m_* \end{bmatrix}, \begin{bmatrix} A & C \\ C^T & B \end{bmatrix} \right) \tag{10.T8}$$

Subtract the best affine correction and examine the residual. It is jointly Gaussian with the observations because it is an affine transformation of their joint Gaussian vector.

$$r = f_* - m_* - C^T A^{-1}(y - m_X) \tag{10.T9}$$

$$\operatorname{Cov}(r, y) = C^T - C^T A^{-1} A = 0 \tag{10.T10}$$

Uncorrelated jointly Gaussian blocks are independent: their Gaussian characteristic function factors when its cross term is zero. Thus conditioning on observations does not change the residual distribution. Expanding its covariance gives the next expression; the two cross terms remove twice the covariance correction, while the correction's own variance adds one copy back.

$$S = \operatorname{Cov}(r) = B - C^T A^{-1} C \tag{10.T11}$$

$$f_* \mid y \sim \mathcal{N}\big(m_* + C^T A^{-1}(y - m_X), S\big) \tag{10.T12}$$

A residual covariance is positive semidefinite. The removed covariance is also positive semidefinite because the inverse of a positive-definite matrix is positive definite. Consequently every linear combination of query values has no more conditional variance than prior variance. This proves both the conditional formula and its variance ordering, including singular residual covariances. For a new independent noisy observation, add its noise covariance afterward. Implement the matrix products with linear solves; an explicit inverse is not required.

□

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**

Let the prior mean be zero and prior variance 1 at one input. Observe target 2 there with noise variance 0.25. First, the conditioning denominator is 1 plus 0.25, or 1.25, so the posterior mean is 2 divided by 1.25, or 1.6. Next, the latent variance is 1 minus 1 divided by 1.25, or 0.2. Finally, a new independent measurement adds noise variance 0.25, giving 0.45. Its standard deviation is approximately 0.671, versus 0.447 for the latent function. Thus noisy observations need not be interpolated, and measurement prediction remains less certain than latent-function inference.

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.

10.4 Active learning asks which label improves prediction

In pool-based active learning, a learner starts with labeled training examples and a separate collection of eligible but unlabeled inputs. It selects a pool item, receives its label from an oracle, adds that pair to training, and repeats. The oracle might be a human annotator, a simulator, or a measurement process. The goal is predictive quality for a defined target population under a labeling budget, rather than merely obtaining a large response value.

With constant observation noise, selecting the largest latent posterior variance is a simple uncertainty rule, not a guarantee of the greatest reduction in average prediction error. Check where uncertainty is reduced: an isolated candidate may have high variance but little covariance with deployment inputs. Integrated-variance or density-weighted criteria account for that relevance; diversity and acquisition cost introduce further constraints. With input-dependent noise, compare latent uncertainty separately from measurement noise, since a high observation variance can mainly reflect irreducible 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.

The covariance reduction after one additional observation is an outer product divided by the predictive observation variance. It is nonnegative in every direction and does not depend on the newly observed numerical label.

□ Proof

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.

$$c_+(u,v) = c(u,v) - \frac{c(u,x)\,c(x,v)}{c(x,x)+\tau^2} \tag{10.T13}$$

Setting both query inputs equal yields a squared numerator. Thus the variance decrement is never negative. For multiple queries the numerator becomes an outer product, proving the stronger covariance ordering. The label changes the posterior mean through its prediction residual, but it does not appear in this covariance update. This label independence relies on fixed hyperparameters and a Gaussian observation model.

$$\Delta(x) = \sum_{r=1}^{M} w_r \frac{c(u_r,x)^2}{c(x,x)+\tau^2} \tag{10.T14}$$

Summing the pointwise decrement with the fixed population weights proves the integrated-variance criterion. Maximizing candidate variance alone ignores both the candidate's correlation with the population and the noise in its measurement. An isolated input can have large variance and almost zero covariance with every population reference input, giving negligible integrated reduction. Conversely, a modestly uncertain point strongly correlated with many future cases can have greater value. Positive weights also show why the criterion cannot increase average model variance after acquisition.

□

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

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.

Notation for Figure 10.1

SD	Posterior standard deviation of the indicated latent response or future observation, measured in response units.

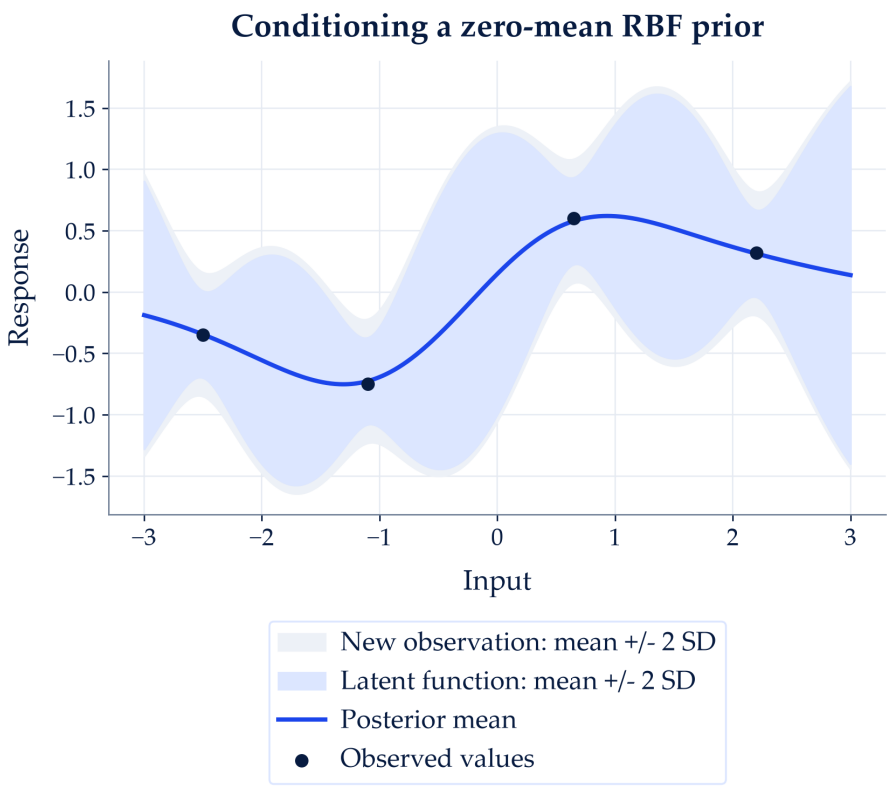


Figure 10.1. A zero-mean Gaussian-process prior with unit signal variance, RBF length scale 0.85 and independent observation-noise standard deviation 0.18 is conditioned on four labels. Latent-function and future-observation bands differ by the noise variance. Bands express the specified model, not empirical coverage under arbitrary shift.

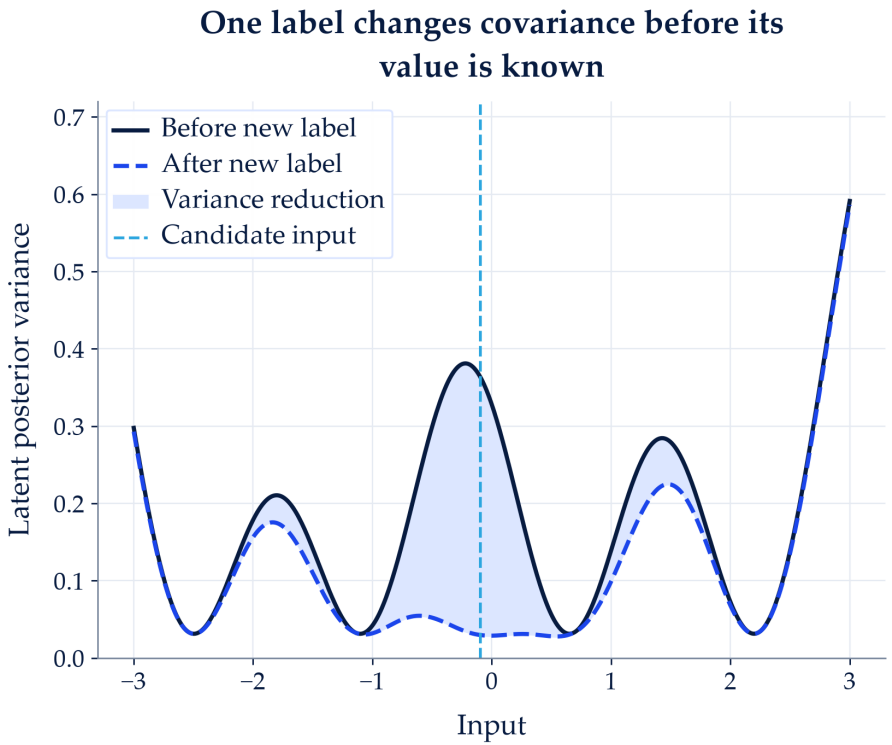


Figure 10.2. Acquiring a label at input -0.1 contracts the latent posterior variance by the exact rank-one covariance update. The reduction is independent of the not-yet-seen label when hyperparameters stay fixed. Direct numerical checks verify nonnegative variance and non-increase at every plotted input.

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 uses a surrogate model to choose evaluations of an expensive objective. Here we adopt a maximization convention: larger objective values are better. The acquisition function is cheaper to evaluate than the objective and combines the current posterior's information into a selection score. Upper confidence bound adds a multiple of latent standard deviation to the mean. The coefficient kappa controls optimism, but a chosen numerical value does not by itself create a universal confidence or regret guarantee.

Expected improvement averages how much a latent response exceeds an incumbent threshold. With a Gaussian posterior, its closed form uses the standard normal density and cumulative distribution functions. A nonnegative exploration offset xi can raise the improvement threshold. At zero posterior standard deviation, use the deterministic positive improvement limit instead of dividing by zero. The formula here treats the incumbent as fixed; noisy observations make the best observed target an optimistic and potentially misleading incumbent.

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.

Complete supporting proof: Mathematical Supplement, A7 (in this book).

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, use posterior means 0.8 and 0.6 and latent standard deviations 0.05 and 0.3. With kappa 2, adding twice each standard deviation

gives UCB scores 0.9 and 1.2, selecting B. For EI, fix the incumbent at 0.75 and improvement offset at zero. The mean improvements are 0.05 and negative 0.15; dividing by uncertainty gives standardized improvements 1 and negative 0.5. EI is approximately 0.054 and 0.059, again selecting B. With kappa zero, UCB instead selects A. These are assumed-posterior calculations for maximization, not experimental performance claims.

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

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

  $\mathcal{D} \leftarrow \mathcal{D} \cup \{(x, r)\}$

  $b \leftarrow \max\{b, r\}$

Algorithm 10.2 (continued)

$R \leftarrow \mathrm{Success}(R, x, r, n_0 + q, b)$

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. Only roundoff-sized negative variances may be clipped before taking square roots. 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.

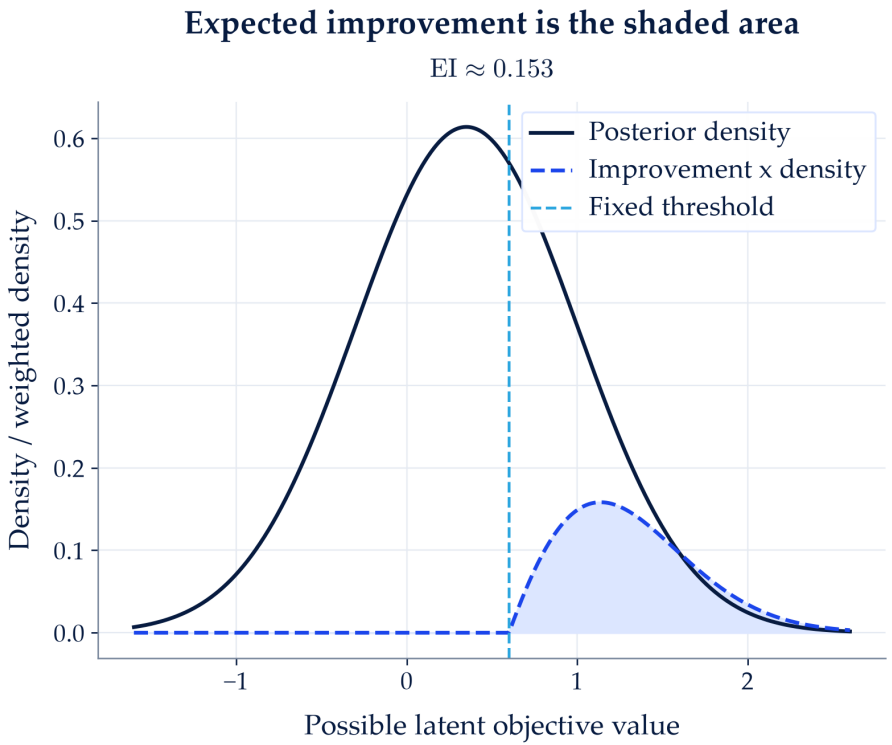


Figure 10.3. Expected improvement for maximization is the integral of positive improvement times posterior density. Here the Gaussian mean is 0.35, standard deviation 0.65 and fixed improvement threshold 0.6. The shaded area agrees with the analytical Gaussian-tail formula to numerical integration tolerance.

Notation for Figure 10.4

SD	Posterior standard deviation of the latent objective at a candidate, measured in objective units.

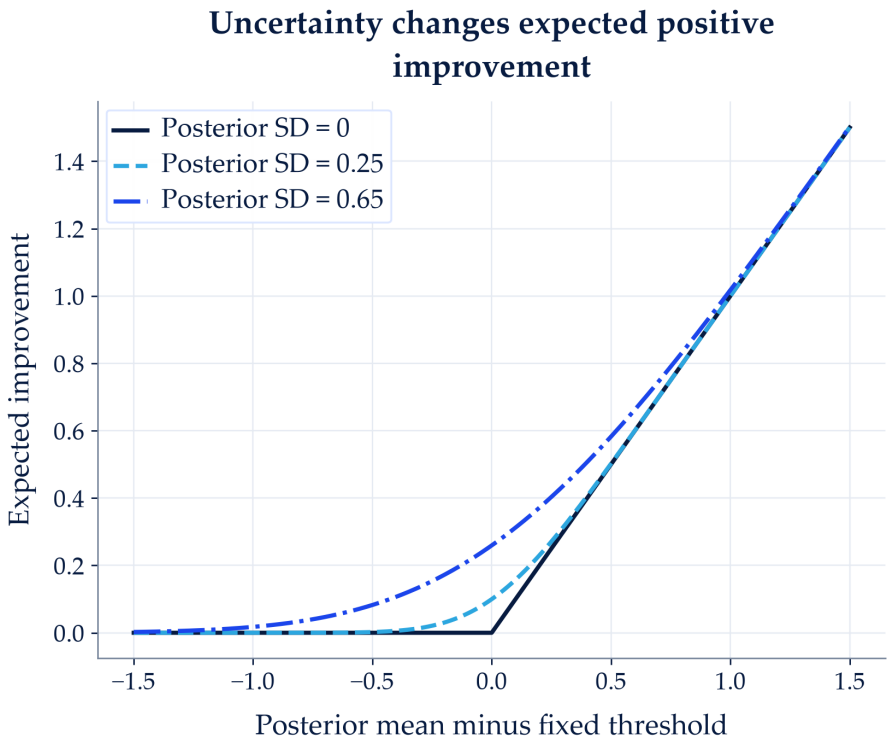


Figure 10.4. The Gaussian expected-improvement formula evaluated against posterior mean minus the same fixed threshold. Different standard deviations change the expectation; zero uncertainty gives the positive-part limit. These curves do not integrate uncertainty in a noisy incumbent.

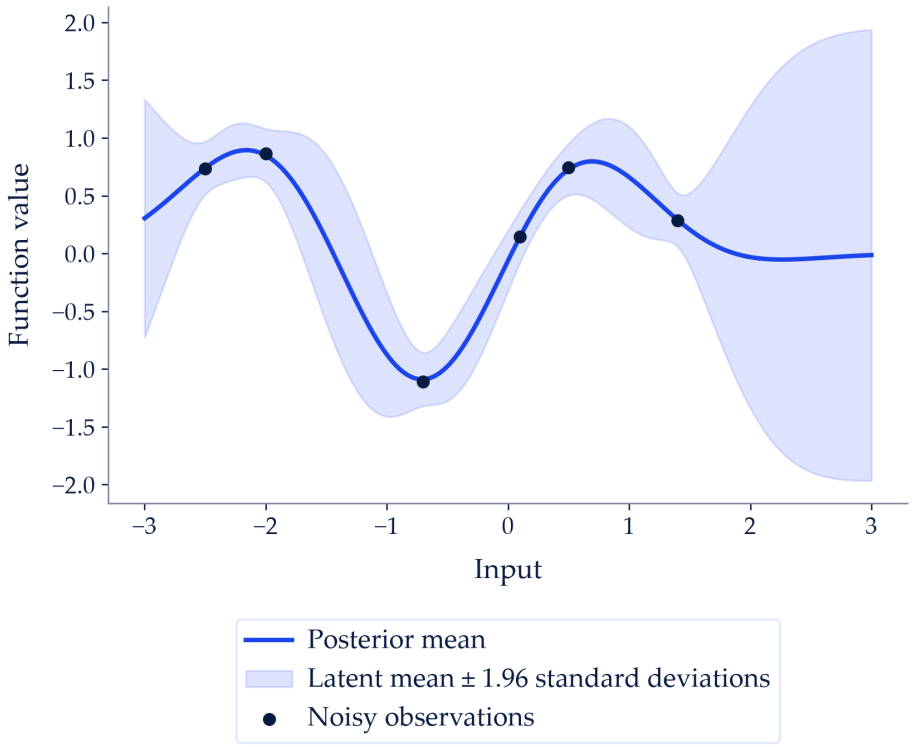


Figure 10.5. Actual zero-mean GP regression with a fixed RBF kernel and known synthetic observation-noise variance. The interval is for the latent function under that model, not for a future noisy measurement and not a guarantee under distribution shift.

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.

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 each pair, but the first pair was nearly tied. At the final label budget, the mean validation RMSE instead favored random acquisition: 59.638 versus 62.739. The finite samples and starting sets do not support a universal acquisition ranking.

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

The first revealed source row and its actual progression score are printed. Express that score using the initial eight-label mean and standard deviation, then reconstruct it in original target units. At a printed final prediction, verify that observation variance minus latent variance equals the declared standardized noise variance multiplied by the squared initial target scale.

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.

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

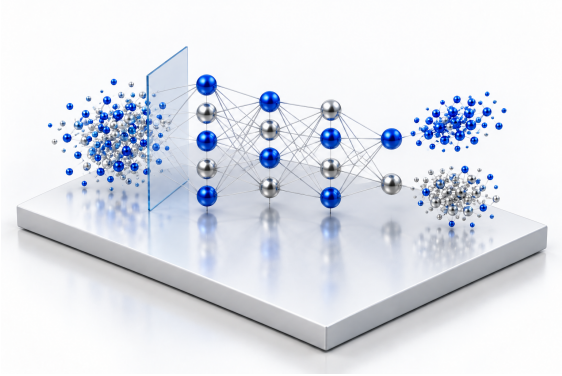
[10.7] Titsias, M. (2009). Variational Learning of Inducing Variables in Sparse Gaussian Processes. Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics, PMLR 5, 567-574. Source / DOI

[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

CHAPTER 11

Neural Network



A neural network extends familiar regression ideas by learning intermediate features. Trace one prediction and its derivatives before asking an automatic differentiation library to repeat the calculation.

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 same calculation numerically and with automatic differentiation.

Before you begin: Linear regression and gradient descent; Matrix multiplication, derivatives, and the chain rule; NumPy arrays and classification losses.

Core route: 11.1, 11.2, 11.3, 11.4, 11.5. Revisit the proofs after reproducing the Examples.

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 regression and logistic regression share a useful structure: fixed input features are combined by learned weights, then interpreted through a task-specific output rule. A neural network extends that structure by learning intermediate features as well as the final combination. The central question is therefore not how to abandon the earlier models, but how to compose them so that the representation can adapt to the target.

This chapter follows one calculation from input to prediction, from prediction to loss, and from loss back to every parameter. A small tanh regressor supplies a running numerical example. Its hand-derived derivatives are then checked by finite differences and reproduced with automatic differentiation. Initialization, minibatch training, regularization, and optimizer selection are developed in Chapter 12; here the objective is to understand precisely the function and gradient that those procedures will use.

11.1 From linear models to a neural network

A neural unit starts with the affine score already used in linear models. Its weights determine a direction in input space, and its bias sets an offset. An activation function converts that score into the unit's output. With an identity activation, the unit remains affine; with a sigmoid interpreted as a probability, it recovers the familiar binary logistic model. A hidden unit differs mainly in its role: its output becomes a feature for another learned operation rather than the final answer.

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.

Predictions and scalar targets will both be columns throughout the chapter. A flat target array and a prediction column can otherwise broadcast into an all-pairs residual matrix. The arithmetic may run successfully while comparing the wrong examples. Treat equal prediction and target shapes as part of the mathematical definition, not merely an implementation preference.

Rosenblatt's perceptron [11.1] is a foundational example of a learned decision rule built from weighted inputs. Goodfellow, Bengio, and Courville [11.4] develop the modern feedforward view in which composing affine maps with nonlinear activations produces learned intermediate representations.

Notation for equation (11.1)

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.

$$z = w^T x + b \tag{11.1}$$

One unit forms a scalar preactivation from an input column and a weight column of equal length.

Notation for equation (11.2)

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.

$$a = \phi(z) \tag{11.2}$$

The scalar activation becomes either an output or a learned feature passed to the next layer.

Notation for equation (11.3)

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.

$$Z = X\,W_1 + \mathbf{1}_n b_1^T \tag{11.3}$$

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 (11.4)

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.

$$H = \phi(Z) \tag{11.4}$$

Apply the activation independently to each entry. The hidden feature matrix retains the n-by-h shape.

Notation for equation (11.5)

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.

$$\widehat{y} = H\,W_2 + \mathbf{1}_n b_2 \tag{11.5}$$

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.

11.2 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 forward path becomes nonlinear at the activation, not at the matrix multiplication. Without that operation, substituting one affine layer into the next produces another affine map. With an activation, a hidden coordinate responds differently in different regions of the input space. Training changes both where those regions occur and how later layers combine them. This is the sense in which the hidden representation is learned.

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. Chapter 12 discusses how input scale and initialization affect the regions encountered during training.

A hidden activation and an output rule have different jobs. Our running regressor keeps an unrestricted scalar output. Binary classification instead commonly supplies a logit to a stable binary cross-entropy implementation, as

in Chapter 3; multiclass classification supplies one logit per class. Applying an extra sigmoid or softmax before a loss that already expects logits changes the objective. Representational capacity also differs from learnability: the constructive result below exhibits weights that approximate a continuous one-dimensional function, without claiming an optimizer will discover them.

Cybenko [11.2] 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 [11.7] empirically study rectifiers and their sparse activations. Leshno and colleagues [11.8] characterize approximation capacity under explicit activation assumptions and emphasize the role of thresholds; this is not a guarantee that training finds the required weights.

Notation for Figure 11.1

z	Real scalar preactivation on the horizontal axis; the lower panel is the ordinary derivative with respect to it.

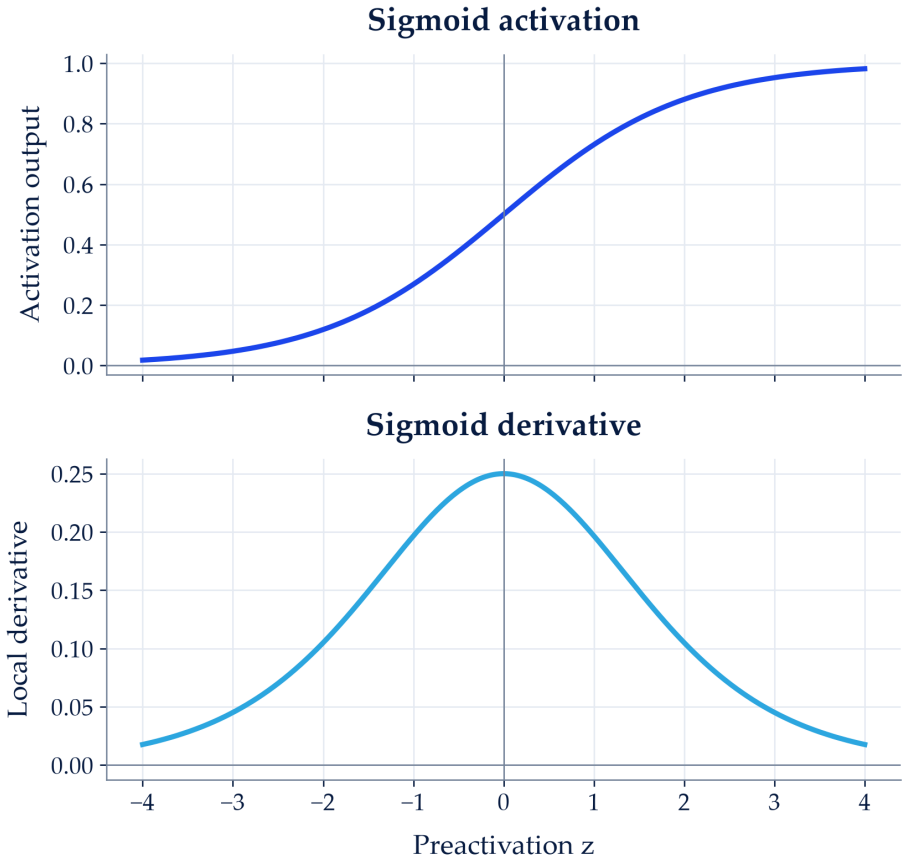


Figure 11.1. Sigmoid and its local derivative are shown alone so the scales remain explicit. The output lies between zero and one; the derivative peaks at 0.25 and tends to zero in either saturated tail.

Notation for Figure 11.2

z	Real scalar preactivation; the two panels show hyperbolic tangent and its derivative with respect to that argument.

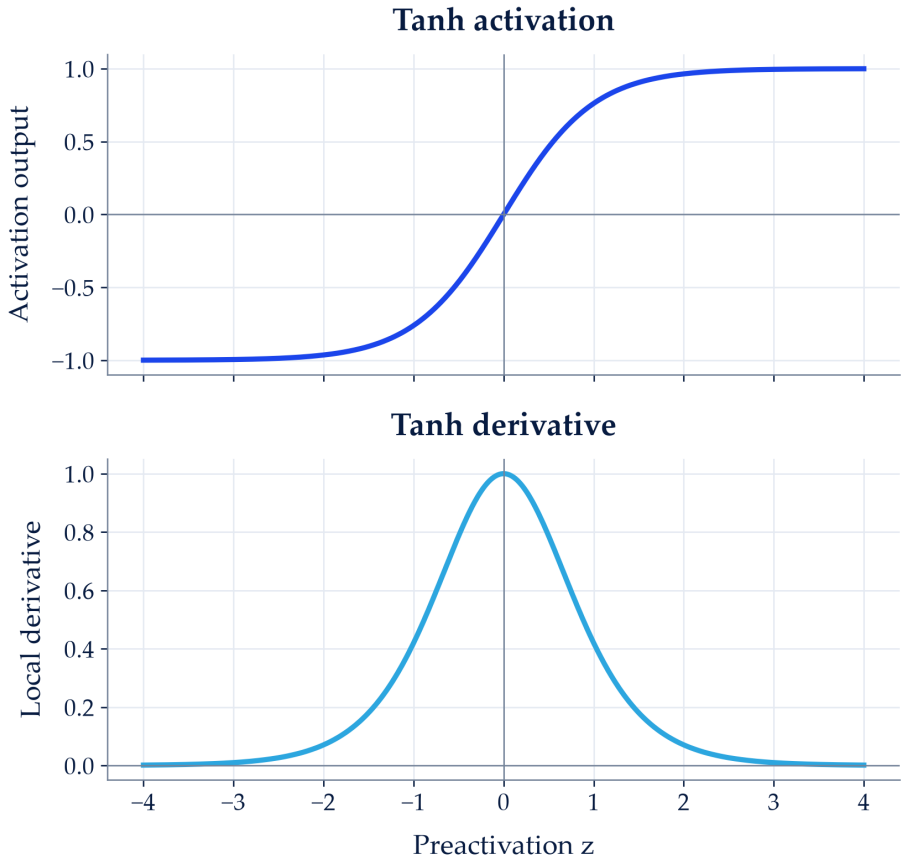


Figure 11.2. Tanh and its local derivative are shown alone so the scales remain explicit. The output lies between minus one and one; the derivative peaks at one and tends to zero in either saturated tail.

Notation for Figure 11.3

z	Real scalar preactivation; the classical ReLU derivative exists for nonzero values and is undefined at zero.

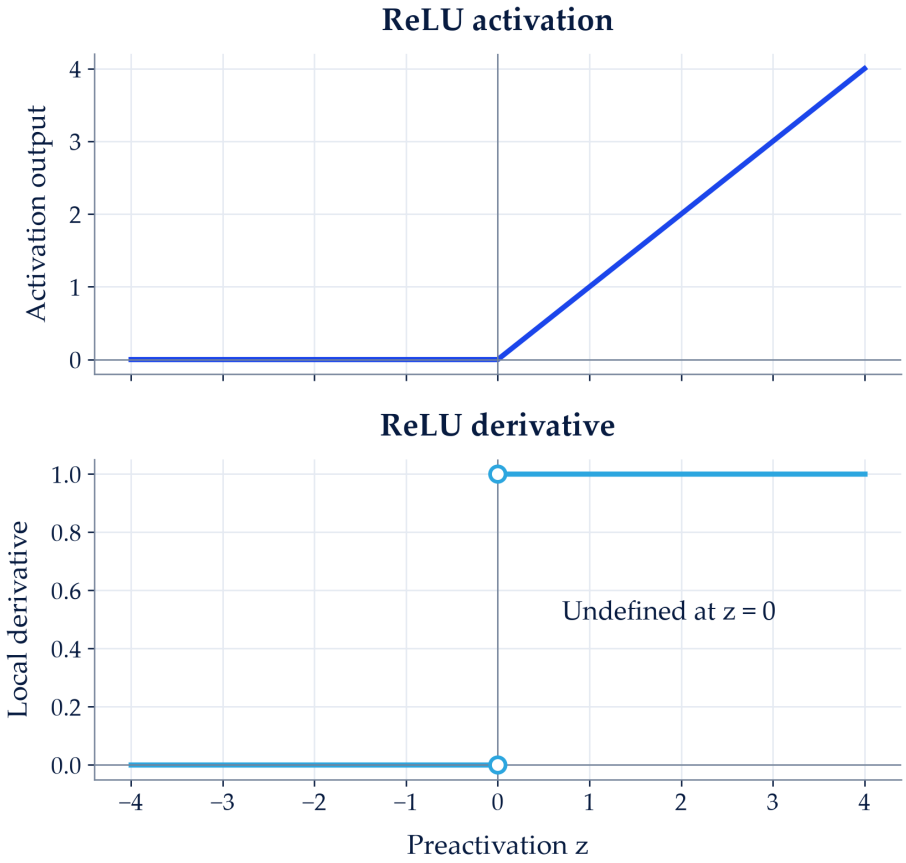


Figure 11.3. ReLU and its local derivative are shown alone so the scales remain explicit. The output is zero for negative inputs and equals the input for positive inputs. Its derivative is zero on the negative side and one on the positive side, and is undefined at zero. Automatic differentiation selects a declared convention at the kink.

Notation for equation (11.6)

z	Real scalar preactivation of one unit.
σ	Logistic sigmoid, taking values strictly between zero and one.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{11.6}$$

The sigmoid is a smooth bounded activation and also the probability link used in binary logistic regression.

Notation for equation (11.7)

z	Scalar preactivation at which the derivative is evaluated.
$\sigma, \sigma^{'}$	Logistic sigmoid and its ordinary derivative with respect to its scalar argument.

$$\sigma^{'}(z) = \sigma(z)\big(1 - \sigma(z)\big) \tag{11.7}$$

Saturation makes the derivative small even though the function remains differentiable.

Notation for equation (11.8)

z	Real scalar preactivation of one unit.
$\tanh$	Hyperbolic tangent, taking values strictly between negative one and one.

$$\tanh(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}} \tag{11.8}$$

Tanh is an odd, zero-centered activation with outputs strictly between negative one and one.

Notation for equation (11.9)

z	Scalar hidden preactivation.
$\tanh, \tanh^{'}$	Hyperbolic tangent and its derivative with respect to the preactivation.

$$\tanh^{'}(z) = 1 - \tanh^{2}(z) \tag{11.9}$$

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 (11.10)

z	Real scalar preactivation.
ReLU	Rectified linear unit.
$\max$	Maximum of the displayed scalar arguments.

$$\mathrm{ReLU}(z) = \max(0, z) \tag{11.10}$$

The rectified linear unit is continuous and piecewise affine.

Notation for equation (11.11)

z	Scalar preactivation, restricted here to nonzero values.
ReLU'	Ordinary derivative of the rectified linear unit away from its corner.

$$\mathrm{ReLU}'(z) = \begin{cases} 0, & z < 0, \\ 1, & z > 0. \end{cases} \tag{11.11}$$

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 11.1: 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.

Without nonlinear activations, any depth represents an affine map. With ReLUs, each fixed activation pattern defines a possibly empty polyhedral region on which the network is affine. Consequently a finite ReLU network is continuous and piecewise affine.

□ **Proof**

$$a_0 = x \tag{11.T1}$$

$$a_l = W_l{}^T a_{l-1} + b_l \tag{11.T2}$$

The first layer is affine. If the preceding activation is an affine function of the original input, substituting it into the next affine layer produces another affine function. Induction through the finite layers proves collapse, including the propagated biases. Additional depth may alter parameterization and optimization without enlarging this function class.

$$a_l = D_l(W_l{}^T a_{l-1} + b_l) \tag{11.T3}$$

For a fixed collection of diagonal activation matrices, the same induction proves that every hidden activation is affine in the original input. Consistency of the chosen pattern requires each preactivation to be nonnegative when active and nonpositive when inactive. Each requirement is a linear inequality in the original input within the fixed pattern. Their intersection is a polyhedron, which may be empty. There are finitely many patterns because there are finitely many hidden units. These regions cover the input space, with possible boundary overlap.

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.

◇ Theorem 11.2: A constructive one-dimensional approximation result

Notation for this result

$[a, b]$	Closed bounded input interval with distinct ordered endpoints.
t_j, m	Ordered grid points and positive interval count; j runs from zero through m.
f	Continuous real target function on the specified interval.
s_j, j	Interval-j interpolation slope; j runs from zero through m-minus-one.
p, x	Polygonal approximant and input in the closed target interval.
$\mathrm{ReLU}(u)$	Maximum of zero and real u; shifted hinge at each interior grid point.
ω_f, δ	Target modulus of continuity and nonnegative maximum input separation.
$u, v, \sup$	Two interval inputs and supremum over eligible displayed pairs.

Assumptions. The target is continuous on a closed bounded interval. Choose a finite ordered grid including both endpoints and interpolate target values linearly between neighboring grid points. Only uniform function-value approximation is claimed.

The polygonal interpolant can be represented exactly by a finite sum of shifted ReLUs plus an affine term. Making the largest grid gap sufficiently small gives any prescribed positive uniform error tolerance.

Complete supporting proof: Mathematical Supplement, A8 (in this book).

This finite constructive result explains representational capacity in one dimension. It does not prove a general multivariate universal-approximation theorem or guarantee that training data and gradient descent recover the constructed weights.

▶ 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.

Notation for Figure 11.4

x	Dimensionless scalar input from negative two through two in this constructed interpolation example.
sin	Sine function with its dimensionless argument interpreted in radians; it specifies the known continuous target, not a fitted model.

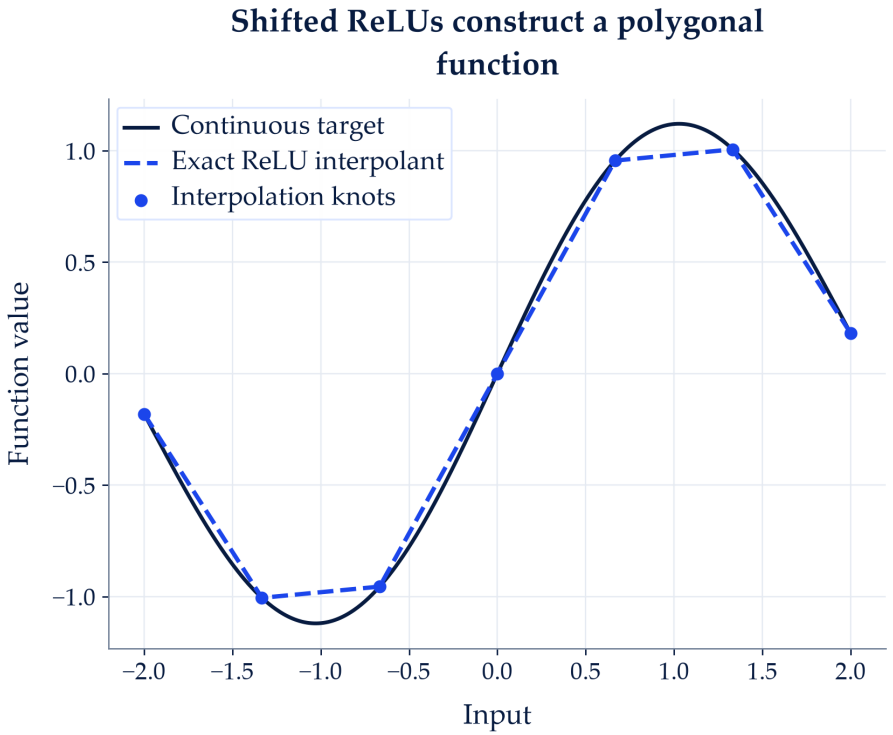


Figure 11.4. Samples of the known target sin(1.6x) + 0.12x on seven uniform knots are interpolated exactly by a sum of shifted ReLU hinges. The hinge formula is checked against piecewise-linear interpolation. This constructive representation is not a trained model or a generalization result.

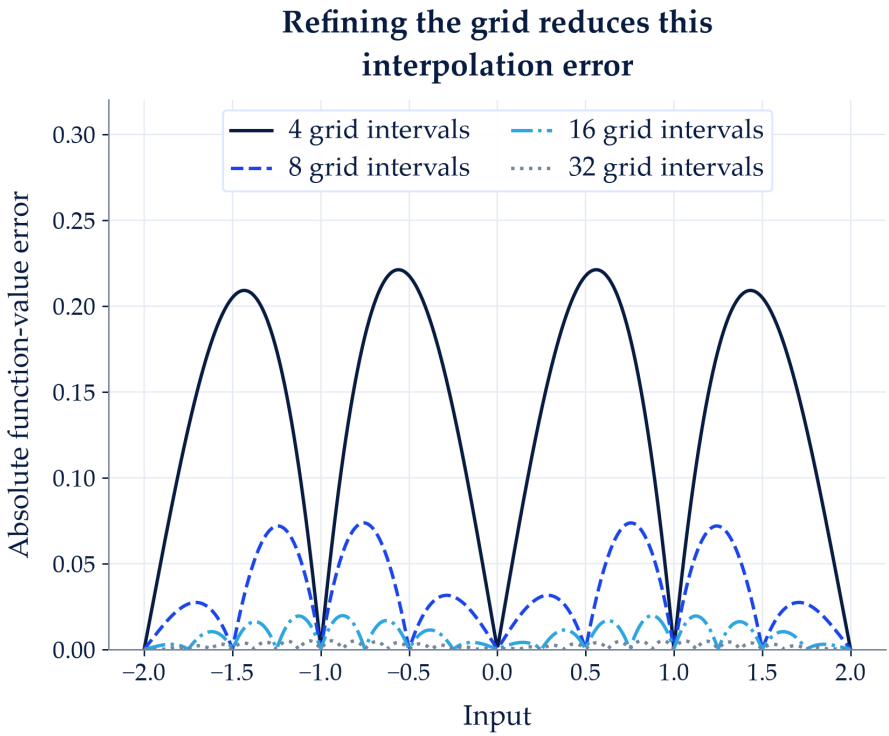


Figure 11.5. Absolute approximation error of piecewise-linear ReLU interpolants as the uniform grid is refined from four to 32 intervals. The error decreases for this specified smooth target. The plot illustrates a constructive approximation argument, not an empirical deep-network training guarantee.

Optional interactive companion | Spatial representations

A convolutional neural network (CNN) reuses local filters across an image. Here a kernel is a small spatial filter, not the similarity kernel of Chapter 6. Inspect how a filter changes an activation map, then vary stride (the filter's step), padding (added border values), or pooling (local aggregation). Connect these transformations to the distinction between a representation and a final prediction.

Open CNN Lab

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

11.3 From prediction error to backpropagation

A prediction becomes a learning problem only after a scalar loss has been specified. Continue with one real-valued target per example and mean squared error. The mean is over examples, not hidden units, and its normalization appears once in the derivative. For a different output task, replace the output loss and its initial derivative; the backward propagation through the hidden layers follows the same chain-rule logic.

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 [11.3] 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 (11.12)

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.

$$L = \frac{1}{n}\sum_{i=1}^{n}(\widehat{y}_i - y_i)^2 \tag{11.12}$$

The scalar objective averages squared residuals over the batch using one scalar target per example.

Notation for equation (11.13)

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.

$$G_2 = \frac{2}{n}(\widehat{y} - y) \tag{11.13}$$

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 11.3: 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.

Every parameter derivative follows by collecting coefficients in the differential of the scalar loss. Shared biases sum over examples; averaging the already averaged incoming gradient again would be incorrect.

□ Derivation

Use exactly the forward computation introduced in Section 11.1, with tanh applied to each hidden preactivation. Its three stages are recorded separately so each backward derivative can be matched to a forward operation.

$$Z = X W_1 + \mathbf{1}_n b_1^T \tag{11.T8}$$

$$H = \tanh(Z) \tag{11.T9}$$

$$\widehat{y} = H W_2 + \mathbf{1}_n b_2 \tag{11.T10}$$

$$L = \frac{1}{n} \|\widehat{y} - y\|_2^2 \tag{11.T11}$$

$$G_2 = \frac{2}{n} (\widehat{y} - y) \tag{11.T12}$$

$$dL = G_2^T \, d\widehat{y} \tag{11.T13}$$

Apply the product rule to the prediction: its change is the hidden change times output weights, plus hidden activations times the weight change, plus the broadcast bias change. Substitute this sum into the loss differential. The coefficient of each output-weight change is the corresponding hidden activation weighted by the incoming error, summed over examples.

$$\nabla_{W_2} L = H^T G_2 \tag{11.T14}$$

$$\frac{\partial L}{\partial b_2} = \mathbf{1}_n^T G_2 \tag{11.T15}$$

The remaining coefficient of a hidden-activation change is the outer product of the incoming prediction gradient and output weights. Tanh differentiates coordinatewise. Multiplying by its coordinatewise derivative gives the preactivation gradient; multiplying by the input transpose then sums each input-weight contribution across the batch.

$$G_1 = (G_2 W_2^T) \odot (1 - H \odot H) \tag{11.T16}$$

$$\nabla_{W_1} L = X^T G_1 \tag{11.T17}$$

$$\nabla_{b_1} L = G_1^T \mathbf{1}_n \tag{11.T18}$$

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 11.1, 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 11.1. One tanh-network gradient step

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 each gradient with exactly its parameter's shape; fail before updating if a loss or required gradient is nonfinite. The parameter loop visits W1, b1, W2 and b2 exactly once.

Algorithm 11.1

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

01 **Forward** $Z \leftarrow X W_1 + \mathbf{1}_n b_1^T$

Algorithm 11.1 (continued)

Activate $H \leftarrow \tanh(Z)$
Affine $\widehat{y} \leftarrow H\,W_2 + \mathbf{1}_n b_2$
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.

Notation for Figure 11.6

x, y	Scalar input and target for the one-example, one-hidden-unit regressor in this figure; their values are two and one.
w_1, b_1, w_2, b_2	Scalar hidden weight, hidden bias, output weight and output bias, respectively. Lowercase weights specialize the earlier matrices to one hidden unit and one input.
$z, h, \widehat{y}$	Scalar hidden preactivation, tanh hidden activation and prediction computed in the forward column; h here is an activation value, not hidden-layer width.

L	Squared loss for the one-example batch; its mean equals its sum because the batch contains one example.
g_y, g_h, g_z	Loss derivatives with respect to the prediction, hidden activation and hidden preactivation, respectively. The subscript y denotes the prediction sensitivity, not differentiation with respect to the fixed target.

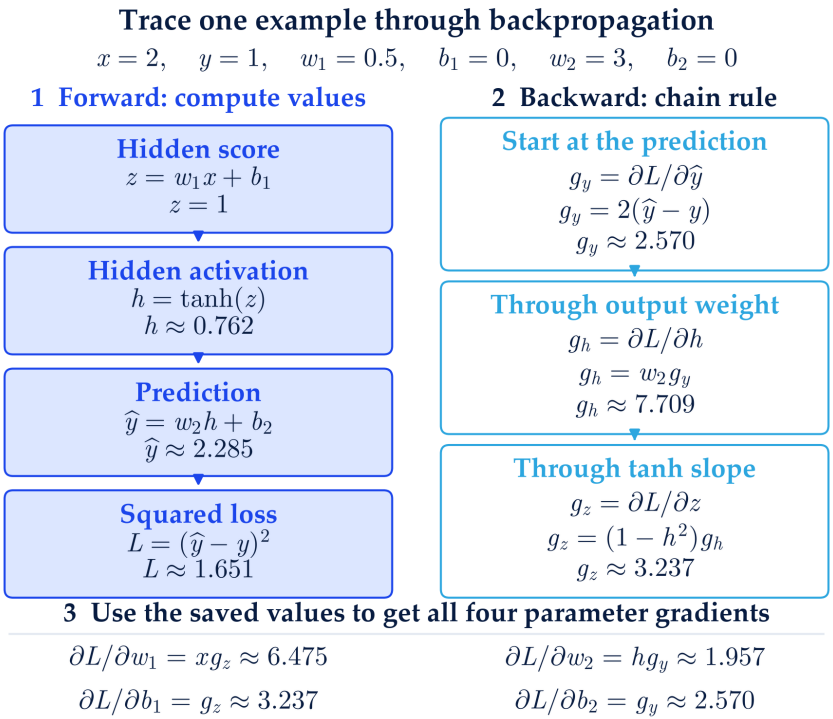


Figure 11.6. Read the left column downward first, then the right column downward. Backward sensitivities reverse the output-affine and tanh operations. This is the same one-example tanh regressor used in Sections 11.1 and 11.3; all numbers are computed from unrounded forward values. The bottom block gives both weight and both bias derivatives. For a batch, use the preceding matrix derivation: introduce the mean-loss factor once and sum shared-parameter contributions over examples.

From A Scalar Regressor To A Multiclass Neural Classifier

The preceding derivation sends a mean-squared-error derivative into one unrestricted output. Chapter 12 instead trains a classifier with several mutually exclusive classes. The hidden computation and its chain rule remain the same; the output has one real score, or logit, per class, and the loss must compare the whole score vector with the observed class. Here the assumptions are finite logits, exactly one observed class per example, no class weights or label smoothing, and an ordinary arithmetic mean over a nonempty batch. A multilabel task, where several labels may simultaneously be true, needs a different outcome model.

C, j, k Class count C at least two; j and k index classes from zero through C-minus-one.

$z \in \mathbb{R}^C,\ p_j$ One example's finite logit column and its positive probability assigned to class j.

$\exp$ Natural exponential; the shared denominator normalizes all class probabilities together.

$$p_j = \frac{\exp(z_j)}{\sum_{k=0}^{C-1} \exp(z_k)} \tag{11.B1}$$

The softmax probabilities are positive and sum to one. Increasing one logit changes every class probability because they share a denominator. Adding the same scalar to all logits changes no probability: multiply numerator and denominator by the same exponential and cancel. Consequently logits are relative evidence, not individually identified probabilities. With two classes, the positive-class probability is the sigmoid of the difference between the two logits, connecting this construction to Chapter 3.

y Observed integer class index from zero through C-minus-one; its probability is the yth coordinate of p.

$\ell(z, y), \log$ One example's categorical negative log likelihood, using the natural logarithm and therefore measured in nats.

$$\ell(z, y) = -\log p_y = \log\left(\sum_{k=0}^{C-1} e^{z_k}\right) - z_y \tag{11.B2}$$

This is cross-entropy against a one-hot target: only the observed class contributes to the sum of negative target-weighted log probabilities. It rewards the probability placed on that class, not merely whether its logit is largest. Directly evaluating large exponentials can overflow, and taking a logarithm after rounding a small probability to zero can produce an artificial infinity. Algebraically shift before exponentiating. Let m be the largest logit for this example; then every exponent below is nonpositive and at least one exponential is one.

$m = \max_k z_k$ Largest of the example's finite logits, used only as a numerical shift; m is not a class label.

$$\ell(z, y) = (m - z_y) + \log\left(\sum_{k=0}^{C-1} e^{z_k - m}\right) \tag{11.B3}$$

Proof of the output gradient. Differentiate the smooth log-sum-exp expression in (11.B2): the derivative of the outer logarithm divides by the exponential sum, and differentiating that sum with respect to one logit leaves its exponential. Differentiating the observed-class logit contributes minus one only in that class. Thus the loss gradient has a particularly simple form.

$\mathbf{1}[j = y]$ One for the observed class and zero otherwise; the jth coordinate of its one-hot target.

$$\frac{\partial \ell}{\partial z_j} = p_j - \mathbf{1}[j = y] \tag{11.B4}$$

The gradient sums to zero because the probabilities and one-hot target each sum to one. A small negative-gradient step lowers competing logits and raises the observed-class logit. The proof uses the smooth expression before shifting, so ties in the maximum used for numerical stabilization do not create a corner in the loss. Softmax alone has a matrix Jacobian, but its composition with this cross-entropy simplifies to probability minus target; one must not substitute that derivative for an unrelated loss.

Table 11.B1. A verified three-class example: target class 1

Class	Logit	Probability	Logit derivative
0	1002	0.665	0.665
1	1001	0.245	-0.755
2	1000	0.090	0.090

Subtracting the maximum gives shifted logits (0, -1, -2), so the normalization sum is approximately 1.503. The target-class loss is approximately 1.408 nats. The largest score belongs to class 0, so the predicted class is wrong even though the loss and all gradients are finite. The same probabilities, loss, and derivatives result from logits (2, 1, 0). The accompanying numerical check confirms this shift invariance, compares the gradient against central differences, and checks it against PyTorch in double precision. Displayed decimals are rounded only after those checks.

Batch shapes and the connection to backpropagation. Keep examples as rows. The hidden matrix H has n rows and h columns, while the new output weights W-out have h rows and C columns and the bias b-out is a length-C column. The logits Z-out and their probability matrix P both have n rows and C columns. Integer targets are stored as a length-n array; the mathematical one-hot matrix Y has n rows and C columns but need not be allocated by the implementation.

n, h, H Positive batch size, hidden width, and n-by-h hidden-activation matrix from the preceding network.

$W_{\text{out}}, b_{\text{out}}, Z_{\text{out}}$ h-by-C output weights, length-C bias column, and n-by-C logit matrix.

$\mathbf{1}_n$ Length-n column of ones, broadcasting the shared class biases across examples.

$$Z_{\text{out}} = H\,W_{\text{out}} + \mathbf{1}_n b_{\text{out}}^T \tag{11.B5}$$

i, y_i, L Example index, its integer class target, and scalar mean categorical loss over the batch.

$Z_{\text{out},i,:}$ All C logits in example i's row, interpreted as the example's score vector.

$$L = \frac{1}{n} \sum_{i=1}^{n} \ell(Z_{\text{out},i,:}, y_i) \tag{11.B6}$$

P, Y n-by-C rowwise softmax probabilities and matching one-hot targets; row i of Y has one in its observed-class column and zeros elsewhere.

G_{out} n-by-C derivative of the mean loss with respect to logits; the division by n occurs here once.

$$G_{\text{out}} = \frac{P - Y}{n} \tag{11.B7}$$

$\nabla_{W_{\text{out}}} L$ h-by-C output-weight gradient, obtained by summing the feature-weighted errors over examples.

$$\nabla_{W_{\text{out}}} L = H^T G_{\text{out}} \tag{11.B8}$$

$\nabla_{b_{\text{out}}} L$ Length-C shared-bias gradient; do not average this already-normalized signal again.

$$\nabla_{b_{\text{out}}} L = G_{\text{out}}^T \mathbf{1}_n \tag{11.B9}$$

G_H n-by-h derivative with respect to hidden activations, ready for the preceding hidden-layer chain rule.

$$G_H = G_{\text{out}} W_{\text{out}}^T \tag{11.B10}$$

These last three identities follow by expanding the differential of the output affine map and collecting coefficients, exactly as in the scalar derivation. For tanh hidden units, multiply G-H entrywise by one minus H squared, then use the input transpose for hidden weights and sum for hidden biases. Every derivative is evaluated at the same unchanged weights. This establishes the complete bridge: replace the scalar MSE output derivative by (P minus Y)/n and widen the output matrix, while preserving the hidden backpropagation logic.

Implement and audit the multiclass boundary

01. Assert finite floating-point logits with shape (n, C), integer targets with shape (n,), and every target in 0 through C-minus-one.

02. Pass logits directly to cross_entropy with mean reduction. Do not apply a separate softmax before a loss that expects logits.

03. For reporting probabilities, apply softmax across the class axis. For class predictions, the largest logit and largest probability select the same class; declare a tie rule.

04. On a tiny fixed batch, compare the stable manual loss and (P minus Y)/n with autograd and finite differences. Check every logit-gradient row sums to zero.

05. If class weights, ignored targets, label smoothing, or multilabel outputs are introduced, derive their reduction and target convention separately; the plain mean formula above no longer applies unchanged.

Worked normalization check. In the supplied two-example, three-class network calculation, duplicating both examples makes a four-example batch. Each repeated row then carries half its earlier logit-gradient contribution, while the sum forming each shared-parameter derivative is unchanged. The mean loss is unchanged too. A missing mean factor instead doubles the gradients. Executed verification confirms this invariance and checks the hidden-weight, output-weight, and both bias gradients against automatic differentiation.

PyTorch, CrossEntropyLoss documentation: logits, integer class targets, and reduction conventions. Source

Goodfellow, Bengio, and Courville (2016), Deep Learning, Section 6.2.2.3: softmax output units and categorical negative log likelihood. Source

11.4 Checking derivatives independently

A hand derivation and an implementation can make the same mistake, so the gradient needs an independent check. Hold the data and all nonparameter state fixed, move one parameter slightly in both directions, and compare the resulting loss difference with the analytic derivative. Central differences test the implemented scalar objective, rather than a remembered formula for what that objective was intended to be.

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.

Agreement establishes that the forward objective and its derivative are consistent. It does not establish that the objective matches the prediction task: an unintended all-pairs loss can have perfectly consistent derivatives. Retain the shape assertions and hand-computed forward values from Section 11.1. Only after these checks should a small negative-gradient step be used as an optimization smoke test.

Griewank and Walther [11.6] 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 (11.14)

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.

$$\frac{\partial L}{\partial \theta_j} \approx \frac{L(\theta + \varepsilon e_j) - L(\theta - \varepsilon e_j)}{2\varepsilon} \tag{11.14}$$

Perturb one coordinate of the flattened parameter vector while holding every other coordinate and the data fixed.

Notation for equation (11.15)

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.

$$\nabla L(\theta)^T v \approx \frac{L(\theta + \varepsilon v) - L(\theta - \varepsilon v)}{2\varepsilon} \tag{11.15}$$

A fixed direction checks a linear combination of parameter derivatives using only two loss evaluations.

◇ Proposition 11.4: Why smaller finite-difference steps are not always better

Notation for this result

L, θ, v	Deterministic scalar loss, length-P parameter column, and fixed unit-length direction in the same parameter space.
P, t, q	Parameter count, scalar displacement, and loss restricted to that displacement line.
$q'(0)$	Derivative of the restricted loss at zero displacement.
$\nabla L(\theta)$	Length-P parameter gradient at the unperturbed point.
ε, M	Positive perturbation size and an upper bound on the absolute third derivative throughout the perturbation interval.
D_{num}	Central-difference value formed from the two numerically evaluated losses.
δ	Upper bound on absolute error in either individual loss evaluation.

Assumptions. The scalar loss restricted to the tested parameter-space direction has a continuous third derivative near the current parameters. Each computed loss has an absolute numerical error bounded by a known or hypothetical nonnegative tolerance.

Central differences have a truncation error proportional to the square of the step, while bounded evaluation error is amplified inversely with the step. This explains a useful intermediate range of perturbation sizes.

□ **Proof**

$$q(t) = L(\theta + tv) \tag{11.T19}$$

$$q^{'}(0) = \nabla L(\theta)^{T} v \tag{11.T20}$$

Expand the restricted loss at a positive and a negative step using Taylor's theorem with a third-derivative remainder. Subtract the two expansions. The constant and quadratic terms cancel, while the two third-order remainders remain. Divide by twice the step and use the derivative bound.

$$\left| \frac{q(\varepsilon) - q(-\varepsilon)}{2\varepsilon} - q^{'}(0) \right| \leq \frac{M\varepsilon^2}{6} \tag{11.T21}$$

Now replace each exact loss by its numerical evaluation. Their two errors can have opposite signs, so the difference error is at most twice the evaluation tolerance. Division by twice the step gives the additional inverse-step bound.

$$|D_{\text{num}} - q^{'}(0)| \leq M\varepsilon^2/6 + \delta/\varepsilon \tag{11.T22}$$

When both constants are positive, differentiating this upper bound balances its increasing truncation term against its decreasing numerical-error term, producing a cube-root scale for the useful step. The exact best step remains problem dependent because the bounds are generally unknown and floating-point error is not constant across evaluations. Testing a range of steps is therefore more informative than demanding agreement at one extremely small value. At ReLU boundaries the differentiability assumption fails; a mismatch there need not expose a backpropagation bug. Random masks or changing data invalidate the deterministic comparison altogether.

□

A derivative check establishes consistency between an implementation and its own loss, not generalization, task correctness, or convergence of its optimizer.

▶ 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 11.2. Central-difference gradient audit

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 theta0 and cannot mutate it. Store records the coordinate, step, discrepancies and flags; any nonfinite value receives a failure flag, never a passing comparison.

Algorithm 11.2

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

Copy $\theta_0 \leftarrow \mathrm{Copy}(\theta)$
$g \leftarrow \nabla L(\theta_0)$
Assert $\mathrm{shape}(g) = \mathrm{shape}(\theta_0)$
$T \leftarrow [\,]$
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.
        $b \leftarrow [\Delta \le \tau_a + \tau_r \max\{|d|, |g_j|\}]$
        $T \leftarrow \mathrm{Store}(T, j, \varepsilon, d, \Delta, b)$
    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.

11.5 Automatic differentiation of the same network

Automatic differentiation removes the need to write a custom backward formula for every supported composition. During the forward pass it records operations and the values needed by their local derivative rules. Reverse mode starts with the scalar loss and propagates sensitivities backward through the executed graph. It performs the same calculation derived in Section 11.3; it is neither symbolic manipulation into a closed-form expression nor repeated finite-difference evaluations.

Implement the running network with gradient-enabled parameter tensors, compute tanh and the mean squared loss, and request a backward pass. The resulting four derivatives should match the hand calculation and the independent finite differences. This three-way agreement connects the mathematical model, the numerical program, and the differentiation engine. It also makes a copied parameter orientation or an unintended broadcast visible before a larger training loop hides the error.

A leaf parameter's gradient field usually accumulates contributions. Two fresh forward-and-backward passes at unchanged parameters therefore add the same derivative twice unless the field is cleared. Accumulation is useful for several losses or microbatches, but an ordinary per-batch update must deliberately begin with an empty gradient buffer. The equation below describes that storage behavior, not a change in the mathematical derivative.

Graph connectivity remains the programmer's responsibility. Detaching a needed intermediate, converting it to a host scalar, or using unsupported operations can sever a path to a parameter. In-place updates can overwrite values needed for backward. Evaluation mode and disabling gradient recording are separate controls: one changes the behavior of certain layers, while the other controls graph construction. Chapter 12 builds a complete training procedure around this now-verified forward and backward computation.

Baydin and colleagues [11.5] survey automatic differentiation as computation on programs. Griewank and Walther [11.6] explain its forward and reverse

accumulation rules, which is why an autograd result should match the hand-derived gradient of the same computation rather than a different loss or reduction. Paszke and colleagues [11.9] describe PyTorch's imperative execution architecture. Automatic differentiation operates on the implemented program, so it does not replace checks of graph connectivity, tensor shapes, and loss normalization.

Notation for equation (11.16)

$g_{\text{buffer}}^{\text{old}}$	Length-P stored gradient column before a backward contribution.
$g_{\text{buffer}}^{\text{new}}$	Stored gradient column after adding the current backward contribution.
θ	Length-P column of current parameter values.
$L, \nabla_\theta L$	Scalar loss from the current forward computation and its parameter gradient.
P	Total scalar parameter count.

$$g_{\text{buffer}}^{\text{new}} = g_{\text{buffer}}^{\text{old}} + \nabla_\theta L \tag{11.16}$$

One backward contribution is added to the stored gradient buffer; clearing that buffer is a separate operation.

In implementations, the flattened column is stored as separate gradient arrays with the corresponding parameter shapes.

▶ Example: Observe accumulation on the running network

At unchanged weights, one backward pass stores hidden-weight gradient approximately 6.475. Recompute a fresh forward loss and run backward again without clearing: the buffer becomes approximately 12.950. The function's derivative is unchanged; the buffer contains two contributions. Clear it before a third fresh pass and the value returns to approximately 6.475. Repeat for all four parameters. Holding weights fixed separates accumulation from optimization, while recomputing the forward pass avoids confusing buffer accumulation with reuse of a released computation graph.

Interpretation and limits

Automatic differentiation follows the program that was executed. It cannot infer a missing dependency, a correct target shape, or the intended gradient-clearing schedule.

Table 11.1. 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.
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 11.2. 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 autograd can check a small analytic derivation.	Use deterministic double precision and avoid nondifferentiable kink points.

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 analytic, numerical, and automatic 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 several unit parameter directions with central differences, then compare every gradient array with double-precision PyTorch. 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 11.3. 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

The selected network's validation MSE was 0.05, versus 0.069 for ridge. Three directional finite differences and a same-state autograd comparison agreed closely with the handwritten gradients. These checks support the implementation on the inspected batch, not every possible 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. PyTorch was used for the gradient check, not to replace the handwritten training loop.

▶ Example: inspecting the real data

A displayed actual MNIST image has 28 times 28, or 784, recorded pixel coordinates. The 784-to-12-to-1 network has 9,433 trainable scalars: 9,408 hidden weights, 12 hidden biases, 12 output weights, and one output bias. Verify that count from the real parameter arrays before running the image-batch derivative audit.

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.

Chapter Capability

Implement a small neural network, verify derivative calculations independently, and compare manual results with automatic differentiation.

Week 11: separate practice, solutions and assessment

References

[11.1] Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. Psychological Review 65(6), 386-408. Source / DOI

[11.2] Cybenko, G. (1989). Approximation by superpositions of a sigmoidal function. Mathematics of Control, Signals and Systems 2, 303-314. Source / DOI

[11.3] Rumelhart, D. E.; Hinton, G. E.; Williams, R. J. (1986). Learning representations by back-propagating errors. Nature 323, 533-536. Source / DOI

[11.4] Goodfellow, I.; Bengio, Y.; Courville, A. (2016). Deep Learning. MIT Press. Source / DOI

[11.5] 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

[11.6] Griewank, A.; Walther, A. (2008). Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation. 2nd ed., SIAM. Source / DOI

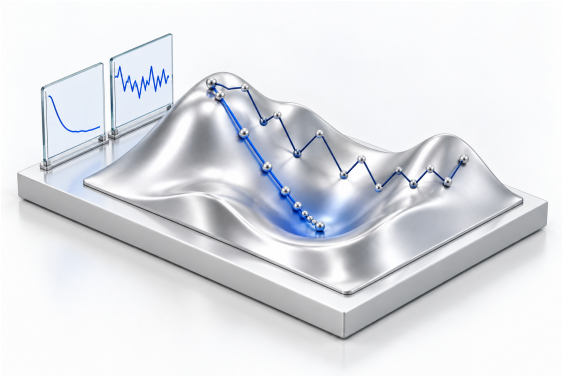
[11.7] Glorot, X.; Bordes, A.; Bengio, Y. (2011). Deep Sparse Rectifier Neural Networks. Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, PMLR 15, 315-323. Source / DOI

[11.8] 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

[11.9] 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

CHAPTER 12

Training And Debugging Neural Networks



A poor training curve does not identify its own cause. Establish a small reproducible experiment and use controlled checks to distinguish a fitting bug from an unsuitable model or training choice.

What You Will Be Able To Do

- Implement a shape-safe PyTorch mini-batch training and evaluation loop.
- Compare SGD and Adam with controlled initialization and validation data.
- Restore an early-stopped checkpoint and reproduce predictions after saving and loading.
- Diagnose optimization failures separately from generalization failures.

Before you begin: Forward passes, losses, backpropagation, and autograd; Training, validation, and test partitions; Basic Python classes and tensor operations.

Core route: 12.1, 12.2, 12.3, 12.4, 12.5. Revisit the proofs after reproducing the Examples.

Chapter notation

CPU	Central processing unit: general-purpose processor for the small training examples.

SGD	Stochastic gradient descent using mean mini-batch gradients for parameter updates.
Adam	Adaptive moment estimation: bias-corrected moving first and second gradient moments.
AdamW	Adam with decoupled weight decay: shrinkage separate from the adaptive gradient step.
ReLU	Rectified linear unit: maximum of zero and scalar preactivation.

A correct gradient is only one component of a successful learning system. Data must arrive in the right format, parameters need a usable starting scale, updates must be stable, and model selection must respect the boundary around the test set. A training loop is therefore both numerical software and an experimental protocol. Its small implementation details can change the question being answered.

This chapter builds a complete CPU-friendly PyTorch workflow around a synthetic classification problem. We compare optimizers, add regularization and early stopping, verify a saved model, and deliberately create a failure case. The aim is not to identify a universally best configuration. It is to make each training decision observable, reproducible within a declared environment, and defensible using data that are permitted to influence that decision.

12.1 A mini-batch loop has a precise order

Separate training, validation, and test examples before fitting preprocessing. Training examples supply gradient updates; validation examples guide architecture, optimizer settings, and stopping; test examples estimate the final chosen procedure once those choices are frozen. Fit scaling parameters on training inputs and reuse them unchanged elsewhere. For grouped or temporal data, replace a convenient random split with a partition that represents the intended generalization task. Neural networks do not make inappropriate splitting less consequential.

A mini-batch is a subset of training examples used for one update. In a standard PyTorch loop, enter training mode, clear accumulated gradients, calculate logits, compute a scalar loss, call backward, and then let the optimizer update parameters. With independent examples, shuffling each epoch helps avoid a fixed ordering of class or acquisition conditions. An epoch

is one traversal of the training data; its number of updates depends on batch size and whether an incomplete final batch is retained.

With integer class targets, cross-entropy expects one logit per class and valid integer target indices; continuous features normally use floating-point tensors. Accumulate each batch's mean loss times its example count, then divide by the total count. Label an aggregate collected during updates as an online training loss; use a separate fixed-weight evaluation for the final model's loss. Track task metrics too: changing predicted probabilities can improve cross-entropy without changing any predicted class.

Goodfellow, Bengio, and Courville [12.1] develop the optimization and practical-methodology foundations of neural-network training. The mini-batch loop must consistently connect forward evaluation, loss reduction, gradient computation, and parameter updates before larger-scale experiments are informative.

Notation for equations (12.1), (12.2)

t	Nonnegative update index; usually several updates per training-data traversal.
B_t	Nonempty mini-batch training-index set at update t.
$\lvert B_t \rvert$	Current batch size, including a potentially shorter final batch.
n, d, C, P	Positive training count, input width, class count and scalar-parameter count.
i	Training index from one through n, restricted to the current mini-batch sum.
x_i, y_i	Length-d input column and matching integer class label from zero through C-minus-one.
θ	Length-P flattened parameter column; batch derivatives evaluated at theta-t.
f_θ	Neural map from a length-d input column to a length-C logit column under current parameters.
ℓ	Scalar example loss against its target, e.g. logit cross-entropy.
$\nabla_\theta \ell$	Example-loss gradient over all parameters at their current values.
g_t	Mean current mini-batch gradient, matching the flattened parameter dimension.
θ_t, θ_{t+1}	Parameters before and after the current SGD update.

η Positive plain-SGD learning rate multiplying the mean mini-batch gradient.

$$g_t = \frac{1}{|B_t|} \sum_{i \in B_t} \nabla_\theta \ell(f_\theta(x_i), y_i) \tag{12.1}$$

$$\theta_{t+1} = \theta_t - \eta g_t \tag{12.2}$$

B_t is the mini-batch of example indices at update t and its size is $|B_t|$. The per-example loss is ell, f_θ produces logits, g_t is the mean batch gradient, and eta is the learning rate. This is the plain SGD update.

All per-example gradients in this update are evaluated at the current parameter vector. The data term is averaged, not summed.

▶ Example: The short final batch changes loss accounting

For batches of 64, 64, and 22 examples with mean losses 0.4, 0.5, and 0.9, accumulate loss totals 25.6, 32, and 19.8. Their sum, 77.4, divided by 150 examples is 0.516. The unweighted mean of batch means is 0.6, which overweights the short final batch. If optimizer updates occurred between batches, 0.516 summarizes losses encountered during training at different parameter states; it is not the final model's training loss. To measure that quantity, evaluate the complete training set again at fixed final weights.

Interpretation and limits

Do not select stopping time, learning rate, or architecture from test performance. A final test set loses its evaluation role as soon as it repeatedly guides those choices.

Algorithm 12.1. Mini-batch training and checkpoint selection

Notation and assumptions

$\mathcal{D}_{\text{tr}}, \mathcal{D}_{\text{val}}, \mathcal{D}_{\text{test}}$ Nonempty disjoint training, validation and untouched final-test data.

S, o, f_S, θ Complete current model state, optimizer state, logits function and trainable parameters inside S.

T, e, p, δ, ρ Positive epoch budget, current epoch, positive patience or infinity when disabled, nonnegative minimum improvement and stop reason.

$B, X_B, y_B, |B|$ Nonempty batch, its aligned inputs/targets, and example count.

ℓ_B, g, A, N	Batch mean loss, parameter gradients, accumulated weighted loss and accumulated example count.
v, b, r	Validation loss, accepted best loss and consecutive non-improvement count.
S^*, e^*, H	Independent selected model-state copy, selected epoch and training/validation history.

Init seeds before model, optimizer and batch-order initialization; sets best loss to infinity, checkpoint/epoch absent, counter zero, history empty and stop reason to budget. Batches uses the seeded order and must yield at least one nonempty batch per executed epoch. The history operator appends a row. Loss checks finite logits, aligned targets and integer class range. Step performs one clipped optimizer update. Copy independently saves all model state. Eval uses evaluation mode without gradient recording. Finish rejects absent checkpoints; otherwise restores the copy and evaluates final test once.

Algorithm 12.1

Input: Architecture, optimizer, declared batching/loss/clipping policy and seed.

Disjoint data and finite training budget. $\mathcal{D}_{\mathrm{tr}}, \mathcal{D}_{\mathrm{val}}, \mathcal{D}_{\mathrm{test}}, T$

Checkpoint rule and optional patience. $\delta \geq 0, \quad p \in \mathbb{N}_{>0} \cup \{\infty\}$

Output: Selected inference artifact, history, selected epoch, stop reason and test metrics; or an explicit failure without final-test evaluation.

Initialize $\mathrm{Init}(\mathrm{seed})$
For $e = 1, \ldots, T$
  Set Training mode.
  Reset $(A, N) \leftarrow (0, 0)$
  For $B \in \mathrm{Batches}(\mathcal{D}_{\mathrm{tr}}, e)$
    Clear Parameter gradient buffers.
    Forward Mean batch loss. $\ell_B \leftarrow \mathrm{Loss}(f_S(X_B), y_B)$
    Backward $g \leftarrow \nabla_\theta \ell_B$
    $(S, o) \leftarrow \mathrm{Step}(S, o, g)$
    Accumulate $(A, N) \leftarrow (A + |B|\ell_B, N + |B|)$
  End for

Algorithm 12.1 (continued)

If $N = 0$
    Fail Empty epoch. Stop the entire procedure; do not validate or test.
End if
Validate $v \leftarrow \mathrm{Eval}(S, \mathcal{D}_{\mathrm{val}})$
$H \leftarrow H \,\|\, [(e, A/N, v)]$
If $v < b - \delta$
    Save $(b, S^*, e^*, r) \leftarrow (v, \mathrm{Copy}(S), e, 0)$
Else $r \leftarrow r + 1$
End if
Break with patience status if $r \geq p$
End for
Return $\mathrm{Finish}(S^*, H, e^*, \rho, \mathcal{D}_{\mathrm{test}})$

Why this works. Copy is deep and includes model buffers. Evaluation mode does not itself disable gradient recording; Eval and Finish apply both controls.

An empty epoch stops before validation, history append or division by the example count, even if a previous checkpoint exists. Missing/nonfinite required gradients, nonfinite loss or invalid targets also stop without final-test evaluation. Record empty-epoch failure, numerical failure, patience or budget as the stop reason. Minimum improvement can reject a numerically lower loss. Save preprocessing, architecture and feature/class order with the inference artifact; exact training resumption additionally needs optimizer, random-generator and loader state.

12.2 Initialization and optimizers shape the trajectory

Initialize hidden units asymmetrically so they can learn different features. Setting all hidden weights identically can make units receive identical gradients and remain redundant. The initialization scale should also preserve useful activation and gradient magnitudes across layers. For ReLU networks, Kaiming-style initialization accounts for the fraction of activations suppressed by the nonlinearity. Xavier-style schemes are often associated with balanced variance in smoother activations. These are informed starting points, not substitutes for inspecting actual activations and gradients.

SGD uses the current batch gradient. Momentum retains contributions from previous gradients, while Adam also estimates coordinatewise second moments to rescale updates and corrects both moment estimates for their zero initialization. These are different update rules, not substitutes for choosing a learning rate. Compare optimizers with a declared tuning budget and the same validation protocol; copying one optimizer's preferred rate to another does not establish a fair comparison or a generalization advantage.

Control comparisons carefully. Use the same data partitions, preprocessing, architecture, initial weights when feasible, and a declared computation or epoch budget. Give each optimizer a reasonable validation-based search rather than comparing one well-tuned setting with another arbitrary default. Fixed seeds aid debugging but do not guarantee identical results across all hardware, kernels, or software versions. Record package versions, device, seed, and actual curves, and repeat promising comparisons when variation could change the conclusion.

Glorot and Bengio [12.2] analyze how initialization affects signal and gradient propagation. Kingma and Ba [12.3] introduce Adam’s adaptive moment estimates; these address different parts of training dynamics, so an adaptive optimizer does not remove the need for suitable initialization and learning-rate choices. He and colleagues [12.6] derive an initialization scale adapted to rectified activations, supplying the source for the fan-in rule used here. Sutskever and colleagues [12.8] show experimentally that initialization and momentum schedules can materially affect deep-network training. Optimizer comparisons should therefore control the starting state and tune the declared schedules instead of treating a method name as an explanation.

Read an optimizer curve within its assumptions

A deterministic gradient-descent guarantee does not automatically apply to Adam, momentum or mini-batch training. Supplement S3 distinguishes convex objective-gap guarantees from smooth nonconvex stationarity guarantees, with exact assumptions and proofs. Supplement S4 applies that distinction to this chapter: controlled initialization, validation-based tuning and measured variability remain necessary evidence for the executed neural-network comparison.

Mathematical Supplement: S4

Notation for equation (12.3)

W_{ij}	Random initialized scalar input-to-output connection weight.
i, j	Input index from one through fan-in and output-neuron index across layer width.
Var	Variance over independent initialization draws, before training.
d_{in}	Fan-in: incoming units/scalar inputs per output preactivation.

$$\mathrm{Var}(W_{ij}) \approx \frac{2}{d_{\mathrm{in}}} \tag{12.3}$$

This common ReLU-oriented initialization scale uses d_{in} incoming units per neuron. W_{ij} is an initialized weight. It is a variance guideline under simplifying independence assumptions, not a guarantee for every architecture.

This ReLU-oriented variance guideline uses simplifying independence assumptions. A library may store the weight matrix transposed without changing the guideline.

Notation for equations (12.4), (12.5), (12.6), (12.7), (12.8)

t	Positive update index for moment recurrences and bias corrections.
g_t	Current coordinatewise mini-batch gradient at current parameters.
m_t, m_{t-1}	Current/previous exponential first-gradient moments, initialized to zero.
v_t, v_{t-1}	Current/previous exponential second raw-gradient moments, initialized to zero.
β_1, β_2	First/second moment decay coefficients, each at least zero and below one.
g_t^2	Coordinatewise gradient square, not matrix product or squared norm.
$\hat{m}_t$	First moment divided by one minus beta-one to power t.
$\hat{v}_t$	Second moment divided by one minus beta-two to power t.

θ_t, θ_{t+1}	Parameter arrays before/after Adam, matching moment-array shapes.
η	Positive base learning rate; corrected moments rescale coordinates.
ϵ	Small positive stabilizer after coordinatewise square root, preventing zero denominators.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1) g_t \tag{12.4}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \tag{12.5}$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{12.6}$$

$$\hat{v}_t = \frac{v_t}{1-\beta_2^t} \tag{12.7}$$

$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{12.8}$$

Adam tracks first and second moment arrays m_t and v_t of gradient g_t using decay factors β_1 and β_2 . Squares and division are elementwise. Hats denote bias-corrected moments; epsilon is a small positive numerical stabilizer, and eta is the learning rate.

Squares, square roots, and divisions are coordinatewise. Initialize both moments to zero and begin the moment index at one. The explicit bias corrections use that positive update count. This is Adam without decoupled weight decay.

◇ Theorem 12.1: Curvature controls both stable step size and convergence speed

Notation for this result

P	Positive number of scalar parameters in this quadratic; P is a count, not a preconditioner matrix.
J, θ, θ_*	Quadratic objective, length-P parameter column, and its unique minimizer.
A	Real symmetric positive-definite P-by-P Hessian of the quadratic objective.
t, e_t	Nonnegative update index and the length-P parameter-error column before that update.

θ_t	Length-P parameter column at update t; its zero-indexed value is the chosen initial point.
I, η	P-by-P identity matrix and fixed positive learning rate.
Q, Λ	Orthogonal eigenvector matrix and diagonal matrix of positive Hessian eigenvalues.
$u_t, u_{t,j}, j$	Error in eigenvector coordinates, its j-th component, and index from one through P.
λ_j	Positive Hessian eigenvalue corresponding to coordinate j.
$\lambda_{\min}, \lambda_{\max}$	Smallest and largest positive Hessian eigenvalues, respectively.
$\rho(\eta)$	Worst eigen-direction contraction factor for one gradient-descent update.
η_*, ρ_*	Optimal fixed learning rate for worst-direction contraction and its resulting contraction factor.
κ	Spectral condition number of this positive-definite Hessian, at least one.

Assumptions. Consider a strictly convex quadratic objective with a real symmetric positive-definite Hessian. Gradient descent uses an exact full gradient and a fixed positive learning rate. The conclusion is global for this quadratic, not for an arbitrary neural-network loss.

Gradient descent converges from every starting point exactly when its step size is below twice the reciprocal of the largest Hessian eigenvalue. The optimal fixed step for the worst eigen-direction depends on both extreme eigenvalues, so poor conditioning slows the best possible fixed-step rate.

□ **Proof**

$$J(\theta) = \tfrac{1}{2}(\theta - \theta_*)^T A(\theta - \theta_*) \tag{12.T1}$$

$$e_{t+1} = (I - \eta A)\, e_t \tag{12.T2}$$

$$e_t = \theta_t - \theta_* \tag{12.T3}$$

The spectral theorem provides orthonormal Hessian eigenvectors. Expressing the error in that basis decouples the iteration into scalar recurrences. Each coordinate is multiplied repeatedly by one minus the learning rate times its eigenvalue.

$$A = Q\Lambda Q^T \tag{12.T4}$$

$$u_t = Q^T e_t \tag{12.T5}$$

$$u_{t,j} = (1 - \eta\lambda_j)^t u_{0,j} \tag{12.T6}$$

Convergence for every initial error requires every scalar multiplier to have absolute value below one. Solving these inequalities gives the interval below. At its upper endpoint the steepest component alternates without shrinking; above it that component diverges whenever initially present.

$$0 < \eta < 2/\lambda_{\max} \tag{12.T7}$$

$$\rho(\eta) = \max_j |1 - \eta\lambda_j| \tag{12.T8}$$

The maximum absolute affine function over the eigenvalue interval occurs at an endpoint. Minimizing that maximum balances the positive multiplier at the smallest eigenvalue against the negative multiplier at the largest. Solving their equality gives the optimal rate and its condition-number dependence.

$$\eta_* = \frac{2}{\lambda_{\min} + \lambda_{\max}} \tag{12.T9}$$

$$\rho_* = \frac{\kappa - 1}{\kappa + 1} \tag{12.T10}$$

$$\kappa = \frac{\lambda_{\max}}{\lambda_{\min}} \tag{12.T11}$$

□

Feature scaling and preconditioning can improve curvature geometry. For a nonconvex network this quadratic calculation describes local intuition only; curvature and gradient noise change along training.

▶ Example: Initialization scale depends on fan-in

For a ReLU layer with fan-in 50, the variance prescription gives 2 divided by 50, or 0.04. A normal initializer needs its square root: standard deviation 0.2.

With fan-in 100, variance becomes 0.02 and standard deviation approximately 0.141, not 0.02. The scaling compensates for summing more inputs under the stated initialization assumptions. Correlated inputs, biases, nonlinearities, and distribution shifts can still change observed behavior. Inspect activation and gradient magnitudes on a real batch; the variance calculation is a design approximation, not a guarantee for every layer.

Interpretation and limits

An optimizer name does not imply a fair comparison. Control initialization and budgets, tune settings on validation data, and report measured variability instead of assuming Adam or SGD must win.

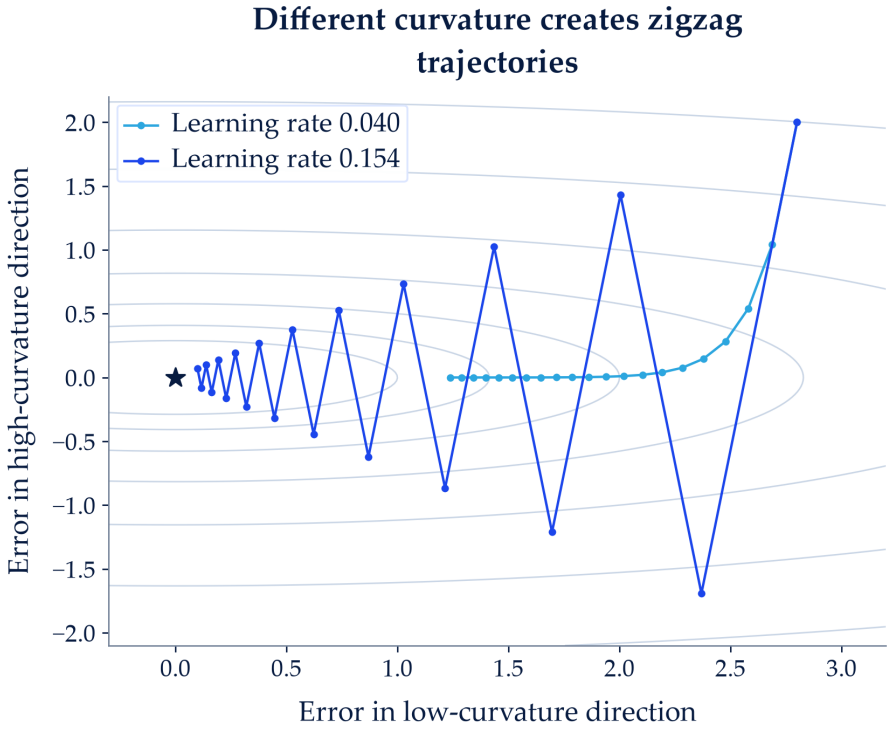


Figure 12.1. Exact gradient descent on a quadratic with Hessian eigenvalues 1 and 12, starting at error (2.8, 2.0). Trajectories compare two stable steps; unequal curvature creates different contraction rates and zigzags. The star is the optimum.

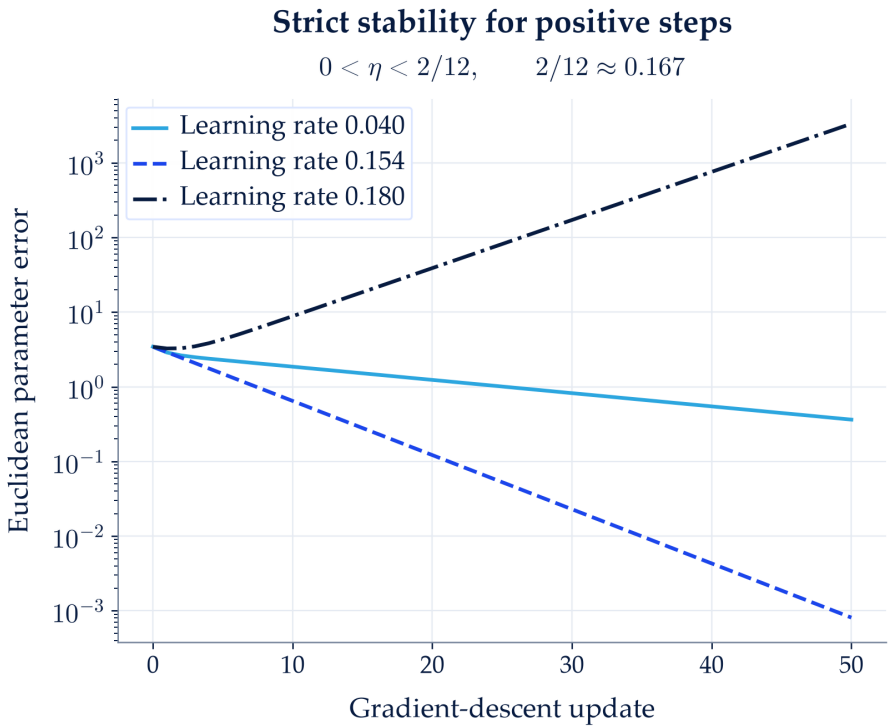


Figure 12.2. Analytical recurrence norms for the same quadratic. The optimal worst-direction fixed step is 2/13. A step of 0.18 exceeds the strict stability limit 2/12 and diverges. These are exact quadratic dynamics, not neural-training benchmark curves.

12.3 Regularization and early stopping constrain fitting

Regularization changes how strongly a model can adapt to the training sample. A squared-weight penalty adds a preference for smaller parameters to the mean training loss. Weight decay instead describes a parameter-shrinkage operation; it coincides with a familiar penalty update for plain SGD under appropriate conventions, but coupling a penalty into Adam's adaptive gradient is not the same as decoupled AdamW weight decay. State the optimizer and convention rather than treating every weight_decay argument as interchangeable mathematics.

Dropout randomly suppresses selected activations during training and rescales the retained values. This encourages the network to function under many perturbed internal representations, but its benefit is task-dependent. Standard dropout is disabled in evaluation mode, which is essential for stable validation predictions. Other regularizers include data augmentation that preserves labels, capacity constraints, and domain-appropriate architectural choices. An augmentation that changes the correct label is not harmless regularization; it creates inconsistent supervision.

Early stopping selects a checkpoint based on validation performance. Track a chosen validation metric, save a copy of the best model state, and stop after a declared patience interval without sufficient improvement. Restore the saved state before final evaluation; the final training epoch is not necessarily the best checkpoint. Validation is now part of model selection, so repeated experimentation can overfit it. Reserve untouched test data and record the stopping rule along with all other hyperparameters.

Srivastava and colleagues [12.4] introduce dropout as randomized unit omission during training. Alongside the regularization discussion of Goodfellow and colleagues [12.1], this motivates evaluating regularization and early stopping by their effect on held-out performance rather than by training loss alone. Loshchilov and Hutter [12.7] distinguish decoupled weight decay from adding a squared-weight penalty to an adaptive-gradient objective. Prechelt [12.9] empirically compares validation-based stopping criteria and their tradeoff between training time and generalization. A patience rule is a declared model-selection choice, not a universal number of epochs or a reason to consult the test set.

Notation for equation (12.9)

$J(\theta)$	Scalar mean prediction loss plus squared-weight penalty.
θ	All trainable parameters, including any excluded from penalization.
n, i	Training count and index from one through n.
x_i, y_i	Training-example-i input vector and target.
f_θ	Theta-parameterized predictor evaluated by the example loss.
ℓ	Scalar example loss, averaged before adding the penalty.
W	Flattened selected weights; biases or other parameters may be excluded.
λ	Nonnegative squared-weight penalty strength relative to mean data loss.
$\| W\|_2^2$	Sum of squared selected-weight entries, not a matrix spectral-norm penalty.

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell(f_\theta(x_i), y_i) + \frac{\lambda}{2}\| W\|_2^2 \tag{12.9}$$

J is the regularized training objective, ell is per-example loss, and W collects weights chosen for penalization, potentially excluding biases. Lambda is a nonnegative penalty strength. The data term uses a mean, so its scale does not grow with n.

Flattening matrix weights makes this equivalent to summing their squared Frobenius norms. Coupled Adam penalties are not AdamW weight decay.

Notation for equations (12.10), (12.11)

h_j	Current example's hidden coordinate j before dropout.
$\tilde{h}_j$	Training activation after random keep mask and inverted-dropout scaling.
j, d	Hidden index from one through d and positive dropout-representation width.
p	Drop probability from zero inclusive to one exclusive; keep probability one minus p.
r_j	Independent binary keep indicator: one retains the activation.
$\text{Bernoulli}(1-p)$	Probability one minus p at one, and p at zero.

$$\tilde{h}_j = \frac{r_j h_j}{1-p} \tag{12.10}$$

$$r_j \sim \text{Bernoulli}(1-p) \tag{12.11}$$

In inverted dropout, h_j is an activation, p is the drop probability with 0 less than or equal to p less than 1, and r_j is an independent keep indicator. During ordinary evaluation, the unmodified h_j is used.

This is inverted dropout during training. Standard evaluation uses the original activations without random masking or additional scaling.

◇ Proposition 12.2: Inverted dropout preserves a mean, not an entire nonlinear prediction

Notation for this result

$h_j, \widetilde{h}_j$	Fixed hidden activation and its randomly masked, rescaled training-time value.
j, d	Coordinate index from one through d and hidden representation dimension.
r_j, p	Independent binary keep indicator for coordinate j and the probability of dropping a coordinate.
$\mathbb{E}, \mathrm{Var}$	Expectation and variance over independently sampled dropout masks, conditional on the fixed hidden activations.
S, w, b	Random scalar readout, fixed length-d readout weight column, and fixed scalar bias.
$h, \widetilde{h}$	Length-d columns of original and masked hidden activations, respectively.
y, w_j	Fixed scalar target and coordinate j of the readout weights.

Assumptions. Condition on a fixed hidden activation vector. Independent Bernoulli masks retain each coordinate with the same positive keep probability. The exact loss identity below assumes a linear readout and squared error.

Inverted dropout preserves each activation's expectation while increasing its variance. For a linear readout it adds an exact data-dependent quadratic penalty to expected squared error. Equality of activation means does not extend to arbitrary nonlinear downstream operations.

□ **Proof**

$$\tilde{h}_j = \frac{r_j h_j}{1-p} \tag{12.T12}$$

$$r_j \sim \text{Bernoulli}(1-p) \tag{12.T13}$$

$$0 \leq p < 1 \tag{12.T14}$$

A Bernoulli variable has mean equal to its success probability and variance equal to success probability times failure probability. Scaling these two moments by the fixed activation gives the mean and variance below. Independence makes distinct masked coordinates uncorrelated conditional on the hidden vector.

$$\mathbb{E}[\tilde{h}_j] = h_j \tag{12.T15}$$

$$\text{Var}(\tilde{h}_j) = \frac{p}{1-p} h_j^2 \tag{12.T16}$$

$$S = w^T \tilde{h} + b \tag{12.T17}$$

$$\mathbb{E}[S] = w^T h + b \tag{12.T18}$$

$$\mathbb{E}[(S-y)^2] = (w^T h + b - y)^2 + \frac{p}{1-p} \sum_{j=1}^{d} w_j^2 h_j^2 \tag{12.T19}$$

To prove the loss identity, center the random readout around its expectation and expand the square. The cross term has zero expectation. Its variance is a sum of weight-squared times activation variances because the masks are independent. Substitution produces the displayed penalty exactly. Averaging over examples weights each squared parameter by the corresponding empirical mean squared activation, so the result is generally not an isotropic weight penalty.

The mean-preservation claim stops before nonlinear downstream computation. For example, applying a square to a nonconstant random activation adds its positive variance to the square of its mean. Evaluation with dropout disabled therefore does not usually equal an exact average over every masked nonlinear network. It is a deterministic convention supported by mean preservation at the masked layer, not a universal identity for the full predictor.

□

Dropout strength changes both noise and an implicit penalty in this restricted model. Whether it improves validation performance in a deep network remains an empirical question.

◇ Derivation 12.3: Gaussian-prior scaling and the limits of the weight-decay equivalence

Notation for this result

$W, n, \mathcal{D}$	Flattened penalized-weight column, positive training sample count, and complete observed training dataset.
i	Training-example index from one through n, including in the abbreviated sum later in this derivation.
$\ell_i(W), \ell_i$	Negative log likelihood for example i at the current parameters; the shorter notation suppresses the weight argument.
τ^2, C	Positive common prior variance and additive term independent of the optimized weights; C is not a class count here.
$p(W \mid \mathcal{D})$	Posterior density of the weights under the stated likelihood and independent Gaussian prior.
MAP	Maximum a posteriori estimation: choose weights that maximize their posterior density under the declared likelihood and prior.
J, λ	Mean-loss maximum-a-posteriori objective and its squared-weight penalty coefficient under this likelihood scaling.
W_+, η, g	Updated weights, positive learning rate, and gradient of the mean data loss before the update.
P	Fixed positive diagonal matrix with one row and column per penalized weight; P is not a parameter count here.
$W_{\text{coupled}}, W_{\text{decay}}$	Weight columns after a preconditioned penalty-gradient step and after a separately applied isotropic decay step.

Assumptions. Training observations are conditionally independent. Each data loss is an exact negative log likelihood. Penalized weights have independent zero-mean Gaussian priors with a common positive variance. Other parameters are unpenalized for this comparison. Treat any unpenalized parameters as fixed in the displayed posterior over weights, or regard the displayed objective as a penalized-likelihood comparison; no proper prior for those omitted parameters is asserted.

MAP estimation gives a squared-weight penalty whose coefficient depends on whether the data loss is summed or averaged. Coupled penalties coincide with multiplicative weight decay for plain gradient descent, but not generally for coordinate-adaptive updates.

□ Derivation

$$-\log p(W \mid \mathcal{D}) = \sum_{i=1}^{n} \ell_i(W) + \frac{\|W\|_2^2}{2\tau^2} + C \tag{12.T20}$$

Bayes' rule multiplies the likelihood and prior. Conditional independence turns the likelihood into a product, so its negative logarithm is the sum of per-example losses. The Gaussian prior contributes the squared norm divided by twice its variance; posterior normalization contributes only the additive constant. Dividing the whole objective by sample count leaves the minimizer unchanged and determines the mean-loss penalty coefficient.

$$J = \frac{1}{n} \sum_{i} \ell_i + \frac{\lambda}{2} \|W\|_2^2 \tag{12.T21}$$

$$\lambda = \frac{1}{n\tau^2} \tag{12.T22}$$

$$W_+ = W - \eta(g + \lambda W) = (1 - \eta\lambda) W - \eta g \tag{12.T23}$$

Differentiating the penalty yields a multiple of the current weight vector. Substituting it into plain gradient descent proves the exact shrinkage equivalence. Additional scaling constants in a loss, such as observation variance in Gaussian regression, change the corresponding prior interpretation and must be retained.

$$W_{\text{coupled}} = W - \eta P(g + \lambda W) \tag{12.T24}$$

$$W_{\text{decay}} = (1 - \eta\lambda)\, W - \eta P g \tag{12.T25}$$

The coupled step shrinks each coordinate according to its preconditioner entry, while decoupled decay uses one common factor. They agree only under special conditions, such as an identity preconditioner on all affected coordinates. In Adam the discrepancy can be larger because adding a penalty also changes moment estimates and future preconditioners. Thus an AdamW decay coefficient cannot in general be identified with the same simple Gaussian-prior MAP coefficient.

□

A Bayesian derivation clarifies a precisely scaled objective; it does not guarantee improved generalization, and it does not make every optimizer's weight-decay option mathematically interchangeable.

▶ Example: Restore the selected checkpoint, not the last one

For validation losses 0.42, 0.35, 0.33, 0.34, 0.36, and 0.35, require a strictly lower loss and set patience to three. The non-improvement counter after each epoch is 0, 0, 0, 1, 2, 3. Save independent checkpoint copies after epochs one, two, and three; stop after epoch six and restore epoch three. Only then evaluate the test set. Testing epoch six would evaluate the wrong selected model. A reference to mutable weights is not a checkpoint: clone tensors or deep-copy the state before subsequent updates can overwrite it.

Interpretation and limits

Early stopping is model selection, not a reason to inspect the test set each epoch. Always restore an independent copy of the best validation checkpoint before final evaluation.

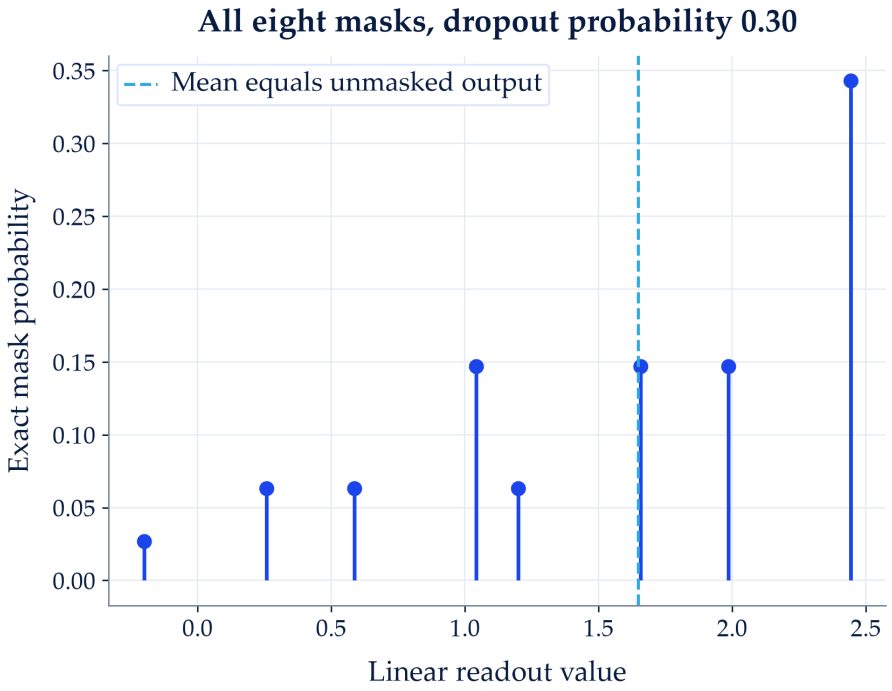


Figure 12.3. All eight independent dropout masks are enumerated for three fixed activations at drop probability 0.30. Inverted scaling preserves the unmasked linear-readout mean. The stem heights are exact mask probabilities rather than a sampling histogram.



Figure 12.4. Expected squared loss from independent mask enumeration agrees with the derived additional dropout penalty to numerical precision. Activations are (1.4, -0.8, 0.5), weights (0.7, -0.4, 1.1), bias -0.2 and target 1.5. The equality assumes a fixed representation and linear readout.

12.4 Training mode, inference mode, and checkpoints are distinct

Calling model.train() or model.eval() changes the behavior of modules that distinguish training from evaluation. Dropout samples masks during training; batch normalization uses different statistics depending on its configuration and mode. Neither call turns automatic differentiation on or off. A no-gradient or inference context controls whether derivative information is recorded. Typical validation therefore combines evaluation mode with disabled gradient recording, then restores training mode when optimization resumes.

A deployable predictor includes more than a parameter file. Preserve architecture settings, learned preprocessing, feature order, class mapping, and any selected threshold together with the state dictionary and version metadata. Loading weights into the wrong architecture should fail loudly rather than silently ignore missing keys. Test a round trip: predictions before saving and after loading should agree on a fixed batch under the same inference settings. The laboratory uses an in-memory buffer so this test needs no external files.

Resuming training requires additional state: optimizer moments, learning-rate scheduler state, epoch or update count, and, when exact continuation matters, random-number generator and data-order states. A weights-only inference checkpoint is not a complete resume checkpoint. Loading serialized objects also raises trust concerns; use trusted artifacts and the most restrictive loading options compatible with the saved format. Store enough metadata to reconstruct a model deliberately, rather than relying on whatever class definitions happen to be available later.

Ioffe and Szegedy [12.5] introduce batch normalization, whose training statistics and inference behavior must be handled consistently. Dropout [12.4] also changes behavior between training and evaluation, so saving only parameter arrays is insufficient when the inference computation depends on additional model state.

Notation for equation (12.12)

X_{check}	Fixed row-example verification batch with saved preprocessing applied.
$\hat{y}_{\text{saved}}$	Original saved-state prediction function in deterministic evaluation mode.

$\hat{y}_{\text{loaded}}$	Reloaded-checkpoint predictor with matching architecture, preprocessing and evaluation settings.
$\approx$	Declared absolute/relative tolerance agreement and identical output shapes.

$$\hat{y}_{\text{loaded}}(X_{\text{check}}) \approx \hat{y}_{\text{saved}}(X_{\text{check}}) \tag{12.12}$$

X_{check} is a fixed verification batch transformed by the saved preprocessing. The two predictions come from the loaded and original checkpoints in evaluation mode. Agreement within a declared numerical tolerance is a round-trip integrity check, not a generalization test.

Compare raw regression outputs, logits, or probabilities consistently. Agreement checks serialization integrity, not predictive generalization.

▶ Example: Weights alone do not reconstruct the predictor

A classifier expects a feature standardized with training mean 10 and scale 2. Raw input 14 must become 2 before entering the network. Loading correct weights but skipping this transformation instead supplies 14, so a valid parameter file can still produce invalid deployment predictions. Save the training mean, scale, feature order, and architecture with the weights. For a round-trip test, record predictions for a fixed raw batch, reload the full preprocessing-plus-model pipeline, and compare its predictions on that same raw batch. Comparing only already-transformed inputs would miss a broken preprocessing step.

Interpretation and limits

model.eval() does not disable gradient recording, and torch.no_grad() does not switch off dropout. Use both controls appropriately and preserve preprocessing with the model.

Optional interactive companion | Frozen blocks and adaptation

Transfer learning reuses a representation learned on an earlier task. Freezing a block prevents its parameter updates; unfreezing makes them eligible for optimization. Compare the trainable parameter count under different choices and distinguish that choice from evaluation mode. Treat any displayed expected accuracy or forgetting risk as illustrative unless a measured evaluation protocol is supplied. Chapter 13 applies this distinction to frozen text encoders.

Open Transfer Learning Lab

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

12.5 Diagnose fitting failures before expanding the model

Notation for this section

w	Only in the final scalar example: real trainable parameter initialized to one.
L	Only in that scalar example: squared-w loss, not the proposition's network depth.

Debug in order: check finite inputs, valid labels, expected shapes, and the initial loss scale; then verify that forward-pass parameters are registered with the optimizer. On one batch, inspect finite gradients, record parameters before and after a step, and confirm an actual update. A missing gradient suggests a detached graph; a finite but ineffective update points instead toward saturation, step size, or parameter mismatch. Repeat this small test before changing the architecture.

Try to overfit a tiny, fixed subset with augmentation and strong regularization disabled. A model with sufficient capacity should usually reduce its loss substantially on that subset; failure points toward data, objective, connectivity, or optimization problems. Success is a diagnostic of fitting ability, not evidence of good generalization. Examine training and validation curves together: high errors on both suggest underfitting or optimization trouble, while low training error and worse validation error suggest overfitting, shift, or an evaluation mismatch.

Investigate nonfinite or exploding losses by reducing the learning rate, checking feature scales and target ranges, and inspecting layerwise activations and gradient norms. Gradient clipping can bound an update's input signal but should not conceal a malformed loss or data pipeline. Keep failure experiments controlled and reversible. Change one factor at a time, preserve the original baseline, and record observations. Finally, compare against a simple non-neural baseline so network complexity must earn its operational cost.

Goodfellow, Bengio, and Courville [12.1] organize practical methodology around controlled experiments, implementation checks, and optimization diagnostics. These support checking loss, gradients, data boundaries, and saved prediction state before increasing model complexity. Pascanu, Mikolov, and Bengio [12.10] analyze vanishing and exploding gradients in recurrent networks and study gradient-norm clipping. The clipping operation has a specific stability motivation; it does not repair a wrong objective or establish a generalization guarantee for a feedforward model.

Notation for equations (12.13), (12.14)

g	Concatenated gradients of every parameter included in global-norm clipping.
g_j	Scalar coordinate j of the concatenated gradient.
j	Index over all included scalar coordinates.
$\|g\|_2$	Full concatenated gradient's Euclidean norm before clipping.
c	Positive maximum permitted clipped-gradient norm.
$\tilde{g}$	Uniformly rescaled gradient above threshold; unchanged otherwise.
$\min$	Minimum choosing a factor at most one, never amplifying gradients.

$$\|g\|_2 = \sqrt{\sum_j g_j^2} \tag{12.13}$$

$$\tilde{g} = g \min\left(1, \frac{c}{\|g\|_2}\right) \tag{12.14}$$

The vector g concatenates parameter gradients, and c is a positive clipping threshold. The clipped vector $\tilde{g}$ preserves direction while limiting its norm; if the norm is zero, leave the gradient unchanged. Clipping is a diagnostic and optimization tool, not a repair for invalid data.

When the gradient norm is zero, leave the vector unchanged rather than dividing by zero. Clipping does not repair nonfinite gradients.

◇ Proposition 12.4: A precise bound on backward gradient amplification

Notation for this result

l, L	Layer index from one through L and finite network depth; L is not the scalar loss in this proposition.
d_l	Positive width at level l, for levels zero through L; level zero is the input width.
a_0	Length-d-zero input column, before any affine layer.
a_l, b_l	Layer-l activation and bias columns, each of length d-l.
W_l	Weight matrix with d-(l-minus-one) input rows and d-l output columns.
ϕ_l, D_l	Coordinatewise activation and d-l-by-d-l diagonal matrix of its derivatives at the current preactivations.
J_l	d-l-by-d-(l-minus-one) Jacobian mapping a preceding-layer perturbation to the layer-l activation perturbation.
$\mathcal{L}, g_l$	Scalar loss of the network output and its column gradient with respect to layer-l activations.
g_0, g_L	Loss gradients with respect to the network input and final activation, respectively.
$\|\cdot\|_2$	Euclidean norm for vectors and induced spectral norm for matrices in this proposition.

Assumptions. A feedforward composition is differentiable at the evaluated input. Each layer has an affine preactivation followed by a coordinatewise activation. Use the spectral matrix norm and Euclidean vector norm; ReLU inputs exactly at a kink are excluded from the classical derivative claim.

The gradient reaching an early layer is bounded by the output gradient multiplied by the product of layer-Jacobian norms. Small factors can force vanishing gradients; factors above one only permit, rather than guarantee, amplification.

Complete supporting proof: Mathematical Supplement, A11 (in this book).

Monitor layerwise activations and gradients rather than assuming that an optimizer name prevents vanishing or exploding signals. The bound is local to the executed forward pass.

▶ Example: A correct gradient can still diverge

For loss equal to w squared, the gradient is 2w, so each gradient-descent step multiplies w by one minus twice the learning rate. Starting at 1 with rate 10, the multiplier is negative 19: the next weights are negative 19 and 361, with losses 361 and 130321. With rate 0.1, the multiplier is 0.8 and the first loss is 0.64. Convergence requires the multiplier's magnitude below one, hence a rate strictly between zero and one. The derivative is correct in both runs; the unstable setting diagnoses step size, not broken differentiation.

Interpretation and limits

Overfitting a tiny subset verifies only that the pipeline can fit those examples. It is not a validation result, and it must not be used to claim deployment performance.

Notation for Figure 12.5

MLP	Multilayer perceptron: a feedforward network with learned hidden layers, used here for synthetic scalar regression.
MSE	Mean squared error, the average squared prediction-minus-target residual on the indicated training or validation sample.

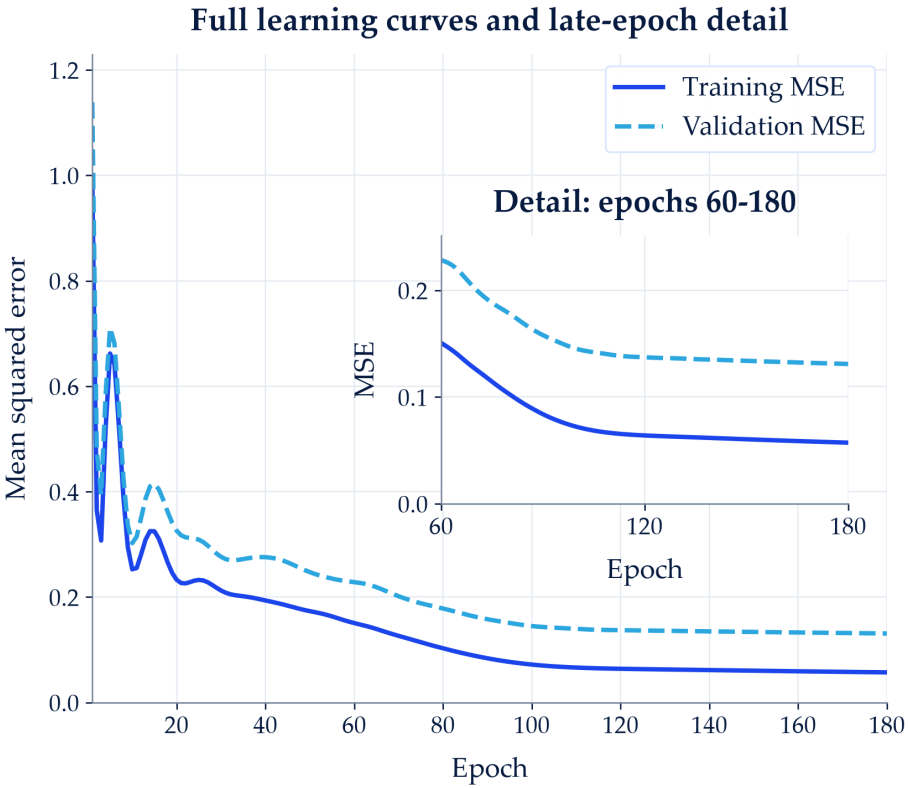


Figure 12.5. Actual learning curves for a small MLP on synthetic noisy regression. The main panel shows all epochs and loss values; the inset magnifies epochs 60-180. Validation is separate from training, and its curve may fluctuate. If this curve selects the checkpoint, it is tuning evidence; a further untouched test set is still required for final reporting.

Table 12.1. Training mechanisms are not interchangeable

Mechanism	What it controls	Important distinction
Mini-batch SGD	Updates from a sampled gradient estimate.	Batch size changes gradient variance and update frequency.
Momentum / Adam	Accumulate gradient information across updates.	Adaptive preconditioning changes the relation between an L2 penalty and weight decay.
Dropout / weight decay	Constrain different aspects of fitting.	Training-time stochastic behavior must be distinguished from inference.
Early stopping	Select a checkpoint using development evidence.	The selection set is not an untouched final test.

Table 12.2. Chapter synthesis

Concept	Main conclusion	Engineering check
Training loop	Mode, gradient clearing, forward, backward and update have distinct roles.	Deliberately test accumulation and a tiny-batch overfit case.
Initialization	Activation scale and gradient flow affect optimization.	Inspect activations and gradient norms before extending training.
Checkpoint	A reproducible restart needs more than weight arrays.	Save preprocessing, optimizer state, configuration and software provenance.
Diagnosis	A failed fit should be localized before model capacity is increased.	Check labels, loss, gradients, modes and data boundaries separately.

Case Study: Select and reload a Fashion-MNIST training run honestly

Problem and data

Fashion-MNIST provides a real ten-class image task for studying training discipline rather than maximizing a benchmark score. Can two optimizer configurations be compared with matched starting parameters and a clear validation rule, and can the selected predictor be reconstructed without changing its outputs? Fixed official-partition subsets keep the CPU budget modest. The key evidence is a reproducible loop, correctly copied checkpoint, and verified inference contract, not an assumed advantage of one optimization method.

Experimental design

Flatten fixed-range pixel values and train the same dropout network with SGD momentum and Adam. Match initialization and deterministic mini-batch order, use a ten-epoch ceiling and three-epoch patience, and save independent copies of improving states. Select the configuration with the lower restored validation cross-entropy. Serialize architecture, weights, pixel conversion, class order, and selected epoch to a memory buffer. Check loaded predictions on validation images before one final original-test evaluation against a majority-class reference.

Measured Reference Run

The run used 4,800 fitting and 1,200 validation images from original training, plus 2,000 original-test images. Two configurations shared a 784-to-64-to-10 network, dropout 0.2, initial weights and prescribed data order. SGD momentum and Adam each had a ten-epoch maximum with patience three. Restored validation cross-entropy selected Adam at epoch ten. Final evaluation followed an inference-artifact reload check.

Table 12.3. Measured case-study evidence

Final procedure	Accuracy	Macro F1	Test count
Selected Adam run	0.836	0.836	2000
Training majority	0.1	0.018	2000

Restored validation cross-entropy was 0.463 for Adam and 0.488 for SGD momentum. The reloaded model reproduced checked validation logits with maximum discrepancy 0.0; its final cross-entropy was 0.492. The artifact therefore preserved this inference computation rather than improving the model.

Failure Analysis And Limits

Aggregate accuracy concealed class-level weaknesses: shirt recall was 0.615 and coat recall 0.660, versus 0.960 for sneakers, with 200 test images per class. The saved inference artifact omitted optimizer and random/loader states, so it did not support exact training resumption. Two bounded configurations on a subset do not establish optimizer-family superiority or retail-image robustness.

Historical CPU saved run. Preserve pixel conversion, class order, architecture and selected weights. Matching logits on the checked batch is an inference round-trip test, not comprehensive deployment certification.

▶ Example: inspecting the real data

The gallery shows actual fitting-set garments with their recorded category names. Each 28-by-28 image becomes 784 normalized inputs. For a real final mini-batch smaller than 128, multiply its observed mean cross-entropy by its actual size before adding it to the epoch total; divide by 4,800 examples, not by the number of batches.

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.

Chapter Capability

Run a correct mini-batch loop, choose a checkpoint with validation evidence, and reproduce its predictions after reloading it.

Week 12: separate practice, solutions and assessment

References

[12.1] Goodfellow, I.; Bengio, Y.; Courville, A. (2016). Deep Learning. MIT Press. Source / DOI

[12.2] Glorot, X.; Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. Proceedings of Machine Learning Research 9, 249-256. Source / DOI

[12.3] Kingma, D. P.; Ba, J. (2015). Adam: A Method for Stochastic Optimization. International Conference on Learning Representations. Source / DOI

[12.4] Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. (2014). Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research 15, 1929-1958. Source / DOI

[12.5] Ioffe, S.; Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Proceedings of Machine Learning Research 37, 448-456. Source / DOI

[12.6] He, K.; Zhang, X.; Ren, S.; Sun, J. (2015). Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification. Proceedings of the IEEE International Conference on Computer Vision. Source / DOI

[12.7] Loshchilov, I.; Hutter, F. (2019). Decoupled Weight Decay Regularization. International Conference on Learning Representations. Source / DOI

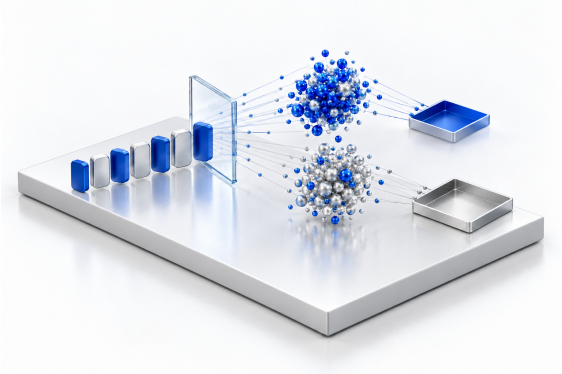
[12.8] Sutskever, I.; Martens, J.; Dahl, G.; Hinton, G. (2013). On the importance of initialization and momentum in deep learning. Proceedings of the 30th International Conference on Machine Learning, PMLR 28(3), 1139-1147. Source / DOI

[12.9] Prechelt, L. (1998). Early Stopping - But When?. Neural Networks: Tricks of the Trade, Lecture Notes in Computer Science 1524, 55-69, Springer. Source / DOI

[12.10] Pascanu, R.; Mikolov, T.; Bengio, Y. (2013). On the difficulty of training recurrent neural networks. Proceedings of the 30th International Conference on Machine Learning, PMLR 28(3), 1310-1318. Source / DOI

CHAPTER 13

Text Classification And Embeddings



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.

- Explain what frozen pretrained embeddings add and how to evaluate them without inventing semantic vectors.

- 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; Neural-network representations and inference.

Core route: 13.1, 13.2, 13.3, 13.5. Deepening route: 13.4.

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 13.2.
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 turns variable-length language into a fixed prediction task: route a request, categorize a document, or detect a specified property of a message. The representation is central. A classifier cannot use distinctions that its input representation discards, and a large representation does not automatically preserve the distinctions that matter. We begin with transparent lexical features before considering representations learned from external text.

The core laboratory uses a small literal corpus bundled directly in the notebook, so every required exercise works without downloads, accounts, or API keys. This corpus is deliberately synthetic and supports software and reasoning checks, not broad claims about production language understanding. A separate opt-in extension uses a real pretrained sentence encoder, documents its external data advantage, and compares validation performance under the same partition rules.

13.1 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.

A bag-of-words representation creates one feature for each vocabulary term and records how often it appears in a document. With n documents and d vocabulary items, the resulting design matrix X has shape n by d. Most entries are zero because an individual document contains only a small fraction of the vocabulary, so sparse storage is natural. Tokenization, case normalization, punctuation treatment, and vocabulary filtering define what counts as the same term and therefore change the model's input.

In ordinary inductive evaluation, fit the vocabulary on training documents, then transform validation and test documents with that fixed mapping. In cross-validation, fit it again inside each training fold. Later-only terms are ignored, so a document containing no recognized terms can become an all-zero row; test that case explicitly. Learning the vocabulary from held-out inputs leaks their feature information even without labels. A transductive protocol may permit such access, but must declare and evaluate that different information boundary.

Unigrams discard word order. Word bigrams preserve local pairs and can distinguish phrases such as not approved from approved, while fixed-length character sequences can capture spelling variants and fragments. These choices trade expressiveness, vocabulary size, and robustness. Removing stop words or stemming tokens is not automatically beneficial: negation, names, or punctuation can carry the target signal. Inspect actual tokenized examples and evaluate changes on validation data rather than treating preprocessing conventions as universally correct.

Manning, Raghavan, and Schütze [13.1] 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 (13.1), (13.2)

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.

$$X_{ij} = \mathrm{count}(v_j, \mathcal{D}_i) \tag{13.1}$$

$$X \in \mathbb{R}^{n \times d} \tag{13.2}$$

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.

13.2 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.

Term frequency measures how often a term occurs within a document. Inverse document frequency reduces the relative influence of terms that appear in many training documents. Their product is TF-IDF. Several formulas are used in practice, so state the convention. The laboratory uses raw term counts, a smoothed inverse-document-frequency expression, and rowwise L2 normalization. These choices match a common scikit-learn configuration and make a manual numerical check possible.

Document frequency counts documents containing a term, not the total number of times that term appears. A token repeated twenty times in one document contributes one to its document frequency. Smoothing avoids extreme behavior for rare terms and keeps the expression well-defined. Row normalization then puts nonzero document vectors on a comparable length scale. It does not make every document equally informative, and an empty or entirely unknown document remains the zero vector rather than becoming 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 [13.2] analyze term weighting for text retrieval, and Manning and colleagues [13.1] 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 (13.3), (13.4)

n	IDF-fitting training-document count, not prediction-batch size.
i, j	Transformed-batch document index 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.

$$\mathrm{idf}_j = \log\left(\frac{1+n}{1+\mathrm{df}_j}\right) + 1 \tag{13.3}$$

$$\widetilde{X}_{ij} = \mathrm{tf}_{ij}\,\mathrm{idf}_j \tag{13.4}$$

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 (13.5)

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.

$$X_{i,:} = \frac{\widetilde{X}_{i,:}}{\|\widetilde{X}_{i,:}\|_2} \tag{13.5}$$

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 13.1: 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

$$D = \mathrm{diag}(\mathrm{idf}_1, \dots, \mathrm{idf}_d) \tag{13.T1}$$

$$z(x) = \frac{Dx}{\|Dx\|_2} \tag{13.T2}$$

Diagonal weighting is linear. Scaling all counts by a positive number scales both the weighted vector and its norm by that number, so the factors cancel. The result is exactly invariant to repeating every token count proportionally, although real text editing may alter tokenization or n-gram features and need not satisfy this condition.

$$z(cx) = z(x) \quad (c > 0) \tag{13.T3}$$

$$\|u - v\|_2^2 = 2 - 2u^T v \tag{13.T4}$$

Expanding the squared distance gives each vector's squared norm minus twice their inner product. Both norms equal one, giving the displayed identity. Their inner product is also their cosine similarity, so nearest-neighbor rankings by distance and cosine are identical under these normalization conditions. Because lexical weights and counts are nonnegative, the cosine lies between zero and one; arbitrary dense embeddings can instead have negative cosine.

$$R(\mathcal{D}_a) = R(\mathcal{D}_b) \implies h(R(\mathcal{D}_a)) = h(R(\mathcal{D}_b)) \tag{13.T5}$$

The last implication is substitution of equal arguments into the same function. Unigram token counts are invariant to word-order permutations, and fixed weighting and normalization cannot recover information already discarded. For example, exchanging the subject and object while retaining the same tokens leaves this representation unchanged. No more expressive downstream classifier can distinguish that pair without additional features. A zero row is an exception to unit-vector geometry and must remain zero rather than being 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 13.1. Leakage-safe TF-IDF model selection

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 13.1

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. $c^* \leftarrow \arg\max_{c \in \mathcal{C}_{\text{ok}}} a\, s_c$

$v^* \leftarrow \text{VecFit}(\mathcal{D}_{\text{dev}}, c^*)$

$h^* \leftarrow \text{Fit}(v^*(\mathcal{D}_{\text{dev}}), y_{\text{dev}}, c^*)$

Freeze Include label mapping and preprocessing. $\text{Save}(v^*, h^*, c^*)$

Algorithm 13.1 (continued)

17 Exactly once. $\widehat{y} \leftarrow h^*(v^*(\mathcal{D}_{\text{test}}))$

18 **Return** $(v^*, h^*, c^*), \{s_c\}, \widehat{y}$

Why this works. The final label vector is abbreviated 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.

Notation for Figure 13.1

A	Document identifier for the three-word sentence customer refunded merchant in this constructed illustration.
B	Document identifier for merchant refunded customer; its words match document A but their semantic roles differ.
C	Document identifier for the two-word sentence parcel arrived; these capital letters name documents, not matrices.

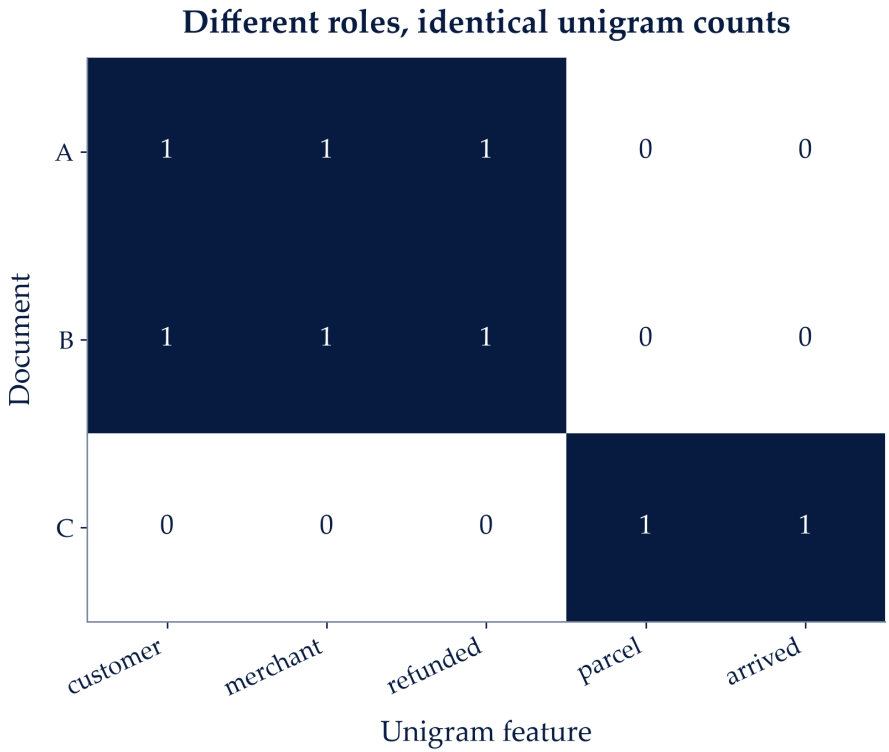


Figure 13.1. Unigram counts for three teaching documents: A is customer refunded merchant; B is merchant refunded customer; C is parcel arrived. Swapping subject and object leaves the unigram counts unchanged. No classifier can recover discarded role information from these count vectors alone.

Notation for Figure 13.2

A, B, C	The same three document identifiers: customer refunded merchant, merchant refunded customer, and parcel arrived, respectively.

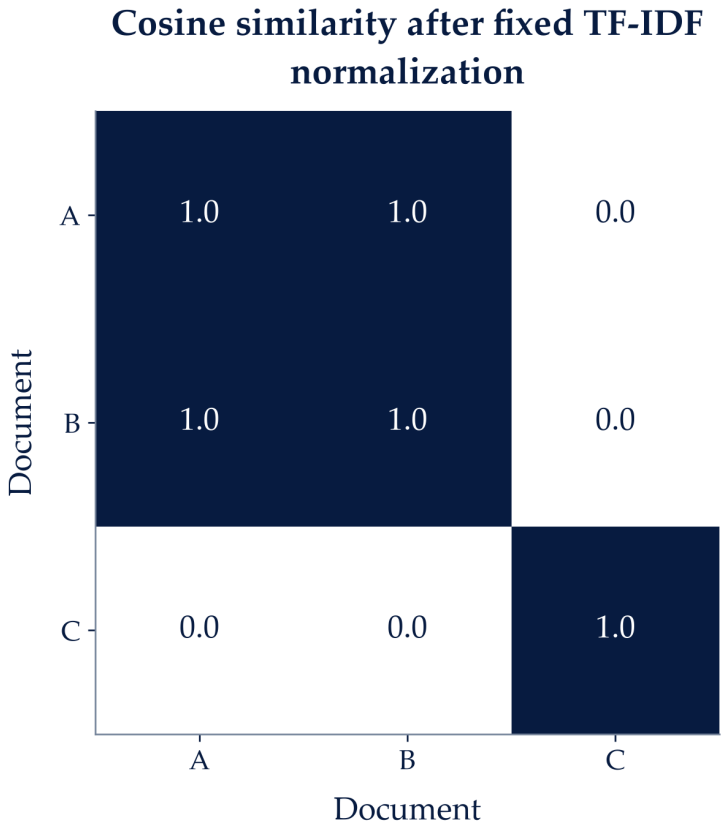


Figure 13.2. Cosine similarity after smoothed-IDF weighting and unit normalization of the same three documents. Documents A and B are identical in this representation despite their different subject-object roles. Vocabulary and IDF are fitted to this tiny corpus solely for arithmetic illustration.

13.3 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 combines class priors with class-specific token probabilities. Conditional on class and document length, token identities are drawn independently; vocabulary counts are dependent because their sum is fixed. Real language violates this token assumption, yet classification can remain useful. Additive smoothing avoids a zero document probability caused by an unseen class-term pair. Use nonnegative counts for the literal generative interpretation. Nonnegative TF-IDF weights are a common practical substitute, but fractional values are not observed token counts and do not restore the model's independence assumptions.

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 [13.4] demonstrates the suitability of support-vector methods for high-dimensional text features. Wang and Manning [13.3] show the strength of carefully constructed linear and naive-Bayes-inspired baselines, motivating a serious sparse-model benchmark before attributing improvements to a more elaborate representation. McCallum and Nigam [13.7] compare Bernoulli word-presence and multinomial word-count event models for naive Bayes text classification. The event model determines what the likelihood means; it should not be replaced by an informal claim that arbitrary count coordinates are independent.

Notation for equations (13.6), (13.7)

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.

$$s_c(x) = \log p(c) + \sum_{j=1}^{d} x_j \log p(v_j \mid c) \tag{13.6}$$

$$\hat{c} = \arg\max_c s_c(x) \tag{13.7}$$

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 conditions on document length and either treats it as class-independent or uses class priors conditional on that length. Its class-independent multinomial coefficient cancels in the decision. Fractional feature weights are a practical extension, not literal token counts.

Notation for equation (13.8)

$p(v_j \mid c)$	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.

$$p(v_j \mid c) = \frac{N_{cj} + \alpha}{\sum_{r=1}^{d} N_{cr} + \alpha d} \tag{13.8}$$

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 13.2: 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

$$p(x \mid c, N) = \frac{N!}{\prod_{j=1}^{d} x_j!} \prod_{j=1}^{d} \theta_{cj}^{x_j} \tag{13.T6}$$

$$N = \sum_{j=1}^{d} x_j \tag{13.T7}$$

There are factorially many token orderings compatible with one count vector. Every such ordering has the same product probability under the independent-token assumption, so summing their probabilities gives the multinomial coefficient. In a class likelihood ratio that coefficient is identical in numerator and denominator and cancels. Bayes' rule multiplies the remaining ratio by prior odds.

$$\log \frac{p(c = 1 \mid x, N)}{p(c = 0 \mid x, N)} = \beta_0 + \sum_{j=1}^{d} x_j \beta_j \tag{13.T8}$$

$$\beta_0 = \log \frac{\pi_1}{\pi_0} \tag{13.T9}$$

$$\beta_j = \log \frac{\theta_{1j}}{\theta_{0j}} \tag{13.T10}$$

Taking logarithms after cancellation proves the affine rule. Solving the odds equation gives a logistic transform of that score. This resembles discriminative logistic regression at prediction time but does not make the fitting procedures equivalent: Naive Bayes estimates token probabilities, while logistic regression optimizes a conditional classification objective.

If document length has a class-dependent likelihood, add its log likelihood ratio to the score when using unconditional class priors. Equivalently, use priors conditional on the observed length, but then the intercept can depend on length: there need not be one affine boundary across all document lengths. Substituting fractional TF-IDF weights into the scoring rule also no longer describes the literal multinomial count model. Independence is a modeling assumption, not a statement that language lacks syntax; correlated terms can cause overconfident probabilities even when classification remains useful.

□

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 13.3: 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.

Complete supporting proof: Mathematical Supplement, A12 (in this book).

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.

13.4 Frozen pretrained embeddings provide a different representation

An embedding maps a token, sentence, or document into a dense vector learned from a training objective. Static word embeddings assign one vector to a vocabulary item regardless of context. Contextual encoders compute representations using surrounding tokens, so the same word can contribute differently in different sentences. A sentence embedding additionally requires a pooling or output convention that turns variable-length token information into one fixed-length vector. Use the convention intended by the pretrained model rather than arbitrarily averaging incompatible outputs.

A frozen encoder keeps its pretrained parameters fixed while a downstream classifier learns from the task's training labels. This separates representation extraction from classifier fitting and can reduce task-specific optimization cost. It still uses external information: the encoder's pretraining corpus, objective, tokenizer, and learned weights are part of the method. Fine-tuning updates some or all encoder parameters using task data and is a different procedure with different compute, validation, and overfitting considerations. Random vectors are not a substitute for pretrained semantic representations.

Read the model card for intended language, input length, pooling, license, and known limitations. Truncation can silently remove the sentence carrying the classification label, and domain-specific terminology may be poorly represented. Record the model identifier and resolved revision, freeze evaluation behavior, and avoid sending private text to a service without authorization. The optional extension downloads genuine public weights and runs locally; it remains separate from the offline core and reports validation results without silently expanding the required dependency set.

Mikolov and colleagues [13.5] learn distributed word and phrase representations from context. Reimers and Gurevych [13.6] adapt a pretrained encoder to produce useful sentence embeddings; the distinction matters because averaging word vectors and encoding a whole sentence are different representation models. Devlin and colleagues [13.8] learn bidirectional contextual representations through pretraining and adapt them by downstream fine-tuning. That procedure differs from freezing a pretrained encoder and fitting a separate classifier, and neither should be confused with the offline sparse baseline executed here.

Notation for equations (13.9), (13.10)

$\mathcal{D}_i$	Example-i text before encoder tokenization/preprocessing.
i	Document index, unchanged throughout encoding and classification.
g_{ϕ_0}	Fixed pretrained text encoder, including declared document-pooling convention.
ϕ_0	Frozen encoder parameters; zero identifies the starting checkpoint, not iteration.
z_i	Fixed-length dense embedding column for document i in the encoder feature space.
h_θ	Embedding-to-prediction classifier fitted on task training labels.
θ	Trainable classifier parameters; encoder parameters remain frozen.
$\hat{y}_i$	Document-i class or class-score prediction under the declared output convention.

$$z_i = g_{\phi_0}(\mathcal{D}_i) \tag{13.9}$$

$$\hat{y}_i = h_\theta(z_i) \tag{13.10}$$

The document D_i is encoded as dense vector z_i by pretrained encoder g with fixed parameters ϕ_0 . The downstream classifier h has trainable parameters theta. Frozen extraction updates theta but not ϕ_0 .

All document embeddings have the encoder's declared output dimension. This is frozen feature extraction, not encoder fine-tuning.

Notation for equation (13.11)

z_a, z_b	Nonzero equal-dimensional embedding columns from compatible encoders and preprocessing.
a, b	Compared document/encoded-item indices.
$\cos(z_a, z_b)$	Directional cosine similarity, between negative one and one.
$z_a^T z_b$	Scalar Euclidean inner product of the embeddings.
$\|z_a\|_2$	Positive first-embedding Euclidean norm, removing its magnitude.
$\|z_b\|_2$	Positive second-embedding Euclidean norm, removing its magnitude.

$$\cos(z_a, z_b) = \frac{z_a^T z_b}{\|z_a\|_2 \|z_b\|_2} \tag{13.11}$$

Cosine similarity compares two nonzero embedding vectors z_a and z_b by direction. High cosine similarity reflects the encoder's learned geometry, not a universal guarantee of equivalent meaning or identical class labels.

Cosine similarity is undefined for a zero vector. Similar directions do not guarantee equivalent meaning or the same target class.

◇ Proposition 13.4: Masked mean pooling, normalization, and directional stability

Notation for this result

T, i	Positive number of token positions including padding, and token-position index from one through that number.
d	Positive encoder feature dimension in this proposition, not the lexical vocabulary size.
h_i, m_i	Length-d encoder column at position i and its inclusion indicator, equal to zero or one.
q, s, u	Positive included-token count, their length-d vector sum, and their length-d mean-pooled embedding, respectively.
z	Unit-length normalized pooled embedding, defined when the masked sum is nonzero.
v	Perturbed nonzero pooled embedding with the same dimension as the original pooled embedding u.

Assumptions. An encoder provides compatible token vectors in one common feature space. Binary masks distinguish included tokens from padding, and at least one token is included. Mean pooling is appropriate only if the selected pretrained model specifies it. Normalization is applied only when the pooled vector is nonzero.

Holding the included token vectors fixed, masked mean pooling is invariant to adding masked padding. Normalizing the pooled mean gives the same direction as normalizing the masked sum. Nevertheless the direction can be unstable when token contributions nearly cancel; a quantitative bound depends inversely on the pooled norm.

Complete supporting proof: Mathematical Supplement, A9 (in this book).

Correct masking and normalization are part of the model definition, not interchangeable implementation details. Algebraic validity of a pooled vector does not prove semantic usefulness or compatibility with a particular pretrained checkpoint.

▶ Example: Freeze means one part of the system stops learning

An encoder producing 384 coordinates maps 600 training documents to a 600 by 384 matrix. A three-class linear head has 384 weights and one intercept per score: 1155 fitted parameters before parameterization redundancy. Frozen training optimizes only this head; the encoder remains unchanged. Transform validation documents with the same encoder and pass them through the fitted head. Updating encoder weights with training labels is fine-tuning, a different experiment whose training cost and tuning choices must be reported separately. This dimension calculation does not claim that an optional pretrained encoder was actually executed.

Interpretation and limits

Do not label random vectors or arbitrary hand-assigned numbers as pretrained embeddings. Genuine pretrained comparisons must identify the real model, revision, tokenizer, and external-data advantage.

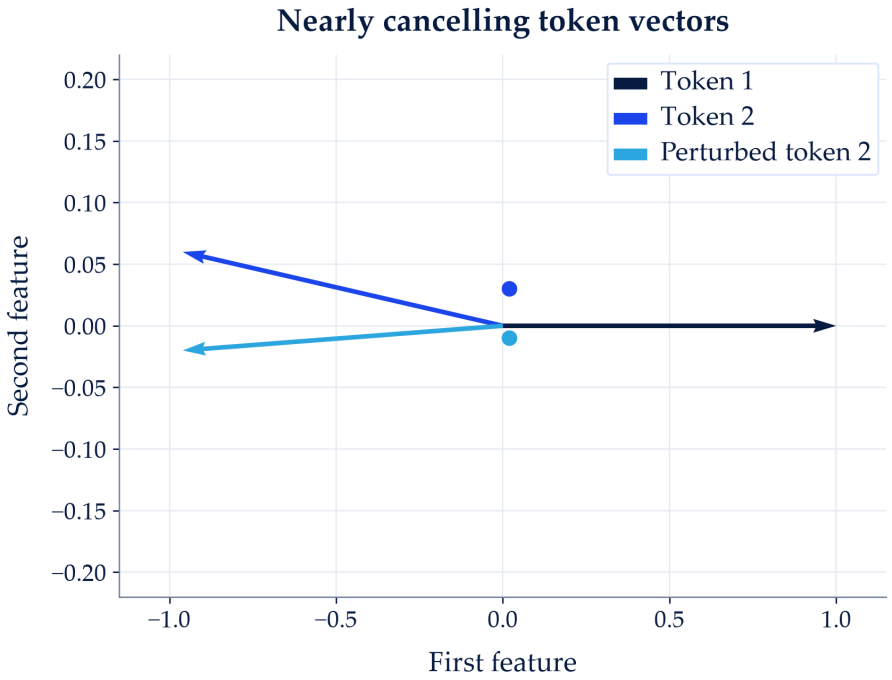


Figure 13.3. Token vectors (1, 0) and (-0.96, 0.06) nearly cancel. Changing the second token to (-0.96, -0.02) is a perturbation of magnitude 0.08. Dots identify the two pooled means near the origin. These are constructed vectors, not outputs from a pretrained encoder.

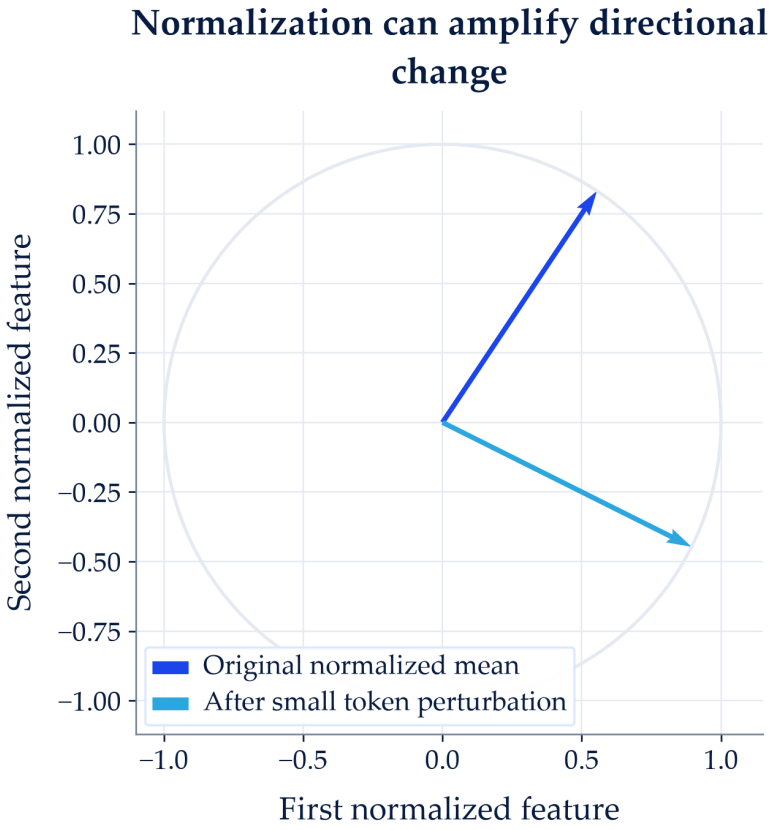


Figure 13.4. Unit normalization of the two near-zero pooled means produces a large directional change after the small token perturbation. The instability occurs because the pooled norm is small. A representation pipeline must handle zero and near-zero mean norms explicitly.

Optional interactive companion | Context changes a representation

Self-attention mixes token information using input-dependent weights. Query and key projections determine matching scores; value projections supply the information being combined. Trace the displayed projections and mixing, then distinguish the resulting contextual representation from a static word vector. The attention pattern describes a computation: it is not automatically an explanation of a classifier, evidence of learned semantics, or a sentence embedding.

Open Self-Attention Lab

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

13.5 Compare representations fairly and inspect failures

Fair comparison starts with the same prediction target, eligible examples, partition rules, and metric definitions. Keep duplicate documents, near-duplicate templates, and related messages from one conversation together when they could leak information across partitions. Time-based evaluation may be necessary when vocabulary and label policies change. Match tuning budgets where practical, and disclose when pretrained representations use external data unavailable to lexical baselines. This does not invalidate pretraining; it identifies what the comparison actually 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, document length, language, rare terms, negation, and source. Distinguish representation blindness from an incorrectly labeled or genuinely ambiguous example. Challenge cases help expose limitations but do not replace a representative evaluation sample. Freeze the

selected pipeline before one final test evaluation, and resist changing it after seeing test mistakes without acknowledging a new development cycle. The small bundled corpus teaches this protocol; production conclusions require a larger, representative dataset and uncertainty around the resulting estimates.

Wang and Manning's controlled baseline comparisons [13.3] and the task-based evaluation of Sentence-BERT [13.6] motivate comparing complete pipelines under shared splits and metrics. Better semantic similarity on a benchmark does not by itself establish better classification on a new domain. Gao, Yao, and Chen [13.9] train sentence embeddings with a contrastive objective and evaluate semantic textual similarity. Those results motivate examining the encoder's objective and evaluation task; they do not establish performance on this chapter's frozen text-classification split.

Notation for equation (13.12)

$\widehat{R}_{\text{test}}$	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.

$$\widehat{R}_{\text{test}} = \frac{1}{n_{\text{test}}} \sum_{i \in \mathcal{T}} \ell(\hat{f}(\mathcal{D}_i), y_i) \tag{13.12}$$

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, scores of 10 and 11 correct differ by one net correct prediction, or approximately 8.333 percentage points. They need not differ on only one document: the second model might fix two errors while introducing

one. Build a paired correctness table and inspect both gains and regressions without tuning on the test set. This small net count does not establish general superiority; wording, sampling, and ambiguous labels matter. Seek larger representative evaluation samples and use repeated development experiments for selection. Compare latency and maintenance requirements 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.

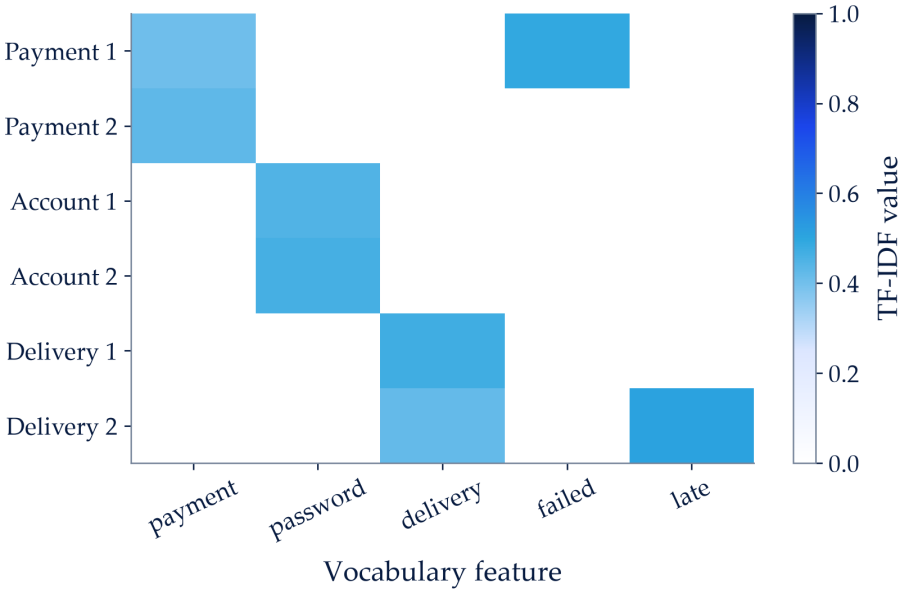


Figure 13.5. Actual TF-IDF values for six explicitly constructed teaching sentences. Shared words create similar coordinates, but synonyms with no lexical overlap need not be close. These are TF-IDF features, not pretrained semantic embeddings. Vocabulary fitting belongs inside the training boundary.

Table 13.1. 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 13.2. 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.

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 13.3. 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 was 0.884 for Naive Bayes, 0.867 for logistic regression and 0.881 for linear SVM. A manual check reproduced the observed term's document-frequency-based IDF. The final comparison supports this frozen lexical procedure on the declared 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.

Historical saved run; preserve filtering counts, source indices and fitted vocabulary. No pretrained encoder or external inference service was evaluated.

▶ Example: inspecting the real data

For the actual training vocabulary term space, when present, the notebook prints its document frequency and fitted IDF. Recompute log((1 plus the number of fitting documents)/(1 plus that document frequency)) plus one and compare it with the library value. The printed fallback term is used if space is absent; no corpus frequency is invented.

Dataset source and documentation

Companion: Week 13, Real-Data Case Study notebook. Reproduce the declared run, then investigate a changed condition. No fixed benchmark score is promised.

Measured Extension: A Genuine Frozen Encoder On The Same Public-Text Task

The preceding historical case executed lexical models only. This separate, newly executed development experiment adds actual pretrained representations. It asks whether a frozen sentence encoder helps a small four-topic classifier, and what additional computation and input restrictions come with that representation. The corpus is the same public 20 Newsgroups task: graphics, baseball, hockey, and space. It is not a test of general language understanding, nor a new final score for the earlier selected lexical procedure.

Protocol fixed before this run. Reuse the original-training-only 75/25 fitting/validation boundary with seed 1310, after the same header/footer/quotation removal, whitespace normalization, blank removal, and keep-first exact deduplication. This leaves 2,307 training-origin messages. Take stratified subsets of 400 fitting and 200 validation messages, with seeds 1312 and 1313 respectively. The same retained rows and class order enter both arms. Their class counts are (99, 100, 101, 100) for fitting and (49, 50, 51, 50) for validation. No official-test message or label enters fitting or evaluation.

The lexical arm fits unigram/bigram TF-IDF on the 400 fitting documents only, with minimum document frequency two, at most 15,000 features, raw counts, smoothed IDF, and L2 normalization. The encoder arm uses sentence-transformers/all-MiniLM-L6-v2 at the exact public revision linked below. It remains in evaluation mode, all parameters are frozen, and inference runs locally without gradient recording. Both arms fit logistic regression with C equal to two, the lbfgs solver, and a 1,000-iteration limit. There is no hyperparameter search, fine-tuning, or validation refit. Equal settings control this comparison but do not imply equally optimal regularization in the two feature spaces.

Table 13.B1. The encoder contract actually executed

Component	Recorded setting
Weights	all-MiniLM-L6-v2; 22,713,216 parameters; safetensors
Tokenizer	BertTokenizerFast; uncased WordPiece; fixed vocabulary
Input limit	256 token positions including special tokens; longer inputs truncated
Pooling	Mean of final token vectors weighted by the attention mask; unmasked special tokens included
Normalization	One L2-normalized 384-coordinate vector per document
Model license	Apache-2.0, as declared by the pinned model card
Execution	Local CPU; two PyTorch threads; batches of 16; no text sent to an inference API

The model card identifies a pretrained short-text encoder further trained with a contrastive objective on a large external sentence-pair collection. That

information is an advantage unavailable to the fitting-only lexical arm. The experiment neither reconstructs that pretraining nor establishes that these historical messages were absent from it. The model's license also does not independently establish redistribution rights for every corpus message. The released experiment preserves source indices and hashes rather than redistributing message text.

Trace one observed example. The first selected fitting message is source row 400 in the loader's fixed shuffled training-origin collection, labeled space. It tokenizes to 257 positions including special tokens, of which 256 enter the encoder. Its masked pooled vector has norm approximately 1.847 before normalization. The first six normalized coordinates are approximately (0.047, 0.064, 0.048, 0.008, 0.156, -0.024). These are outputs from the pinned model, not hand-assigned semantic numbers. A coordinate has no declared standalone topic meaning; the fitted head uses the complete vector.

Table 13.B2. Measured validation results on the same 200 messages

Representation + same logistic head	Features	Accuracy	Macro F1	Log loss
Fitting-only TF-IDF	10,363	0.805	0.805	0.890
Frozen MiniLM	384	0.910	0.910	0.374

The encoder arm classified 182 of 200 validation messages correctly, versus 161 for TF-IDF. Pairing their predictions is more informative than comparing rounded totals: both were correct on 152 messages, only the encoder was correct on 30, only the lexical arm was correct on nine, and both were wrong on nine. Thus the improvement does not mean the encoder dominates on every document. These are observations from one development sample and two fixed configurations, not a population confidence interval or independent final-test confirmation.

Table 13.B3. Measured local times in seconds, separated by operation

Operation	TF-IDF arm	Encoder arm
Fitting-set feature computation	0.716	41.769
Validation feature computation	0.193	25.742
Classifier fitting	0.340	0.029

The short classifier fit does not make the dense pipeline cheap: encoding dominates its local work. Timings are one Windows CPU run with batch size 16 and two PyTorch threads; they are not an isolated hardware benchmark. They exclude model pretraining, public-data acquisition, and model download/loading. The downloaded safetensors file contains 90,868,376 bytes; download/cache resolution took approximately 51.212 seconds and loading approximately 0.951 seconds in this run. Neither download time nor one warm software environment is a portable speed promise.

Failure analysis. Truncation affected 91 of 400 fitting documents and 42 of 200 validation documents. In contrast, the lexical arm saw each whole cleaned document and produced two all-zero validation rows. This compares complete representation pipelines with unequal external information and different input coverage; it does not isolate a pure architectural effect. No claim is made that a discarded tail was irrelevant. Exact deduplication and heuristic metadata removal do not eliminate near duplicates or every shortcut, and historical language need not represent contemporary messages. All pooled norms exceeded one in this run, but that observation does not remove the zero-vector safeguard required by the general pooling definition.

Reproducibility companion. The separate Frozen_Encoder_Comparison notebook audits the saved per-document predictions and provides an opt-in full local reproduction using run_encoder.py. Its declared protocol, selected source indices, weight hashes, and saved output arrays permit independent rescoring; the notebook contains the learner exercises. A stronger comparative claim would require a frozen full procedure and suitable new held-out data, with paired uncertainty analysis and checks for duplicates and possible pretraining overlap.

Reproduction record. The saved run used Python 3.12.14, NumPy 2.5.3, scikit-learn 1.9.1, PyTorch 2.14.1+cpu, Transformers 4.44.2, and tokenizers 0.19.1. The companion records all selected source rows, cleaned-text hashes,

exact model-file hashes, package versions, predictions, confusion counts, and the declared protocol. Public weights are downloaded to a local cache; no remote model code is executed. This makes the numerical claim auditable without presenting one small success as a universal preference for embeddings.

Executable companion notebook: Frozen Encoder Comparison, including saved prediction audits and optional local reproduction. Source

Pinned model card and files: sentence-transformers/all-MiniLM-L6-v2, revision 1110a243fdf4706b3f48f1d95db1a4f5529b4d41. Source

scikit-learn dataset documentation: original 20 Newsgroups split and optional metadata removal. Source

Chapter Capability

Compare text representations within fixed data boundaries, keep vocabularies training-only, and inspect errors without using test documents to tune the model.

Week 13: separate practice, solutions and assessment

References

[13.1] Manning, C. D.; Raghavan, P.; Schütze, H. (2008). Introduction to Information Retrieval. Cambridge University Press. Source / DOI

[13.2] Salton, G.; Buckley, C. (1988). Term-weighting approaches in automatic text retrieval. Information Processing & Management 24(5), 513-523. Source / DOI

[13.3] 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

[13.4] Joachims, T. (1998). Text categorization with Support Vector Machines: Learning with many relevant features. Machine Learning: ECML-98, 137-142. Source / DOI

[13.5] 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

[13.6] Reimers, N.; Gurevych, I. (2019). Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. Proceedings of EMNLP-IJCNLP, 3982-3992. Source / DOI

[13.7] 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

[13.8] 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

[13.9] 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

From machine learning to deep learning

Deep learning extends the ideas developed here: objectives define what is fitted, gradients change parameters, regularization constrains flexibility, and representations determine which relationships are easy to express. Larger architectures do not remove the need for clean experimental boundaries, informative baselines or careful diagnosis.

Spatial structure motivates convolution and visual representation learning. Sequences and context motivate attention and language models. Action, dynamics and feedback motivate physical AI. In each case, the central statistical questions remain: what information is available, which assumptions are being made, how are the parameters estimated, and what evidence supports performance outside the fitting data?

A reproducible model is more than a set of weights. It includes the preprocessing, feature definitions, software environment and decision rule that give those weights meaning. The most useful foundation for future architectures is therefore not familiarity with one library interface, but the ability to reason precisely about data, computation and evidence.

Further information

Visit the website
Contact the publisher

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. 11.5

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; 13.3

Batch. A set of training observations used together to compute an update. The aggregation convention for the loss determines the scaling of its gradient. 12.1

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. 12.4

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. 13.4

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. 11.4

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. 13.3

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. 11.1; 11.2

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; 12.3

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. 13.2

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. 337
Algorithm - 2.3, p. 63; 8.1, p. 262
Anomaly detection - 8.5, p. 284
Area under the ROC curve (ROC AUC) - 3.3, p. 100
Automatic differentiation - 11.5, p. 390
Bagging - 7.3, p. 241
Baseline - 1.3, p. 24; 13.3, p. 441
Batch - 12.1, p. 397
Bayesian optimization - 10.5, p. 344
Bias-variance decomposition - 1.5, p. 32
Calibration - 9.1, p. 292; 9.2, p. 297
Checkpoint - 12.4, p. 418
Class imbalance - 3.5, p. 122
Clustering - 8.1, p. 262; 8.3, p. 273; 8.4, p. 276
Conformal prediction - 9.5, p. 311
Confusion matrix - 3.3, p. 100
Cross-validation - 4.1, p. 134; 4.3, p. 142
Data leakage - 1.3, p. 24; 4.3, p. 142
DBSCAN - 8.4, p. 276
Decision threshold - 3.4, p. 114
Dimensionality - 1.4, p. 28; 5.1, p. 164
Embedding - 13.4, p. 448
Empirical risk - 1.2, p. 20
Ensemble - 7.3, p. 241; 7.4, p. 245; 7.5, p. 251
Exchangeability - 9.5, p. 311
Feature - 1.1, p. 18; 2.1, p. 50
Finite difference - 11.4, p. 384
Gaussian process - 10.1, p. 327; 10.3, p. 333
Generalization - 1.2, p. 20; 1.3, p. 24
Gradient boosting - 7.4, p. 245

Gradient descent - 2.3, p. 63
Hierarchical clustering - 8.3, p. 273
Hyperparameter - 4.4, p. 144
Inductive bias - 1.5, p. 32
K-means - 8.1, p. 262
K-nearest neighbors (KNN) - 6.1, p. 196
Kernel - 6.4, p. 208; 10.2, p. 330
Likelihood - 2.2, p. 56; 3.2, p. 91
Logistic regression - 3.1, p. 89; 3.2, p. 91
Loss function - 1.2, p. 20; 3.2, p. 91
Margin - 6.2, p. 198
Naive Bayes - 13.3, p. 441
Nested cross-validation - 4.4, p. 144
Neural network - 11.1, p. 359; 11.2, p. 362
Overfitting - 1.5, p. 32; 4.4, p. 144
Partial least squares (PLS) - 5.4, p. 179
Pipeline - 2.1, p. 50; 4.3, p. 142
Population risk - 1.2, p. 20
Precision and recall - 3.3, p. 100
Principal component analysis (PCA) - 5.2, p. 167; 5.3, p. 177
Proper scoring rule - 9.1, p. 292
Random forest - 7.3, p. 241
Regularization - 2.4, p. 71; 12.3, p. 409
ROC curve - 3.3, p. 100
Support vector - 6.3, p. 202
Support-vector machine (SVM) - 6.2, p. 198; 6.4, p. 208
Support-vector regression (SVR) - 6.5, p. 217
t-SNE - 5.5, p. 188
Test lock - 4.5, p. 154
TF-IDF - 13.2, p. 432
Validation set - 1.3, p. 24; 4.4, p. 144

About The Author



Dr. Barak Or

Dr. Barak Or is an artificial intelligence researcher, entrepreneur, author and educator. His work connects machine learning and deep learning with autonomous systems, navigation and practical engineering.

He is the founder of ArtificialGate, whose work spans applied AI research, consulting, implementation and technical education. His research output includes dozens of AI papers and five patents.

His writing and teaching bring mathematical reasoning together with implementation: formulating a problem, stating assumptions, designing experiments and interpreting evidence. He emphasizes understanding when a method works, why it can fail and how to test a proposed improvement.

Modern Machine Learning reflects this approach. It develops classical methods as useful tools in their own right and as a foundation for deep learning, combining theory and proofs with examples, algorithms and experiments on established datasets.

Mathematical Supplement

A mathematical refresher, finite-class generalization, and the guarantees behind gradient descent. This appendix extends the main book without changing its practical reading route.

Read S1 as needed before Chapters 1-2, S2 alongside Chapter 1, and S3 alongside Chapter 2. Return to S4 when interpreting the training experiments in Chapter 12. Assumptions and conclusions are deliberately separated: a guarantee for one objective, distribution or algorithm cannot silently certify another.

Exercises, prerequisite diagnostics and solutions remain in the separate course-aids package. The numerical examples below illustrate stated mathematical assumptions; they are not measured industry performance claims.

Return to the study route

S1 Mathematical Refresher: The Tools Used In The Proofs

Read this section before S2 if probability inequalities are unfamiliar, or before S3 if vector derivatives need refreshing. It supplies the specific tools used below, not a replacement for a linear algebra or probability course.

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.

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

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

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

S2 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.

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

$\psi, \log$ 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.

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

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.

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

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

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 epsilon 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.

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

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

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

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.

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

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

Theorem: Uniform Accuracy And Selection Error

Let delta be a chosen failure probability strictly between zero and one. Define the deviation allowance epsilon_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 epsilon_n into the preceding tail bound. Because it holds for every member at once, it also holds for a member selected using D. Let h_hat 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_star 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.

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

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

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

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

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

S3 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.

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

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.

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

One Exact Gradient Step

Let theta_k be the parameter vector at nonnegative integer iteration k. Let g_k denote the exact full gradient there, and let eta be a positive constant step size. Substitute the gradient-descent update into the lemma. This gives a decrease whenever eta 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 eta no greater than one divided by L, the coefficient of the squared gradient is at least eta divided by two. In the

particular choice eta 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 theta_star. Write F_star 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 eta no greater than one divided by L, starting from any theta_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 theta_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 eta 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 eta 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 eta 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 theta equal to pi 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 rho 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

Input. A continuously differentiable objective F and exact gradient; initial p-entry vector theta_0; known positive valid smoothness bound L; fixed positive eta no greater than 1/L; positive integer update limit K; nonnegative gradient-norm tolerance rho.

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

1	Initialize theta to theta_0 and the accepted-update count to zero.
2	Evaluate F and the exact full gradient at the current theta. If either contains a nonfinite value, return numerical failure without accepting another update. Record the values.
3	If the gradient norm is at most rho, return gradient tolerance. Otherwise, if K updates have already been accepted, return update budget.
4	Form the candidate by subtracting eta times the current gradient from theta. Evaluate its objective. If the candidate or objective is nonfinite, return numerical failure at the last accepted theta.
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

S4 Using A Theorem Without Overstating Its Conclusion

Translate the new results into the regression, validation and neural-network workflows already used in the book. Optimization error and statistical error remain different quantities.

Example: A Nonlinear Objective With A Certified Step

Consider the scalar objective F defined below, with its scalar argument theta measured in a chosen dimensionless coordinate system. The natural logarithm is log, exp is the natural exponential, and lambda is a strictly positive penalty coefficient. The logistic function sigma gives a compact derivative. This example supplies a nonlinear smooth objective without claiming that every neural-network objective has the same geometry.

θ, F, λ Scalar real parameter, scalar objective in this example, and strictly positive quadratic-penalty coefficient.

$$F(\theta) = \log(1 + e^{\theta}) + \frac{\lambda}{2}\theta^2 \tag{S4.1}$$

σ Logistic function with values strictly between zero and one.

$$\sigma(\theta) = \frac{1}{1 + e^{-\theta}} \tag{S4.2}$$

F', F'' First and second ordinary derivatives of the scalar objective.

$$F'(\theta) = \sigma(\theta) + \lambda\theta \tag{S4.3}$$

$$F''(\theta) = \sigma(\theta)(1 - \sigma(\theta)) + \lambda \tag{S4.4}$$

The second derivative is positive and at most lambda plus one quarter everywhere. Thus F is convex and a valid global smoothness bound L is lambda plus one quarter. Its quadratic term makes it tend to infinity in both directions, so it attains a minimum. With lambda equal to 0.5, L equal to 0.75 is valid and eta equal to four thirds satisfies S3. Compute the log term with a numerically stable softplus implementation; the displayed mathematical expression is not an instruction to overflow an exponential.

Regression Normalization And Scaling

For the half-mean-squared objective in S1 and Chapter 2, the Hessian is the design transpose times the design divided by the sample count. Its largest eigenvalue is a valid global gradient-Lipschitz constant. If the design is identically zero, every gradient is zero; any positive upper bound suffices for the smoothness statement. A rank-deficient design may have many coefficient minimizers, but the convex objective-gap theorem remains valid for any chosen minimizer. It does not require strong convexity.

Multiplying an objective by a positive constant leaves its minimizers unchanged but multiplies its gradient and smoothness bound by that constant. The corresponding conservative step must shrink reciprocally. Changing feature units changes the Hessian differently and may change a regularization preference; it is not merely a cosmetic relabeling. These facts explain why the mean-loss convention and the unpenalized intercept in Chapter 2 must be matched before comparing a matrix calculation with a library fit.

Optimization Success Is Not Prediction Success

The convex theorem compares the current objective with the minimum of that same objective on the given training problem. The finite-class theorem compares population losses among a specified fixed collection under i.i.d. sampling. To connect them, the objective optimized must actually correspond to the empirical risk and tolerance used in the statistical theorem. A penalty changes that objective; a neural parameter family is generally infinite; a mini-batch update is not an exact full gradient. None of these differences should be silently removed when presenting a guarantee.

For the Fashion-MNIST optimizer comparison in Chapter 12, a declining training curve is evidence about the executed optimization run. The convex objective-gap guarantee does not apply to that nonconvex network. Even the smooth nonconvex result requires an appropriate smoothness bound and exact gradients, assumptions not supplied by merely choosing a small learning rate. The useful empirical comparison therefore keeps the data split, initialization and computation budget controlled, tunes on development data, and reports variability. It should not replace that evidence with a theorem about a different algorithm.

A practical reading order is S1 then S2 alongside Chapter 1, and S1 then S3 alongside Chapter 2; return to this section before interpreting Chapter 12's

training curves. The proofs provide an assumption checklist, not a universal certificate. All displayed numerical examples in this supplement are declared mathematical illustrations. Dataset-specific performance belongs to the measured case results and their evaluation boundaries.

References

Lieven Vandenberghe, Gradient Method, lecture notes (2022) Source

Shai Shalev-Shwartz and Shai Ben-David, Understanding Machine Learning: From Theory to Algorithms (2014), Chapters 4 and 12 Source

A1 Local Coverage In A Uniform Hypercube

Supporting proof for Chapter 1, Section 1.4. Original T-series equation numbers are retained.

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**

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

A2 ROC Area Is Pairwise Ranking Accuracy With Half-Credit For Ties

Supporting proof for Chapter 3, Section 3.3. Original T-series equation numbers are retained.

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**

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

A3 The Estimand And Dependence Structure Of Cross-Validation

Supporting proof for Chapter 4, Section 4.1. Original T-series equation numbers are retained.

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**

$$n_{\mathrm{fit}} = n\frac{V-1}{V} \tag{4.T14}$$

$$S_v = \frac{1}{|D_v|}\sum_{i\in D_v} L(Y_i, A(D_{-v})(X_i)) \tag{4.T15}$$

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.

$$\mathbb{E}[S_v] = \mathbb{E}_{D_{n_{\mathrm{fit}}}}[R(A(D_{n_{\mathrm{fit}}}))] \tag{4.T16}$$

$$\widehat{R}_{CV} = \frac{1}{V}\sum_{v=1}^{V} S_v \tag{4.T17}$$

$$\mathbb{E}[\widehat{R}_{CV}] = \mathbb{E}_{D_{n_{\mathrm{fit}}}}[R(A(D_{n_{\mathrm{fit}}}))] \tag{4.T18}$$

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

A4 The First PLS Covariance Direction Is A Singular-Vector Problem

Supporting proof for Chapter 5, Section 5.4. Original T-series equation numbers are retained.

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

$$M = \frac{X_c^\top Y_c}{n-1} \tag{5.T19}$$

$$\operatorname{Cov}(X_c w,\, Y_c c) = w^\top M c \tag{5.T20}$$

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.

$$|w^\top M c| \le \|M c\|_2 \tag{5.T21}$$

$$\max_{\|c\|_2=1} c^\top M^\top M c = s_1^2 \tag{5.T22}$$

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

A5 Deriving The Soft-Margin Dual And Its Box Constraints

Supporting proof for Chapter 6, Section 6.3. Original T-series equation numbers are retained.

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

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

A6 Ward Linkage Is An Exact Increase In Squared Distortion

Supporting proof for Chapter 8, Section 8.3. Original T-series equation numbers are retained.

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

$$\mu = \frac{n_A \mu_A + n_B \mu_B}{n_A + n_B} \tag{8.T9}$$

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

A7 Expected Improvement From Its Defining Integral

Supporting proof for Chapter 10, Section 10.5. Original T-series equation numbers are retained.

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

$$F = \mu + sZ \tag{10.T15}$$

$$Z \sim \mathcal{N}(0, 1) \tag{10.T16}$$

$$b = f_{\text{best}} + \xi \tag{10.T17}$$

$$\delta = \mu - b \tag{10.T18}$$

$$z = \delta / s \tag{10.T19}$$

$$I = \max(F - b, 0) \tag{10.T20}$$

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.

$$\int_{-z}^{\infty} \phi(t)\, dt = \Phi(z) \tag{10.T22}$$

$$\int_{-z}^{\infty} t\phi(t)\, dt = \phi(z) \tag{10.T23}$$

$$\mathrm{EI} = \delta\Phi(z) + s\phi(z) \tag{10.T24}$$

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

A8 A Constructive One-Dimensional Approximation Result

Supporting proof for Chapter 11, Section 11.2. Original T-series equation numbers are retained.

Notation For This Result

$[a, b]$	Closed bounded input interval with distinct ordered endpoints.
t_j, m	Ordered grid points and positive number of grid intervals; point index j ranges from zero through m.
f	Continuous real target function on the specified interval.
s_j, j	Interpolating slope on interval j; the slope index ranges from zero through m-minus-one.
p, x	Polygonal approximation and an input within the closed target interval.
$\mathrm{ReLU}(u)$	The larger of zero and the real argument u; a shifted hinge at each interior grid point.
ω_f, δ	Modulus of continuity of the target and a nonnegative maximum input separation.
$u, v, \sup$	Two interval inputs and the least upper bound over the displayed eligible pairs.

Assumptions. The target is continuous on a closed bounded interval. Choose a finite ordered grid including both endpoints and interpolate target values linearly between neighboring grid points. Only uniform function-value approximation is claimed.

The polygonal interpolant can be represented exactly by a finite sum of shifted ReLUs plus an affine term. Making the largest grid gap sufficiently small gives any prescribed positive uniform error tolerance.

□ Proof

$$a = t_0 < t_1 < \cdots < t_m = b \tag{11.T4}$$

$$s_j = \frac{f(t_{j+1}) - f(t_j)}{t_{j+1} - t_j} \tag{11.T5}$$

$$p(x) = f(a) + s_0(x - a) + \sum_{j=1}^{m-1} (s_j - s_{j-1}) \operatorname{ReLU}(x - t_j). \tag{11.T6}$$

At the left endpoint the expression equals the target. Inside any grid interval, differentiating the active hinges produces a telescoping sum equal to that interval's prescribed slope. Continuity then establishes the correct value at every successive endpoint, proving exact equality with the polygonal interpolant. The affine input term can itself be implemented using a positive and negative ReLU pair, so this is a finite one-hidden-layer construction rather than an assumption about unavailable operations.

$$\omega_f(\delta) = \sup_{\substack{u,v \in [a,b] \\ |u-v| \le \delta}} |f(u) - f(v)| \tag{11.T7}$$

Inside one grid interval, the interpolant is a convex combination of its two endpoint values. Subtract the target value at the input and apply the triangle inequality. Both endpoint distances are bounded by the largest grid gap, so the interpolation error is bounded by the modulus at that gap. Continuity on a compact interval implies uniform continuity, and therefore this bound tends to zero as the grid is refined. Choosing a gap for which the bound is below the requested tolerance completes the proof.

□

This finite constructive result explains representational capacity in one dimension. It does not prove a general multivariate universal-approximation theorem or guarantee that training data and gradient descent recover the constructed weights.

Return to the chapter and its references

A9 Masked Mean Pooling, Normalization, And Directional Stability

Supporting proof for Chapter 13, Section 13.4. Original T-series equation numbers are retained.

Notation For This Result

T, i	Positive number of token positions including padding, and token-position index from one through that number.
d	Positive encoder feature dimension in this proposition, not the lexical vocabulary size.
h_i, m_i	Length-d encoder column at position i and its inclusion indicator, equal to zero or one.
q, s, u	Positive included-token count, their length-d vector sum, and their length-d mean-pooled embedding, respectively.
z	Unit-length normalized pooled embedding, defined when the masked sum is nonzero.
v	Perturbed nonzero pooled embedding with the same dimension as the original pooled embedding u.

Assumptions. An encoder provides compatible token vectors in one common feature space. Binary masks distinguish included tokens from padding, and at least one token is included. Mean pooling is appropriate only if the selected pretrained model specifies it. Normalization is applied only when the pooled vector is nonzero.

Holding the included token vectors fixed, masked mean pooling is invariant to adding masked padding. Normalizing the pooled mean gives the same direction as normalizing the masked sum. Nevertheless the direction can be unstable when token contributions nearly cancel; a quantitative bound depends inversely on the pooled norm.

□ **Proof**

$$q = \sum_{i=1}^{T} m_i \tag{13.T19}$$

$$s = \sum_{i=1}^{T} m_i h_i \tag{13.T20}$$

$$u = s/q \tag{13.T21}$$

With the included token vectors held fixed, an extra padding token contributes zero to the count and sum, whatever vector is assigned to that padding

position. Both quantities and their quotient therefore remain unchanged. This is an invariance of the pooling operation, not a guarantee that rerunning an arbitrary encoder leaves its unmasked outputs unchanged. Since the included-token count is positive, scaling the sum by its reciprocal affects magnitude but not its normalized direction.

$$z = \frac{u}{\|u\|_2} = \frac{s}{\|s\|_2} \tag{13.T22}$$

To quantify sensitivity, compare a nonzero pooled vector with a perturbed nonzero pooled vector. Add and subtract the perturbed vector divided by the original norm. The triangle inequality bounds the first difference by perturbation size divided by the original norm. The second difference becomes the absolute difference of the two norms divided by the original norm. The reverse triangle inequality bounds that norm difference by the perturbation size as well.

$$\left\| \frac{v}{\|v\|_2} - \frac{u}{\|u\|_2} \right\|_2 \leq \frac{2\|v - u\|_2}{\|u\|_2} \tag{13.T23}$$

This proves a local stability guarantee when the original norm is safely bounded away from zero. If contributions nearly cancel, that denominator is small and a small absolute perturbation can cause a large directional change. A deterministic zero-vector handling rule is therefore essential. Pooling is permutation invariant for a fixed collection of token vectors, but a contextual encoder may change those vectors when word order changes; the full encoder is not thereby permutation invariant. Averaging already normalized tokens is also a different operation because it removes each token's magnitude before pooling.

□

Correct masking and normalization are part of the model definition, not interchangeable implementation details. Algebraic validity of a pooled vector does not prove semantic usefulness or compatibility with a particular pretrained checkpoint.

Return to the chapter and its references

A10 The Marginal Likelihood Balances Residual Fit And Covariance Volume

Supporting proof for Chapter 10, Section 10.2. Original T-series equation numbers are retained.

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

$$e = y - m_X \tag{10.T1}$$

$$A = RR^T \tag{10.T2}$$

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

A11 A Precise Bound On Backward Gradient Amplification

Supporting proof for Chapter 12, Section 12.5. Original T-series equation numbers are retained.

Notation For This Result

l, L	Layer index from one through L and finite network depth; L is not the scalar loss in this proposition.
d_l	Positive width at level l, for levels zero through L; level zero is the input width.
a_0	Length-d-zero input column, before any affine layer.
a_l, b_l	Layer-l activation and bias columns, each of length d-l.
W_l	Weight matrix with d-(l-minus-one) input rows and d-l output columns.
ϕ_l, D_l	Coordinatewise activation and d-l-by-d-l diagonal matrix of its derivatives at the current preactivations.
J_l	d-l-by-d-(l-minus-one) Jacobian mapping a preceding-layer perturbation to the layer-l activation perturbation.
$\mathcal{L}, g_l$	Scalar loss of the network output and its column gradient with respect to layer-l activations.
g_0, g_L	Loss gradients with respect to the network input and final activation, respectively.
$\|\cdot\|_2$	Euclidean norm for vectors and induced spectral norm for matrices in this proposition.

Assumptions. A feedforward composition is differentiable at the evaluated input. Each layer has an affine preactivation followed by a coordinatewise activation. Use the spectral matrix norm and Euclidean vector norm; ReLU inputs exactly at a kink are excluded from the classical derivative claim.

The gradient reaching an early layer is bounded by the output gradient multiplied by the product of layer-Jacobian norms. Small factors can force

vanishing gradients; factors above one only permit, rather than guarantee, amplification.

□ **Proof**

$$a_l = \phi_l(W_l^T a_{l-1} + b_l) \tag{12.T26}$$

$$J_l = D_l W_l^T \tag{12.T27}$$

$$g_{l-1} = J_l^T g_l \tag{12.T28}$$

$$g_l = \nabla_{a_l} \mathcal{L} \tag{12.T29}$$

The chain rule gives the transpose-Jacobian recurrence. Repeated substitution expresses the input gradient as an ordered product of transpose Jacobians times the output gradient. The spectral norm is induced by the Euclidean norm, so each multiplication is bounded by its matrix norm times the current vector norm. Applying this inequality repeatedly proves the product bound.

$$\|g_0\|_2 \leq \|g_L\|_2 \prod_{l=1}^{L} \|J_l\|_2 \leq \|g_L\|_2 \prod_{l=1}^{L} \|D_l\|_2 \|W_l\|_2 \tag{12.T30}$$

For a diagonal derivative matrix, its spectral norm is the largest absolute activation derivative. The sigmoid derivative is at most one quarter, whereas the tanh derivative is at most one and becomes small in saturation. Thus, if every layer-Jacobian norm is bounded by the same constant strictly below one and the output-gradient norm remains bounded independently of depth, the input-gradient bound decays geometrically with depth. Large factors do not prove exploding gradients: directions may rotate into small singular directions, and cancellations can reduce the actual product. A single zero derivative can also block a coordinate while other paths remain active.

This result explains why activation scale, weight singular values, and depth must be inspected together. It does not justify treating a gradient norm alone as a measure of learning quality. Gradient clipping limits the final vector supplied to an optimizer, but it cannot restore a dependency already erased by a saturated or disconnected computational path.

□

Monitor layerwise activations and gradients rather than assuming that an optimizer name prevents vanishing or exploding signals. The bound is local to the executed forward pass.

Return to the chapter and its references

A12 Additive Smoothing Is A Posterior Predictive Calculation

Supporting proof for Chapter 13, Section 13.3. Original T-series equation numbers are retained.

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

$$p(\theta) \propto \prod_{j=1}^{d} \theta_j^{\alpha-1} \tag{13.T11}$$

$$\sum_{j=1}^{d} \theta_j = 1 \tag{13.T12}$$

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.

$$a_j = N_j + \alpha \tag{13.T13}$$

$$A = \sum_{j=1}^{d} a_j \tag{13.T14}$$

$$p(\theta \mid N) = \frac{1}{B(a)} \prod_{j} \theta_j^{a_j-1} \tag{13.T15}$$

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.

$$\mathbb{E}[\theta_j \mid N] = \frac{B(a + e_j)}{B(a)} = \frac{a_j}{A} = \frac{N_j + \alpha}{\sum_r N_r + d\alpha} \tag{13.T16}$$

$$\Gamma(t+1) = t\Gamma(t) \tag{13.T17}$$

$$\Gamma(t) = \int_0^\infty u^{t-1} e^{-u}\, du \tag{13.T18}$$

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