Machine Learning & Signals Learning

Dima Bykhovsky

August 24, 2026

This is an on-going project that is frequently updated. The author assumes no responsibility for errors or omissions.

This work is licensed under a Creative Commons "Attribution-NonCommercial-ShareAlike 4.0 International" license.

Contents

I. Machine Learning 5

1. Descriptive Statistics Basics 6
2. Uni-Variate Linear Least-Squares 23
3. Linear Least-Squares 35
4. Model Characterization 45
5. Kernels 70
6. Regression Losses and Metrics 78
7. Learning Systems (Mid-Summary) 89
8. Logistic Regression 97
9. Feature Engineering and Selection 110
10. k-Nearest Neighbors (k-NN) 120
11. Dimensionality Reduction 131
12. Classification Performance Metrics 141
13. Sanity Checks for Cross-Validation of Classifiers 157
14. Classifier Comparison 168
15. Classifier Fusion 202
16. Vanilla NN 219
17. From Machine Learning to Deep Learning 229
18. Computer Vision Metrics 236
19. Visual Representation Learning and Diagnostics 265

II. Linear Least Squares in Signals and Systems Modeling 282

20. Sinusoidal Signal Analysis 284

21.ARMA Model 316

22.ARX 348

23.Statistical Inference 363

24.Regression Metrics 380

III. Signal Classification 387

25.Signal Feature-Extraction Pipeline 388

26.Signal Classifiers 399

27.Exploratory Analysis of Univariate Signal Samples 409

28.Multivariate Signal Classification 418

IV. Appendix 435

A. Notation 436

B. Essential formulas 439

C. Plotting 447

References 503

Part I.

Machine Learning

1. Descriptive Statistics Basics

Goal: Describe the concise characteristics of a data.

Contents

1.1. Sample and Population Variance . . . **6**
1.1.1. Definition . . . 7
1.1.2. Repeated Experiments Characterization . . . 7
1.1.3. Bias . . . 11
1.2. Mode, Skewness and Median . . . **11**
1.2.1. Median . . . 11
1.2.2. Mode . . . 11
1.2.3. Skewness . . . 12
1.3. Boxplot . . . **12**
1.3.1. Prelinaries . . . 12
1.3.2. Plot . . . 13
1.4. Histogram . . . **14**
1.4.1. Discrete Values . . . 14
1.4.2. Continuous Values . . . 15
1.4.3. PDF Approximation of Continuous Random Variable (*) . . . 16
1.5. Violin Plot . . . **18**
1.6. Monte-Carlo Simulation . . . **18**
1.6.1. Number of Experiments . . . 20
1.6.2. Reproducibility . . . 22

1.1. Sample and Population Variance

Preliminaries

We assume a uni-variate random experiment that is described by a real-valued random variable X with

$$\mathbb{E}[X] = \mu$$
$$\mathrm{Var}[X] = \sigma^2.$$

Let

$$\mathbf{x} = \{x_1, \ldots, x_n\} \tag{1.1}$$

be n observations of X, where n may be fixed in advance or chosen arbitrarily.

1.1.1. Definition

Sample mean Given observations $x_1, \ldots, x_n$, the *sample mean* $\bar{\mathbf{x}}$ is given by

$$\bar{\mathbf{x}} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{1.2}$$

Sample variance *Sample variance* is given by

$$s^2_{\text{unbiased}} = \frac{1}{n-1} \sum_{i=1}^{n} \Big(x_i - \bar{x} \Big)^2 \tag{1.3}$$

This formulas is unbiased.

Population variance The more intuitive (biased) *population variance* formula is

$$s^2_{\text{biased}} = \frac{1}{n} \sum_{i=1}^{n} \Big(x_i - \bar{x} \Big)^2 \tag{1.4}$$

Standard Deviation The sample standard deviation (std) is simply the square-root of the (unbiased or biased) sample variance, s.

Table 1.1.: The difference between variance and std.

Aspect	Variance (s^2)	Std. Dev. (s)
Units	(original unit)^2	original unit
Interpretation	"Mean squared deviation"	"Average deviation from the mean"
Ease of communication	Abstract (squared units)	Concrete ($\pm$5 kg)

1.1.2. Repeated Experiments Characterization

Goal: Establish relations between theoretical and evaluated quantities, $\mu \overset{?}{\leftarrow} \bar{x}, \mu \overset{?}{\leftarrow} s^2_{biased}$ and $\mu \overset{?}{\leftarrow} s^2_{unbiased}$.

The *repeated* experiments approach is based on

- k experiments
  - In each experiment n samples $\mathbf{x}_1, \ldots, \mathbf{x}_k$ are drawn from the known random variable X.

Mean

The average of repeated experiments values is itself approximately μ, so on average the sample mean

$$\bar{\mu}_k = \frac{1}{k} \sum_{j=1}^{k} \bar{\mathbf{x}}_j \tag{1.5}$$

recovers the true mean

$$\lim_{k \to \infty} \bar{\mu}_k \to \mu \tag{1.6}$$

This principle is illustrated in Fig. 1.1.

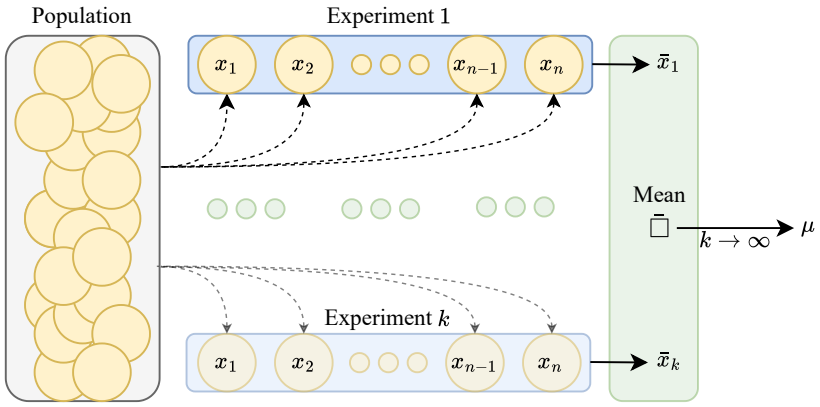


Figure 1.1.: Visualization of convergence of k repeated experiments to the true mean.

Fluctuations of $\bar{\mu}_k$:

- As you gather more observations (higher n), $\bar{\mathbf{x}}$ tends to stabilize: it fluctuates less, and its empirical distribution concentrates around the true center μ.
- Moreover, for large n, the variability of $\bar{\mathbf{x}}$ scales like $\sigma/\sqrt{n}$ (or $\mathrm{Var}[\bar{x}] \approx \frac{\sigma^2}{n}$), meaning the more data we collect, the tighter our estimate of the process's center becomes.

Variance

In the case of biased variance, the mean of the $s_{n_1}, \ldots, s_{n_k}$ values

$$\bar{s}_k^2 = \frac{1}{k} \sum_{j=1}^{k} s_{n_j} \tag{1.7}$$

systematically underestimates the true population variance σ^2,

$$\lim_{k \to \infty} \bar{s}_{k,\text{biased}}^2 \to \frac{n-1}{n} \sigma^2 \tag{1.8}$$

This inherent difference is termed *bias*. However, for the unbiased variance,

$$\lim_{k \to \infty} \bar{s}_{k,\text{unbiased}}^2 \to \sigma^2 \tag{1.9}$$

This principle is illustrated in Fig. 1.2.

Note, the difference between s_{biased} and s_{unbiased} goes smaller for high n.

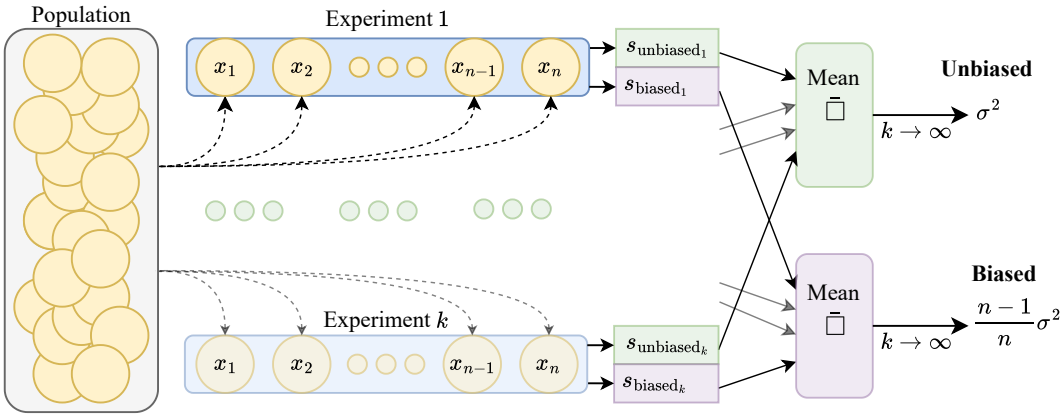


Figure 1.2.: Visualization of convergence of k repeated experiments to the biased and unbiased variances.

Standard Error

The fluctuation of $\bar{x}$ noted above has a name and a closed form. The *standard error* is the standard deviation of an *estimate*, as opposed to the standard deviation of the data the estimate was computed from.

For the sample mean of n uncorrelated samples of variance σ^2, the variances of the n terms add, while the $1/n$ in front of the sum is squared,

$$\operatorname{Var}[\bar{x}] = \frac{1}{n^2}\sum_{i=1}^{n}\operatorname{Var}[x_i] = \frac{n\sigma^2}{n^2} = \frac{\sigma^2}{n}, \tag{1.10}$$

so that

$$\operatorname{SE}(\bar{x}) = \sqrt{\operatorname{Var}[\bar{x}]} = \frac{\sigma}{\sqrt{n}}. \tag{1.11}$$

The square root appears because it is the *variances* that add, not the amplitudes.

Eq. (1.11) is read as a budget: the precision of the mean improves only with the square root of the sample size, so halving the standard error costs four times the data. The population σ is rarely known, and in practice it is replaced by the sample standard deviation s_x,

$$\widehat{\operatorname{SE}}(\bar{x}) = \frac{s_x}{\sqrt{n}}. \tag{1.12}$$

σ or SE

The two answer different questions. σ is the spread of the *data*, and it does not change as more data is collected. The standard error is the spread of the *estimate*, and it shrinks as $1/\sqrt{n}$. Reporting σ where $\operatorname{SE}(\bar{x})$ is meant overstates the uncertainty of the mean, and the reverse hides the variability of the population.

Uncorrelated samples

Eq. (1.11) assumes the n samples are uncorrelated. Correlated samples repeat information the earlier ones already carried, so they count for less than their number suggests: the true standard error is then larger than Eq. (1.12) reports, and any accuracy claim built on it is optimistic.

MSE

The reason that biased estimator is still useful, is the when the empirical mean-squared error (MSE) is of interest. MSE is defined by

$$\overline{\mathrm{MSE}}_{\mathrm{unbiased}} = \frac{1}{k}\sum_{j=1}^{k}\left(s_{\mathrm{unbiased}_j} - \sigma^2\right)^2 \to \frac{2}{n-1}\,\sigma^4 \tag{1.13}$$

$$\overline{\mathrm{MSE}}_{\mathrm{biased}} = \frac{1}{k}\sum_{j=1}^{k}\left(s_{\mathrm{biased}_j} - \sigma^2\right)^2 \to \frac{2n-1}{n^2}\,\sigma^4 \tag{1.14}$$

and

$$\overline{\mathrm{MSE}}_{n-1} \geq \overline{\mathrm{MSE}}_{n} \tag{1.15}$$

Note, that MSE error of the biased formula is slightly lower than unbiased one.

Example 1.1 (Unbiasedness and variance of the sample mean, and biased vs. unbiased variance estimators): Sample mean is unbiased and its variance decays as σ^2/n. The usual sample-variance estimators can be biased or unbiased. We illustrate all three properties by simulation:

1. **Parameters.**
   - True distribution: $X \sim \mathcal{N}(0,1)$, so $\mu = 0$, $\sigma^2 = 1$.
   - Sample sizes: $n \in \{5, 20, 100\}$.
   - Number of replicates: $k = 5000$.
2. **Results.** Results in Table 1.2 agree closely with the theoretical values. Read as standard errors, Eq. (1.11), the three empirical variances of $\bar{x}$ are 0.445, 0.221 and 0.100, against $\sigma/\sqrt{n} = 0.447$, 0.224 and 0.100. Quadrupling n from 5 to 20 halves the standard error, exactly as Eq. (1.11) requires.

Table 1.2.: Simulation results: sample mean, variance estimates, and their MSEs across different sample sizes.

n	$\bar{\mu}_n$	Empirical $\mathrm{Var}[\bar{x}]$	Theoretical $\mathrm{Var}[\bar{x}] = 1/n$	$\bar{s}^2_{\mathrm{biased}}$	$\bar{s}^2_{\mathrm{unbiased}}$	$\overline{MSE}_{\mathrm{biased}}$	$\overline{MSE}_{\mathrm{unbiased}}$
5	0.001	0.198	0.200	0.79	0.99	0.36	0.50
20	−0.000	0.049	0.050	0.95	1.00	0.10	0.11
100	0.000	0.010	0.010	0.99	1.01	0.02	0.02

1.1.3. Bias

The *bias* of a sample statistic is the systematic difference between its long-run average over repeated experiments and the corresponding true value of the random variable. A statistic is called *unbiased* when this difference is zero, and *biased* otherwise.

For example, for the sample variance,

$$\text{bias}(s^2) = \lim_{k\to\infty} \bar{s}_k^2 - \sigma^2. \tag{1.16}$$

Therefore, s^2_{unbiased} is unbiased while s^2_{biased} has bias $-\sigma^2/n$.

The application of biased and unbiased estimators:

- Biased: in ML tasks (e.g., loss function), optimal (maximum-likelihood estimation) for Gaussian distribution
- Unbiased: statistics.

Code implementation default varies:

- **Python** uses biased expression, e.g. `numpy.std`.
- **Matlab** uses **un**biased expression, e.g. `std`.

1.2. Mode, Skewness and Median

1.2.1. Median

Median is a value or quantity lying at the midpoint of a frequency distribution of observed values or quantities, such that there is an equal probability of falling above or below it.

Equivalently, for a sample $x_1, \ldots, x_n$ with order statistics $x_{(1)} \le \cdots \le x_{(n)}$, the *sample median* is

$$\text{median}(x) = \begin{cases} x_{(\frac{n+1}{2})}, & n \text{ odd}, \\ \dfrac{x_{(\frac{n}{2})} + x_{(\frac{n}{2}+1)}}{2}, & n \text{ even}. \end{cases}$$

1.2.2. Mode

For a finite sample $x_1, \ldots, x_n$, the *sample mode* is any value(s) that occur(s) most often among the observations.

Multimodality: A distribution is called *multimodal* if it has more than one local maximum (mode) in its probability density function or histogram. Common cases include:

- Unimodal: a single peak (e.g., normal distribution).
- Bimodal: two distinct peaks, often indicating that the data is a mixture of two subpopulations.
- Multimodal: three or more peaks.

Multimodality suggests that the data may come from a mixture of different underlying processes or groups.

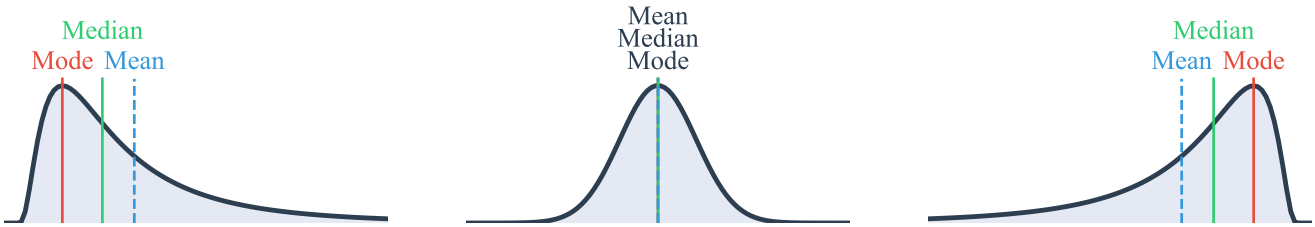
1.2.3. Skewness

Skewness: *Skewness* measures the asymmetry of a distribution about its mean. The sample skewness is defined as

$$g_1 = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i - \bar{x}}{s_n}\right)^3.$$

- $g_1 = 0$: the distribution is symmetric (e.g., normal distribution).
- $g_1 > 0$: the distribution is *right-skewed* (positively skewed) — the right tail is longer, and the mean is typically greater than the median.
- $g_1 < 0$: the distribution is *left-skewed* (negatively skewed) — the left tail is longer, and the mean is typically less than the median.

An illustration of the relationship between mean, median, and mode under different skewness is shown in Fig. 1.3.



(a) Positive skew (b) Symmetrical (c) Negative skew

Figure 1.3.: Relationship between mean, median, and mode for (a) positively skewed, (b) symmetrical, and (c) negatively skewed distributions.

1.3. Boxplot

Goal: Compact visualization of the distribution's location, spread, and potential outliers.

1.3.1. Prelinaries

Quartiles: Given the order statistics $x_{(1)} \leq \cdots \leq x_{(n)}$, the three *quartiles* divide the sorted data into four equal parts:

- Q_1 (first quartile, 25th percentile) — the median of the lower half of the data.
- Q_2 (second quartile, 50th percentile) — the *median* of the entire data set.
- Q_3 (third quartile, 75th percentile) — the median of the upper half of the data.

Equivalently, approximately 25% of the observations fall below Q_1, and another 25% up to Q_3.

Interquartile range: The *interquartile range* (IQR) measures the spread of the central 50% of the data,

$$\text{IQR} = Q_3 - Q_1. \tag{1.17}$$

Outlier: An *outlier* is any observation that falls outside the interval

$$\Big[Q_1 - 1.5\,\mathrm{IQR},\ \ Q_3 + 1.5\,\mathrm{IQR}\Big].$$

Such points are considered unusually far from the bulk of the data and may indicate measurement errors, rare events, or heavy-tailed behavior.

The coefficient 1.5 was proposed empirically by statistician John Tukey in 70th.

1.3.2. Plot

A *boxplot* (box-and-whisker plot) summarizes a data set using five statistics:

1. Minimum non-outlier value.
2. First quartile (Q_1).
3. Median (Q_2).
4. Third quartile (Q_3).
5. Maximum non-outlier value.

The boxplot is drawn as follows:

- A **box** spans from Q_1 to Q_3, with a line at the median Q_2.
- **Whiskers** extend from the box to the most extreme data points that lie within

$$\Big[Q_1 - 1.5\,\mathrm{IQR},\ \ Q_3 + 1.5\,\mathrm{IQR}\Big].$$

- Any observation outside the whisker range is marked individually as a potential **outlier**.

An illustration of a boxplot with its components is shown in Fig. 1.4.

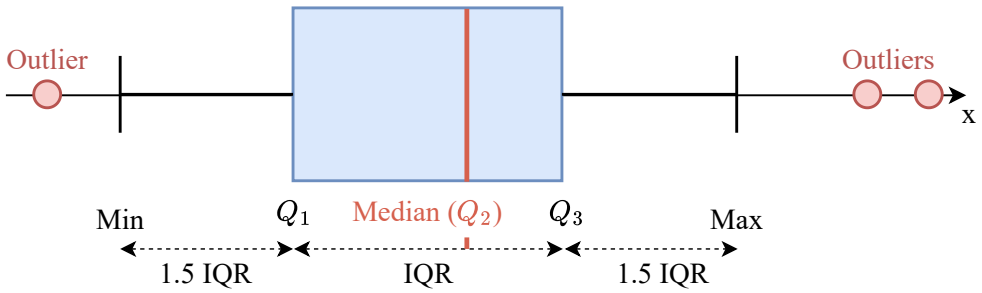


Figure 1.4.: Anatomy of a boxplot: the box spans the IQR from Q_1 to Q_3, the median Q_2 is marked inside, whiskers extend to the most extreme non-outlier observations, and individual outliers are shown beyond.

Interpretation

- The box width (IQR) reflects variability — a wide box indicates high spread.
- The median line position within the box reveals *skewness*: if the median is closer to Q_1, the distribution is right-skewed; if closer to Q_3, left-skewed.

- Whisker lengths indicate the range of typical observations.
- Individual points beyond the whiskers flag potential outliers that may warrant further investigation.

Unlike histograms, boxplots do not show the shape of the distribution (e.g., multimodality). They are most useful for comparing distributions across groups side by side.

1.4. Histogram

Goal: Visualization of the experimental data.

1.4.1. Discrete Values

`count` type

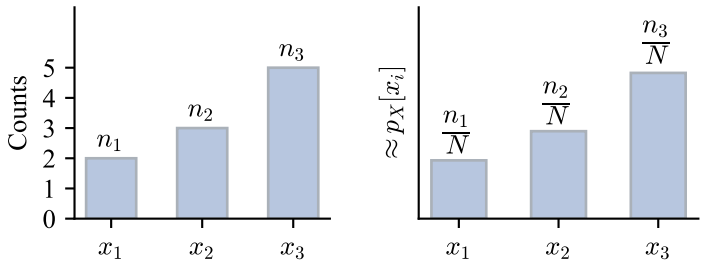
Goal: Show how many times each discrete outcome occurs.

Consider an experiment with:

- k possible distinct outcomes $x_1, x_2, \ldots, x_k$, where k is relatively small.
- A total of N trials.
- Recorded results:
  - n_1 occurrences of x_1,
  - n_2 occurrences of x_2,
  - ... and so on,

  with $\sum_i n_i = N$.

The highest bar in a count histogram identifies the **mode** of the sample. A graphical representation of the outcomes is shown in Fig. 1.5(a).



(a) `count` (b) `probability`

Figure 1.5.: Example of histograms: (a) `count`, (b) `probability`.

`probability` type

Goal: Show the proportion of each discrete outcome, providing an empirical estimate of the PDF.

Approximation to the `PDF`: The probability of a particular outcome is approximated by the ratio of its count to the total number of trials,

$$p_X[x_i] \approx \frac{n_i}{N}, \qquad i = 1, \ldots, k. \tag{1.18}$$

Naturally, the approximation improves as $N \to \infty$.

A graphical example of this histogram type is shown in Fig. 1.5(b).

1.4.2. Continuous Values

When the number of possible outcomes, k, is large (on the order of hundreds or more, or continuous values) two main difficulties arise:

- Presenting the results in a compact, readable form.
- Some outcome categories contain very few observations because their probabilities are small.

A practical way to display the data is:

1. Record the extreme values, $x_{\max}$ and $x_{\min}$.
2. Partition the interval $\left[x_{\min}, x_{\max}\right]$ into k equal-width bins of size Δx.
3. Mark each bin by its midpoint

$$\tilde{x}_i = x_{\min} + \left(i - \frac{1}{2}\right)\Delta x, \qquad i = 1, \ldots, k, \tag{1.19}$$

4. Let n_1 be the count in the first bin, n_2 the count in the second, and so on.
5. Use the pairs the pairs $(\tilde{x}_i, n_i)$ for count or probability type histograms.

An example of a `count` histogram for a large data set is shown in Fig. 1.6.

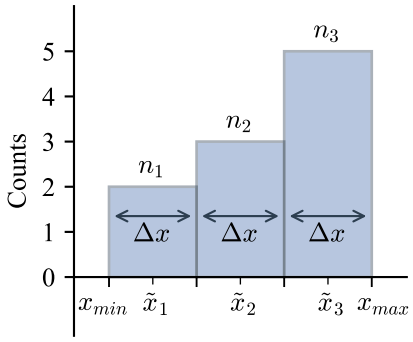


Figure 1.6.: A `count` histogram for a large number of experimental outcomes.

In this representation, the **median** is the value on the x-axis that divides the total area of the histogram into two equal parts. The **variance** s^2 and **standard deviation** s are reflected in the *width* or spread of the histogram: a broad histogram indicates high variability, while a narrow, sharp peak indicates that the data is tightly clustered around the mean.

add illustration/example

1.4.3. PDF Approximation of Continuous Random Variable (*)

Goal: Approximate the PDF of a continuous random variable from experimental data.

In addition to the binning method described above, there is a third histogram type that directly approximates the PDF of a continuous random variable.

Approximation to the PDF of a continuous random variable via histogram:

$$f_X(x_i) \approx \frac{n_i}{N} \cdot \frac{1}{\Delta x}, \quad i = 1, \ldots, k \tag{1.20}$$

Note the normalization factor $1/\Delta x$, which distinguishes this from the discrete case in (1.18).

A graphical example of this histogram type is shown in Fig. 1.7.

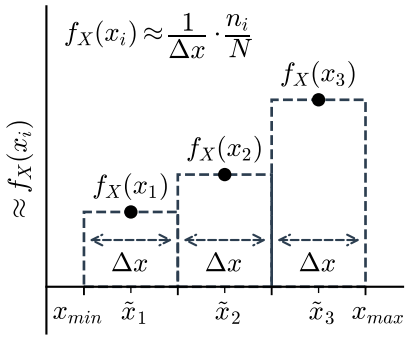


Figure 1.7.: A `pdf` histogram for a large number of experimental outcomes.

Derivation

Based on the principle

$$\Pr(a < X \leq b) = \int_a^b f_X(x)\, dx, \tag{1.21}$$

the probability of falling within a bin of width Δx centered at x_0 can be approximated as

$$\Pr\left(x_0 - \tfrac{\Delta x}{2} < X \leq x_0 + \tfrac{\Delta x}{2}\right) = \int_{\Delta x} f_X(x)\, dx \approx f_X(x_0)\, \Delta x. \tag{1.22}$$

An illustration of this principle is shown in Fig. 1.8. Rearranging,

$$f_X(x_0) \approx \Pr\left(x_0 - \tfrac{\Delta x}{2} < X \leq x_0 + \tfrac{\Delta x}{2}\right) \cdot \frac{1}{\Delta x}. \tag{1.23}$$

The probability of falling in bin i (centered at $\tilde{x}_i$) is approximated by the relative frequency,

$$\Pr\left(\tilde{x}_i - \tfrac{\Delta x}{2} < X \leq \tilde{x}_i + \tfrac{\Delta x}{2}\right) \approx \frac{n_i}{N}, \quad i = 1, \ldots, k. \tag{1.24}$$

Substituting (1.24) into (1.23) yields the PDF approximation formula in (1.20).

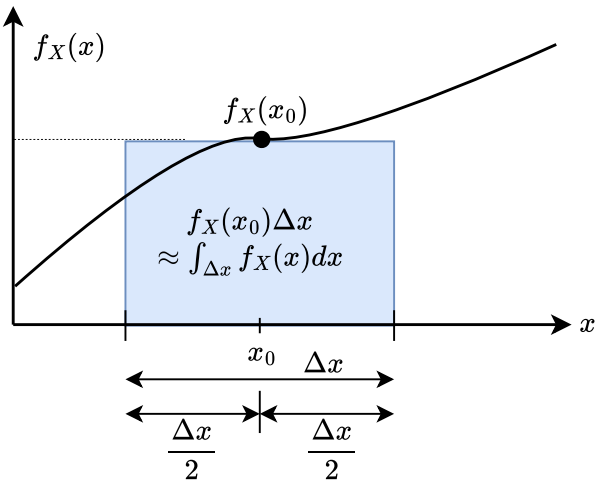


Figure 1.8.: Illustration of the histogram principle: the shaded area $f_X(x_0)\,\Delta x$ approximates the probability of falling within the bin.

Example 1.2 (Computing a PDF histogram): Given the experimental outcomes

$$[16,\ 98,\ 96,\ 49,\ 81,\ 14,\ 43,\ 92,\ 80,\ 96]\,,$$

display the PDF using a histogram with $k = 3$ bins.

Solution:

$$\begin{aligned} N &= 10 \\ x_{\min} &= 14, \quad x_{\max} = 98 \\ x_{\max} &= x_{\min} + k\,\Delta x = 98 \quad \Rightarrow \Delta x = \frac{x_{\max} - x_{\min}}{k} = \frac{84}{3} = 28 \end{aligned}$$

Counts per bin:

$$\begin{aligned} n_1 = 2, &\qquad \leftarrow \{14, 16\} \in [x_{\min},\ x_{\min} + \Delta x] &&= [14, 42] \\ n_2 = 2, &\qquad \leftarrow \{43, 49\} \in (x_{\min} + \Delta x,\ x_{\min} + 2\Delta x] &&= (42, 70] \\ n_3 = 6, &\quad \leftarrow \{80, 81, 92, 96, 96, 98\} \in (x_{\max} - \Delta x,\ x_{\max}] &&= (70, 98] \end{aligned}$$

where $n_3 = 10 - n_1 - n_2$. Bin midpoints:

$$\begin{aligned} \tilde{x}_1 &= x_{\min} + \tfrac{\Delta x}{2} = 28 \\ \tilde{x}_2 &= x_{\min} + \Delta x + \tfrac{\Delta x}{2} = 56 \\ \tilde{x}_3 &= x_{\max} - \tfrac{\Delta x}{2} = 84 \end{aligned}$$

The PDF approximation is therefore

$$f_X(\tilde{x}_i) \approx \frac{n_i}{N \cdot \Delta x} = n_i \cdot \frac{1}{10 \cdot 28}.$$

Summary

Three ways to display experimental results as a histogram:

- `count`: plot n_i — the raw bin counts.
- `probability`: plot n_i/N — the relative frequency per bin.
- `pdf` (or `density`): plot $\dfrac{n_i}{N\,\Delta x}$ — the estimated probability density.

1.5. Violin Plot

Goal: Visualize both the summary statistics (boxplot) and the shape of the data distribution (histogram).

A *violin plot* extends the boxplot by adding a (smoothed[1]) histogram of the data on each side. A median line is typically drawn inside.

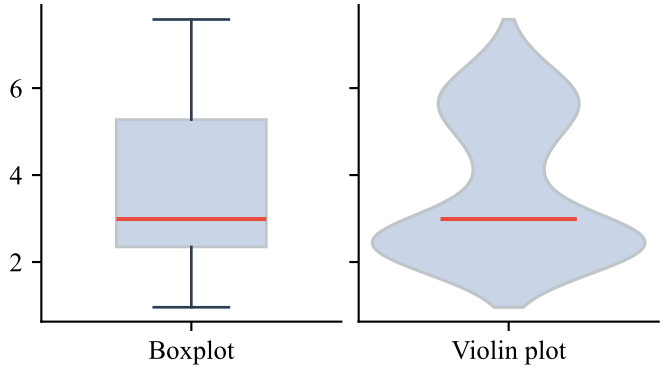


Figure 1.9.: Comparison of a boxplot and a violin plot for the same bimodal data: the boxplot hides the two-group structure, while the violin plot reveals it through its shape.

Fig. 1.9 compares the two representations for the same bimodal data set: the boxplot gives no indication of two groups, whereas the violin plot clearly shows two bulges.

1.6. Monte-Carlo Simulation

Goal: Evaluate a statistical quantity that has no convenient closed form by averaging over many synthetic random realizations.

The repeated experiments of Sec. 1.1 are an instance of a general numerical method, known as *Monte-Carlo simulation*. Instead of deriving the behavior of a statistic analytically, the random experiment is reproduced k times on a computer, and the resulting k values are summarized by the very same descriptive tools of this chapter.

Formally, let g be any statistic computed from a sample of n observations, e.g. $\bar{\mathbf{x}}$, s^2_{biased} or the MSE. Its expected value is approximated by the average over k independent repetitions,

$$\mathbb{E}[g] \approx \bar{g}_k = \frac{1}{k}\sum_{j=1}^{k} g_j, \tag{1.25}$$

[1]This smoothing is termed kernel density estimation (KDE). It is a special case of kernel smoothing in Sec. 5.5. The discussion of this technique is beyond the scope of this chapter.

where the law of large numbers guarantees $\bar{g}_k \to \mathbb{E}[g]$ as $k \to \infty$. The spread of g is estimated from the very same k values,

$$\mathrm{Var}[g] \approx s_g^2 = \frac{1}{k}\sum_{j=1}^{k}\left(g_j - \bar{g}_k\right)^2, \qquad s_g = \sqrt{s_g^2}, \tag{1.26}$$

and $\bar{g}_k - \theta$ estimates the bias with respect to a known true value θ (Sec. 1.1.3).

The value of s_g is set by the sample size n and by the distribution of X: it reports how much the statistic varies from one experiment to the next, and no number of repetitions reduces it. Making the statistic itself less variable requires a larger n. The number of repetitions k controls something different, namely how accurately $\bar{g}_k$ and s_g estimate the quantities they stand for.

The strength of the method is that (1.25) requires no derivation: it holds for any statistic, any distribution of X, and any sample size n, including all the cases where an analytical expression is unavailable or valid only for large n.

Monte-Carlo recipe

1. Fix the data-generating model and the *true* parameter values, e.g. $X \sim \mathcal{N}(\mu, \sigma^2)$ with the selected μ and σ^2.
2. Draw one realization of n observations $x_1, \ldots, x_n$.
3. Compute the statistic of interest g_j from this realization.
4. Repeat items 2–3 for $j = 1, \ldots, k$, storing all the k values.
5. Summarize the stored values: their mean (1.25) estimates $\mathbb{E}[g]$ and the bias (Sec. 1.1.3), their std s_g (1.26) estimates the spread of g, and their histogram (Sec. 1.4), boxplot (Sec. 1.3) or violin plot (Sec. 1.5) reveals the whole distribution of g.

The whole recipe is summarized in Fig. 1.10: the model feeds k independent experiments of n observations each, every experiment is reduced to a single value g_j, and only the collection of the k values is interpreted.

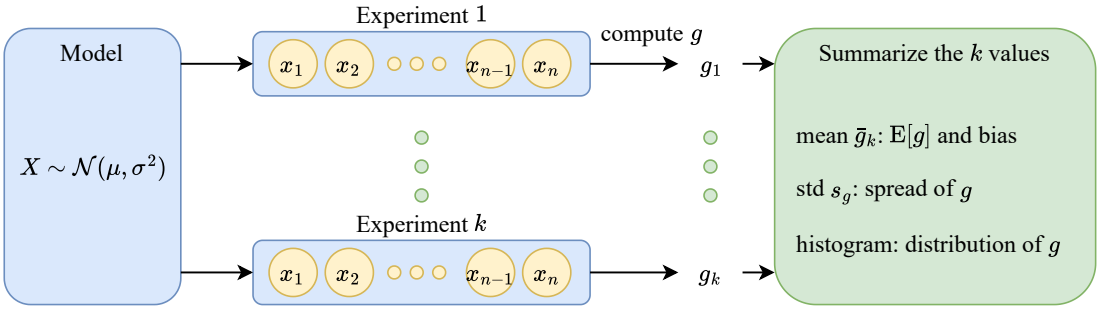


Figure 1.10.: Monte-Carlo recipe. Each of the k experiments draws n observations from the same model and is reduced to one value of the statistic g. The k stored values are then summarized by the descriptive tools of this chapter.

Table 1.2 is precisely the output of this recipe, applied to the sample mean, the two variance estimators and their MSEs, with $n \in \{5, 20, 100\}$ and $k = 5000$. Each of its entries condenses k stored values into a single number; Fig. 1.11 shows the values themselves behind one entry.

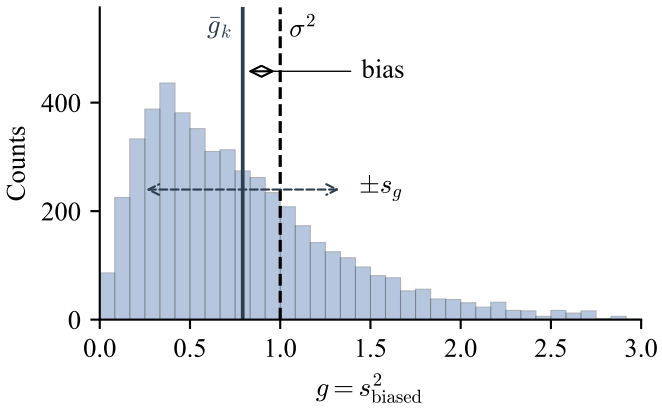


Figure 1.11.: The $k = 5000$ stored values behind the $\bar{s}^2_{\text{biased}}$ entry of the $n = 5$ row of Table 1.2. The statistic is $g = s^2_{\text{biased}}$, simulated from $n = 5$ observations of $X \sim \mathcal{N}(0, 1)$. Their mean $\bar{g}_k = 0.79$ falls short of the true $\sigma^2 = 1$ by the bias $-\sigma^2/n = -0.2$, and their std s_g (1.26) measures how much g moves from one experiment to the next. The histogram adds what these two numbers cannot show: the distribution of g is strongly skewed.

1.6.1. Number of Experiments

The simulated result is itself a random quantity, so it carries an error of its own, and this error is controlled by k alone.

Simulated mean The std of a simulated mean is

$$\sqrt{\text{Var}[\bar{g}_k]} = \frac{\sqrt{\text{Var}[g]}}{\sqrt{k}}, \tag{1.27}$$

where $\text{Var}[g]$ is the spread (1.26) of the individual g_j: the accuracy of a simulated mean is the spread of the statistic divided by $\sqrt{k}$.

Simulated std and variance Similarly, the relative error of a simulated std s_g is

$$\frac{\sqrt{\text{Var}\left[s_g\right]}}{s_g} \approx \frac{1}{\sqrt{2k}}, \tag{1.28}$$

while the relative error of a simulated variance s_g^2 is twice as large,

$$\frac{\sqrt{\text{Var}\left[s_g^2\right]}}{s_g^2} \approx \sqrt{\frac{2}{k}} = \frac{2}{\sqrt{2k}}. \tag{1.29}$$

The factor of two follows from squaring: a variance is the square of a std, and squaring doubles the relative fluctuation. A simulated variance is therefore always twice as uncertain as the simulated std obtained from the very same k values, and the two should not be quoted with the same accuracy.

All three decay as $1/\sqrt{k}$, so halving the simulation error costs four times more experiments. Representative values are listed in Table 1.3 and illustrated in Fig. 1.12 for the simulated mean and in Fig. 1.13 for the simulated std.

Table 1.3.: Relative Monte-Carlo error of a simulated mean (1.27), of a simulated std (1.28) and of a simulated variance (1.29), as a function of the number of experiments k.

k	Mean, $1/\sqrt{k}$	Std, $1/\sqrt{2k}$	Variance, $\sqrt{2/k}$
10^2	10%	7.1%	14%
10^3	3.2%	2.2%	4.5%
10^4	1.0%	0.7%	1.4%
10^5	0.32%	0.22%	0.45%
10^6	0.10%	0.07%	0.14%

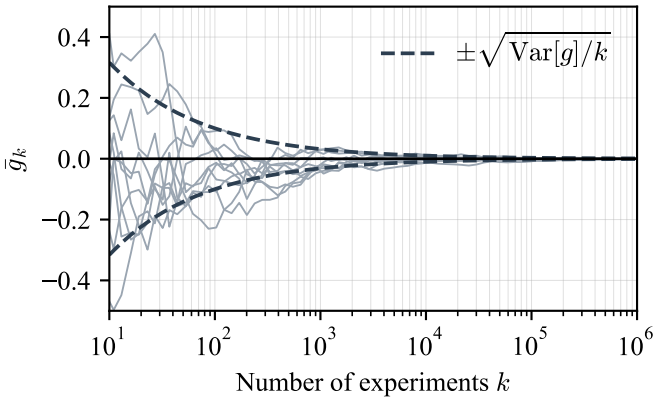


Figure 1.12.: Simulated mean $\bar{g}_k$ as a function of the number of experiments k. Gray: ten independent simulations. Dashed: the $\pm\sqrt{\mathrm{Var}[g]/k}$ envelope of (1.27). The traces stay within the envelope and tighten around the true value as k grows.

The consequence is that a simulated value is never exact, and the number of correct digits it carries is set by k. In Table 1.2, for instance, $k = 5000$ yields a variance error of about 2% (1.29), which is why the $n = 100$ row reads $\bar{s}^2_{\text{unbiased}} = 1.01$ rather than the exact 1.00.

Derivation (*) Where do the three formulas come from? Each one answers the same question: if the whole simulation is repeated with a new seed, how far does the reported number move?

Start with the mean. The stored values $g_1, \ldots, g_k$ are independent, so their variances add up, and dividing their sum by k divides the variance by k^2,

$$\mathrm{Var}[\bar{g}_k] = \frac{1}{k^2}\sum_{j=1}^{k}\mathrm{Var}\left[g_j\right] = \frac{\mathrm{Var}[g]}{k}.$$

The square root of this is (1.27). It is the familiar σ^2/n rule of the sample mean (Sec. 1.1), applied to k experiments instead of n observations. Only independence was used, so this rate is valid for any distribution of g. This is also the one rate given as an absolute value instead of a percentage: a percentage would divide by $\mathbb{E}[g]$, which is zero whenever the simulation checks a bias against zero.

The variance needs one known result. For Gaussian values, the sum of the squared deviations follows a chi-square distribution, whose variance is twice its number of degrees of freedom. With k values this gives

$$\mathrm{Var}\left[s_g^2\right] \approx \frac{2\sigma_g^4}{k},$$

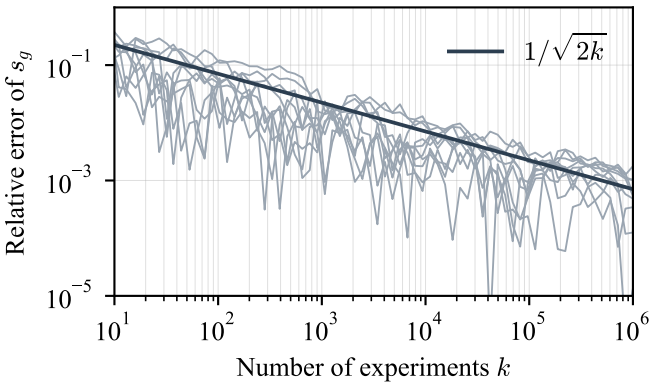


Figure 1.13.: Relative error of the std s_g, estimated from k Monte-Carlo values, as a function of k. Gray: ten independent simulations. Black: the $1/\sqrt{2k}$ prediction of (1.28). Individual simulations scatter around the prediction, and only the decay rate is guaranteed.

and dividing by $s_g^2 \approx \sigma_g^2$ turns it into the percentage (1.29).

The std needs no new assumption, only a square root. A small relative change of the variance shrinks to half of itself after the square root is taken,

$$s_g = \sigma_g\sqrt{1+\varepsilon} \approx \sigma_g\left(1+\frac{\varepsilon}{2}\right),$$

so a variance known to $\pm 10\%$ gives a std known to $\pm 5\%$. Halving $\sqrt{2/k}$ gives $1/\sqrt{2k}$, which is (1.28).

Note that only the middle step assumed a Gaussian. When g has heavier tails, rare large values inflate the squared deviations, and both the variance and the std are then less accurate than these formulas promise. The mean is unaffected. Whether the variance is normalized by k or by $k-1$ makes no visible difference at these accuracies.

1.6.2. Reproducibility

A computer draws *pseudo*-random numbers, produced by a deterministic algorithm from an initial value called the *seed*. Fixing the seed makes the whole simulation exactly repeatable, which is a requirement for a reported result, not an option.

- **Python**: `numpy.random.default_rng(seed)`.
- **Matlab**: `rng(seed)`.

- A single realization proves nothing. Any conclusion has to be drawn from the k collected values.
- Report k and store the seed next to every simulated number, so that its accuracy (1.28) can be judged.
- Never try select the seed that produces "better" result, i.e. "lucky shot". The seed is fixed before the experiment, not after it.

2. Uni-Variate Linear Least-Squares

The goal of this chapter is to define and discuss uni-variate linear least-squares (LLS or LS) for a linear model.

Note that LS is also termed *linear regression*. Uni-variate LS is also referred to as a *linear trend-line*.

Contents

2.1. Uni-variate Linear LS **23**
2.1.1. Definitions 23
2.1.2. Mean and Variance 25
2.1.3. Linear Model 25
2.2. Covariance & Correlation Coefficient **26**
2.2.1. Sample Covariance 27
2.2.2. Correlation Coefficient 27
2.2.3. Normal Equations with Statistical Terms 28
2.3. Metrics (R^2, MSE) **28**
2.3.1. Coefficient of Determination (R^2) 28
2.3.2. MSE of Linear LS 30
2.4. Normalized (z-score) Linear Regression (*) **30**
2.5. Spearman Rank Correlation **33**
2.6. Correlation is not Causation **34**

2.1. Uni-variate Linear LS

2.1.1. Definitions

Dataset: A random experiment produces a *dataset*[1] of M paired observations $\{x_k, y_k\}_{k=1}^{M}$.

ML Model: The assumed model underlying the dataset is

$$y = h(x) + \epsilon, \tag{2.1}$$

where $h(x)$ is a deterministic function and ϵ is zero-mean noise.

[1] Don't confuse with *database* that is something else.

The goal is to propose and apply the *model*,

$$\hat{y}_k = f(x_k) \tag{2.2}$$

that provide the most appropriate results with relation to y_k.

Challenge: there are many possible $f(\cdot)$ options.

Error: The model *error* is

$$e_k = y_k - \hat{y}_k \tag{2.3}$$

Loss

Goal: Find optimal model parameters.

Loss function: Loss (or cost) function $\mathcal{L}(y_k, \hat{y}_k)$ is some distance metric between y_k and $\hat{y}_k$. The optimal model parameters of the applied model are found by minimizing a loss function.

Common examples of loss functions include:

- the sum of squared errors (SSE),

$$\text{SSE} = \sum_{k=1}^{M} e_k^2, \tag{2.4}$$

- mean-square error (MSE) or (biased) error variance

$$\text{MSE} = \frac{1}{M} \sum_{k=1}^{M} e_k^2 = \frac{1}{M} \text{SSE} = s_e^2 \tag{2.5}$$

- and root-mean-square error (RMSE),

$$\text{RMSE} = \sqrt{\text{MSE}}. \tag{2.6}$$

Because constant multiplication factors and monotonic transforms do not affect the location of the minimum, all these loss functions share the same optimal parameters:

$$\begin{aligned} w_0, w_1 &= \arg\min_{w_0, w_1} \text{SSE} \\ &= \arg\min_{w_0, w_1} \text{MSE} \\ &= \arg\min_{w_0, w_1} \text{RMSE}. \end{aligned} \tag{2.7}$$

Metric: A quantifiable measure used to evaluate how well a model is performing.

For example, the of RMSE is significantly easier to interpret due to same values range and measurement units as an original dataset, than MSE or SSE.

It is easy to confuse metrics with loss functions, but they serve different purposes:

- *Loss* function is used by the **algorithm** during training to learn optimal model parameters.
- *Metric* is used by **humans** to evaluate the model's performance.

Summary

1. Select model $f(\cdot)$
2. Find model parameters by the minimization of the loss function, $\mathcal{L}$
3. Use metrics to evaluate model performance.

2.1.2. Mean and Variance

Goal: To provide an *interpretation* of the mean and variance within the context of LS.

A special case of the linear model is one of the form:

$$\hat{y} = w_0 \tag{2.8}$$

The corresponding MSE loss (2.5) function is

$$\mathcal{L}(w_0) = \frac{1}{M}\sum_{k=1}^{M}(y_k - w_0)^2, \tag{2.9}$$

with the related minimum at the mean of y_k values,

$$w_0 = \frac{1}{M}\sum_{k=1}^{M} y_k = \bar{y} \tag{2.10}$$

Proof. To find the minimum,

$$\frac{d}{dw_0}\mathcal{L}(w_0) = 2\frac{1}{M}\sum_{k=1}^{M}(y_k - w_0)(-1) = 0 \tag{2.11}$$

The corresponding MSE of the model (substituting (2.10) into (2.9)) is

$$\text{MSE} = \frac{1}{M}\sum_{k=1}^{M}(y_k - \bar{y})^2 = s_y^2 \tag{2.12}$$

which is the *sample variance* of the y_k values [2].

To summarize, the best predictor is the sample mean, $\hat{y} = \bar{y}$, with MSE $= s_y^2$.

2.1.3. Linear Model

The example of *linear model* is

$$\hat{y}_k = f(x_k; w_0, w_1) = w_0 + w_1 x_k, \tag{2.13}$$

where w_0 and w_1 are the model parameters.

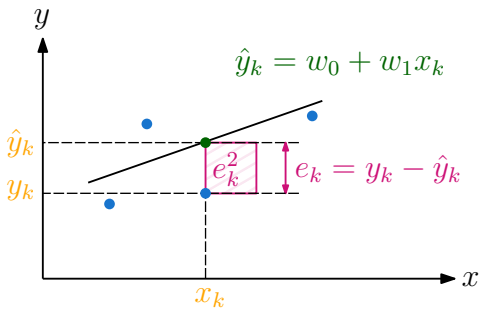


Figure 2.1.: Linear regression visualization. The goal is to minimize SSE, which represents the total area $\sum_k e_k^2$ of the rectangles.

The corresponding linear model error is

$$e_k = y_k - w_0 - w_1 x_k. \tag{2.14}$$

The minimization of the SSE loss function ((2.4) and Fig. 2.1),

$$\mathcal{L}(w_0, w_1) = \sum_{k=1}^{M} \Big(y_k - w_0 - w_1 x_k \Big)^2, \tag{2.15}$$

is performed by setting the partial derivatives of $\mathcal{L}$ to zero:

$$\begin{cases} \dfrac{\partial}{\partial w_0} \mathcal{L}(w_0, w_1) = 0 \\ \dfrac{\partial}{\partial w_1} \mathcal{L}(w_0, w_1) = 0 \end{cases} \tag{2.16}$$

which yields:

$$\begin{cases} 2 \displaystyle\sum_{k=1}^{M} (y_k - w_0 - w_1 x_k) \cdot (-1) = 0 \\ 2 \displaystyle\sum_{k=1}^{M} (y_k - w_0 - w_1 x_k) \cdot (-x_k) = 0. \end{cases} \tag{2.17}$$

Finally, with some basic algebra:

$$\begin{cases} w_0 M + w_1 \displaystyle\sum_{k=1}^{M} x_k = \sum_{k=1}^{M} y_k, \\ w_0 \displaystyle\sum_{k=1}^{M} x_k + w_1 \sum_{k=1}^{M} x_k^2 = \sum_{k=1}^{M} x_k y_k. \end{cases} \tag{2.18}$$

This system of equations is termed the **normal equations**.

2.2. Covariance & Correlation Coefficient

Goal: To rewrite the normal equations (2.18) using statistical terms.

[2]This variance expression is called *biased* and is used throughout this chapter (see Sec. 1.1)

2.2.1. Sample Covariance

Goal: Quantify linear relation between x_k and y_k.

The (biased) sample covariance between $x_1, \ldots, x_M$ and $y_1, \ldots, y_M$ is given by

$$s_{xy} = \frac{1}{M} \sum_{k=1}^{M} (x_k - \bar{x})\,(y_k - \bar{y}) \tag{2.19}$$

The sign and magnitude of the sample covariance s_{xy} indicate the direction and strength of the linear relationship between x and y:

- $s_{xy} > 0$: As x increases, y tends to increase (most products $(x_i - \bar{x})(y_i - \bar{y})$ are positive).
- $s_{xy} < 0$: As x increases, y tends to decrease.
- $s_{xy} \approx 0$: No linear association.

2.2.2. Correlation Coefficient

Goal: Normalized sample covariance.

The *sample Pearson correlation coefficient* between $\mathbf{x}$ and $\mathbf{y}$ is the normalized (dimensionless) covariance:

$$\begin{aligned} r_{xy} &= \frac{s_{xy}}{s_x\, s_y} \\ &= \frac{\displaystyle\sum_{k=1}^{M} (x_k - \bar{x})\,(y_k - \bar{y})}{\sqrt{\displaystyle\sum_{k=1}^{M} (x_k - \bar{x})^2}\,\sqrt{\displaystyle\sum_{k=1}^{M} (y_k - \bar{y})^2}} \end{aligned} \tag{2.20}$$

Interpretation

- Range: $-1 \leq r_{xy} \leq +1$.
- $r_{xy} > 0$: positive linear association (as x increases, y tends to increase).
- $r_{xy} < 0$: negative linear association (as x increases, y tends to decrease).
- $r_{xy} = 0$: no *linear* association (a nonlinear relationship may still exist).
- $|r_{xy}| = 1$: perfect linear relationship ($y_k = a + bx_k$ for all k).

Note that there is *no difference between biased and unbiased* definitions for r_{xy}, since the corresponding coefficients cancel out.

2.2.3. Normal Equations with Statistical Terms

It is numerically convenient to rewrite the normal equations (2.18) in terms of sample moments

$$\begin{aligned} w_1 &= \frac{s_{xy}}{s_x^2} = r_{xy}\,\frac{s_y}{s_x} \\ &= \frac{\sum_k (x_k - \bar{x})(y_k - \bar{y})}{\sum_k (x_k - \bar{x})^2}, \\ w_0 &= \bar{y} - w_1 \bar{x} \\ &= \bar{y} - \frac{s_{xy}}{s_x^2}\bar{x}, \end{aligned}$$

and the predictive form

$$\hat{y} = \bar{y} + w_1\Big(x - \bar{x}\Big). \tag{2.21}$$

Note that these expressions are independent of whether biased or unbiased definitions are used for s_{xy} and s_x, since the corresponding normalization factors cancel out.

Remarks.

- A valid solution requires $s_x \neq 0$, i.e. variability in $x_i \neq$ const.
- If both variables are centered ($\bar{x} = \bar{y} = 0$), then (2.21) reduces to

$$\begin{aligned} w_0 &= 0, \\ w_1 &= \frac{\sum_k x_k y_k}{\sum_k x_k^2} \end{aligned} \tag{2.22}$$

2.3. Metrics (R^2, MSE)

The performance of a model is quantified by a performance metric, which is not necessarily the same as the loss function.

2.3.1. Coefficient of Determination (R^2)

To emphasize the difference between loss and metrics, the following example of an LR metric is provided. The coefficient of determination, denoted R^2 (R-squared), is based on the relation:

$$\underbrace{\sum_{k=1}^{M}(y_k - \bar{y})^2}_{\text{SST}} = \underbrace{\sum_{k=1}^{M}(\hat{y}_k - \bar{y})^2}_{\text{SSR}} + \underbrace{\sum_{k=1}^{M} e_k^2}_{\text{SSE}}, \tag{2.23}$$

where:

- **SST**: Total sum of squares.
- **SSR**: Sum of squares due to regression.

- **SSE**: Sum of squared errors (or residual sum of squares).

The R^2 metric is defined as:

$$R^2 = \frac{\text{SSR}}{\text{SST}} = 1 - \frac{\text{SSE}}{\text{SST}}, \tag{2.24}$$

providing a unitless goodness-of-fit measure that shares an intuitive $[0, 1]$ range:

- $R^2 = 1$: Perfect fit.
- $R^2 = 0$: The model is no better than predicting the mean, $\hat{y}_i = \bar{y}$.
- $R^2 < 0$: The model performs worse than the mean (possible for machine learning models on test data).

Proof. To proof (2.23), observe that

$$y_k - \bar{y} \;=\; (\hat{y}_k - \bar{y}) + (y_k - \hat{y}_k) \;=\; (\hat{y}_k - \bar{y}) + e_k.$$

Hence,

$$\begin{aligned}\sum_{k=1}^{M}(y_k - \bar{y})^2 &= \sum_{k=1}^{M}\Big[(\hat{y}_k - \bar{y}) + e_k\Big]^2 \\ &= \sum_{k=1}^{M}(\hat{y}_k - \bar{y})^2 \;+\; \sum_{k=1}^{M} e_k^2 \;+\; 2\sum_{k=1}^{M}(\hat{y}_k - \bar{y})\, e_k.\end{aligned}$$

It remains to show the cross-term vanishes:

$$\sum_{k=1}^{M}(\hat{y}_k - \bar{y})\, e_k = \sum_{k=1}^{M}\hat{y}_k\, e_k \;-\; \bar{y}\sum_{k=1}^{M} e_k = 0 - 0 = 0,$$

since in LS, $\sum_{k=1}^{M} e_k = 0$ and $\sum_{k=1}^{M}\hat{y}_k\, e_k = 0$.

R^2 of Uni-variate Linear LS For the uni-variate LR case, it is the fraction of the sample variance of y explained by the linear fit,

$$R^2 = r_{xy}^2. \tag{2.25}$$

Proof. Starting with $\hat{y}_k = w_0 + w_1 x_k$:

$$\begin{aligned}SSR &= \sum_{k=1}^{M}(\hat{y}_k - \bar{y})^2 \\ &= \sum_{k=1}^{M}(w_0 + w_1 x_k - \bar{y})^2 \\ &= \sum_{k=1}^{M}(\bar{y} - w_1\bar{x} + w_1 x_k - \bar{y})^2 \\ &= w_1^2\sum_{k=1}^{M}(x_k - \bar{x})^2 \\ &= w_1^2 M s_x^2 = M\left(\frac{s_{xy}}{s_x^2}\right)^2 s_x^2 = M\frac{s_{xy}^2}{s_x^2}, \\ SST &= \sum_{k=1}^{M}(y_k - \bar{y})^2 = M s_y^2\end{aligned} \tag{2.26}$$

Thus, $SSR/SST = s_{xy}^2/(s_x^2 s_y^2) = r_{xy}^2$.

2.3.2. MSE of Linear LS

The general (biased[3]) MSE is given by

$$\mathrm{MSE} = s_y^2\Big(1 - r_{xy}^2\Big) = s_e^2 \tag{2.27}$$

Proof. Using (2.23):

$$\begin{aligned}\mathrm{MSE} &= \frac{1}{M}\mathrm{SSE} = \frac{1}{M}\left(\mathrm{SST} - \mathrm{SSR}\right)\\ &= \frac{\mathrm{SST}}{M}\left(1 - \frac{\mathrm{SSR}}{\mathrm{SST}}\right)\\ &= s_y^2\Big(1 - R^2\Big)\\ &= s_y^2\Big(1 - r_{xy}^2\Big)\end{aligned} \tag{2.28}$$

2.4. Normalized (z-score) Linear Regression (*)

The goal of this section is to provide an intuitive interpretation of the correlation coefficient.

Rescaling To make the slope directly interpretable, we rescale both variables to zero mean and unit variance:

$$z_i = \frac{x_i - \bar{x}}{s_x}, \qquad t_i = \frac{y_i - \bar{y}}{s_y}, \qquad i = 1, \ldots, M \tag{2.29}$$

The properties of z_i and t_i are

$$\begin{aligned}\bar{z} &= \frac{1}{M}\sum z_i = 0,\\ \bar{t} &= \frac{1}{M}\sum t_i = 0,\\ s_z &= s_t = 1.\end{aligned}$$

Proof. Using biased variance definitions [4]:

$$s_x^2 = \frac{1}{M}\sum_{k=1}^{M}(x_k - \bar{x})^2, \qquad s_y^2 = \frac{1}{M}\sum_{k=1}^{M}(y_k - \bar{y})^2. \tag{2.30}$$

[3] The unbiased estimator of s_e^2 requires $1/(M-2)$ factor instead $1/M$.
[4] For unbiased evaluation the factor $M-1$ is to be used.

For the means

$$\begin{aligned}\sum_{k=1}^{M} z_k &= \frac{1}{s_x}\sum_{k=1}^{M}(x_k - \bar{x}) \\ &= \frac{1}{s_x}\left(\sum_{k=1}^{M} x_k - M\bar{x}\right) \\ &= \frac{1}{s_x}(M\bar{x} - M\bar{x}) = 0\end{aligned} \tag{2.31}$$

hence $\bar{z} = 0$ and similarly $\bar{t} = 0$. For the variances:

$$s_z^2 = \frac{1}{M}\sum_{k=1}^{M} z_k^2 = \frac{1}{M}\sum_{k=1}^{M}\frac{(x_k - \bar{x})^2}{s_x^2} = \frac{1}{s_x^2}\left[\frac{1}{M}\sum_{k=1}^{M}(x_k - \bar{x})^2\right] = \frac{s_x^2}{s_x^2} = 1. \tag{2.32}$$

The proof for $s_t^2 = 1$ is analogous.

Normal Equations The linear model in normalized space is

$$\hat{t}_i = r_{xy}\, z_i. \tag{2.33}$$

where

$$r_{xy} = \frac{s_{xy}}{s_x\, s_y} \tag{2.34}$$

is the correlation coefficient.

Proof. Applying (2.22) to

$$\hat{t}_i = w_0^* + w_1^* z_i \tag{2.35}$$

yields

$$w_0^* = 0, \tag{2.36}$$

$$w_1^* = \frac{s_{zt}}{s_z^2} = \frac{1}{M}\sum_{i=1}^{M} z_i t_i = \frac{1}{M}\sum_{i=1}^{M} z_i t_i = \frac{1}{M}\sum_{i=1}^{M}\left(\frac{x_i - \bar{x}}{s_x}\right)\left(\frac{y_i - \bar{y}}{s_y}\right) = r_{xy} \tag{2.37}$$

Regression error as a function of r_{xy}

With the optimal normalized model $\hat{t} = r_{xy} z$ the residuals are

$$e_i = t_i - \hat{t}_i = t_i - r_{xy} z_i \tag{2.38}$$

with

$$\mathrm{MSE} = 1 - r_{xy}^2 = s_e^2. \tag{2.39}$$

Proof.

$$
\begin{aligned}
s_e^2 &= \frac{1}{M}\sum_{i=1}^{M} e_i^2 = \frac{1}{M}\sum_{i=1}^{M}\left(t_i - r_{xy} z_i\right)^2 \\
&= \frac{1}{M}\sum_{i=1}^{M} t_i^2 - 2r_{xy}\frac{1}{M}\sum_{i=1}^{M} t_i z_i + r_{xy}^2 \frac{1}{M}\sum_{i=1}^{M} z_i^2 \\
&= s_t^2 - 2r_{xy}\left(r_{xy}\right) + r_{xy}^2 s_z^2 \\
&= 1 - r_{xy}^2
\end{aligned}
\tag{2.40}
$$

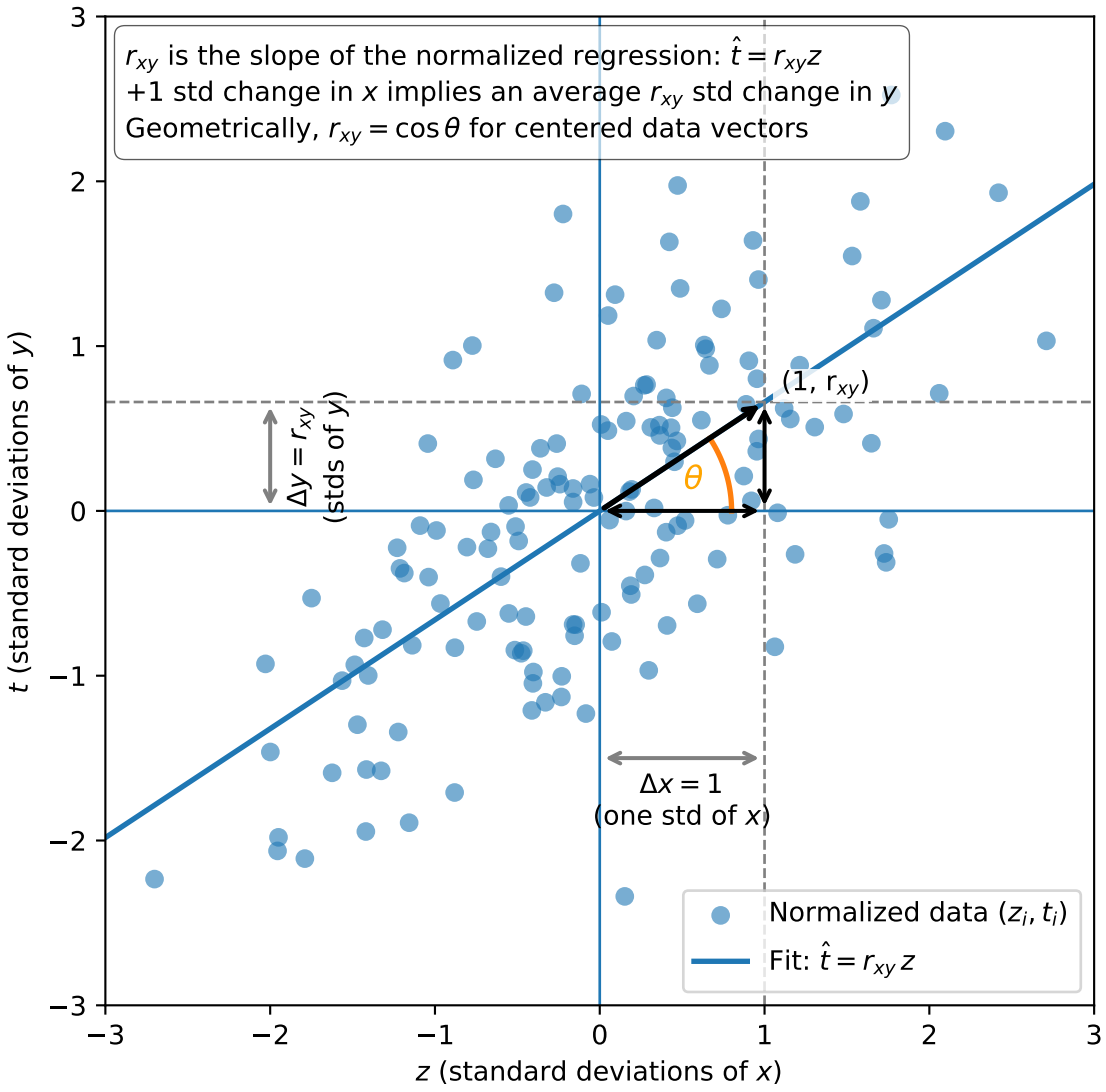


Figure 2.2.: Geometric interpretation of r_{xy} in a normalized LR.

Geometrical interpretation of r_{xy}

- Units of x- and y-axes are standard deviation from the origin since t_i and z_i are centered and standardized.
- In the normalized space, r_{xy} *is* the regression slope. A one-standard-deviation increase in x (z changes by 1) changes y by r_{xy} standard deviations on average ($\hat{t}$ changes by r_{xy}).

- Let's define two vectors, $\mathbf{z}$ and $\mathbf{t}$, both in $\mathbb{R}^M$. The angle between them is θ and $r_{xy} = \cos\theta$ (Fig. 2.2).
  Proof. The corresponding dot product is

$$\mathbf{z} \cdot \mathbf{t} = \|\mathbf{z}\|\|\mathbf{t}\| \cos(\theta) \tag{2.41}$$

  Using (2.37),

$$\mathbf{z} \cdot \mathbf{t} = \sum_{i=1}^{M} z_i t_i = M r_{xy}. \tag{2.42}$$

  Since $\|\mathbf{z}\| = \sqrt{\sum z_i^2} = \sqrt{M s_z^2} = \sqrt{M}$, we have

$$\|\mathbf{z}\|\|\mathbf{t}\| = \sqrt{M} \cdot \sqrt{M} = M \tag{2.43}$$

Consequences:

- Range: $-1 \leq r_{xy} \leq +1$.
- Special cases:
  - $r_{xy} = +1$ indicates perfect positive linear association, $\theta = 0°$.
  - $r_{xy} = -1$ indicates perfect negative linear association, $\theta = 180°$.
  - $r_{xy} = 0$ indicates no linear association, $\theta = 90°$.

2.5. Spearman Rank Correlation

Goal: Non-parametric alternative to Pearson correlation for monotonic (not necessarily linear) relationships.

The Pearson correlation coefficient r_{xy} measures *linear* association. When the relationship between x and y is monotonic but non-linear, or when the data contain outliers, the *Spearman rank correlation* provides a more robust measure.

Let $R(x_i)$ denote the rank of x_i among $\{x_1, \ldots, x_M\}$ (with average ranks for ties), and similarly $R(y_i)$. The Spearman correlation is the Pearson correlation computed on the ranks:

$$r_s = r_{R(x)\,R(y)} \tag{2.44}$$

- Range: $-1 \leq r_s \leq +1$.
- $|r_s| = 1$: perfect *monotonic* (not necessarily linear) relationship.
- Robust to outliers and non-linear monotonic relationships.
- For linear data, $r_s \approx r_{xy}$.

Example 2.1: Consider $M = 5$ observations with an exponential relationship $y \approx e^x$. The table compares a clean dataset (perfectly monotonic) with a noisy one (rank swap at $i = 2, 3$):

	Clean ($y_i = e^{x_i}$)					Noisy				
i	1	2	3	4	5	1	2	3	4	5
x_i	1	2	3	4	5	1	2	3	4	5
y_i	2.7	7.4	20.1	54.6	148.4	3.5	8.1	5.3	52.0	135.7
$R(x_i)$	1	2	3	4	5	1	2	3	4	5
$R(y_i)$	1	2	3	4	5	1	3	2	4	5

- **Clean**: $r_{xy} \approx 0.89$, $r_s = 1$. Pearson underestimates the association (the exponential curve is not a straight line), while Spearman correctly identifies a perfect monotonic relationship.
- **Noisy**: $r_{xy} \approx 0.86$, $r_s = 0.90$. Noise causes $y_2 > y_3$, swapping ranks 2 and 3. Both correlations decrease, but Spearman specifically reflects the broken monotonicity.

2.6. Correlation is not Causation

A non-zero r_{xy} (or r_s) quantifies statistical *association* between x and y, but it says nothing about whether x causes y. Several distinct mechanisms can produce the same correlation:

- *Direct causation*: x genuinely drives y.
- *Reverse causation*: y drives x, not the other way around.
- *Confounding (common cause)*: a third variable z influences both x and y, inducing an association between them even though neither causes the other.
- *Selection bias*: the way the dataset was sampled induces a correlation that does not exist in the underlying population.
- *Coincidence*: in small samples, or when many variable pairs are tested, large $|r_{xy}|$ values arise purely by chance (spurious correlation).

Example 2.2: In a coastal town, daily ice-cream sales (x) and drowning incidents (y) are strongly positively correlated. Ice-cream does not cause drowning; both are driven by a confounder, summer temperature, which independently increases ice-cream consumption and the number of swimmers at the beach.

> Establishing causation requires a controlled experiment or a dedicated causal-inference framework. Correlation magnitude alone, no matter how close to ± 1, is not evidence of a causal link.

Further Reading

- Spurious Correlations by Tyler Vigen is a great illustration of the principle that *correlation is not causation.*

3. Linear Least-Squares

Goal: Re-write LS in the matrix notation and extend it to the multi-variate case.

Contents

3.1. Uni-variate LS **35**
3.2. Multivariate LS **36**
3.2.1. Notation 36
3.3. Properties **37**
3.3.1. Moore–Penrose inverse (pseudo-inverse) 37
3.3.2. Projection matrix 37
3.3.3. Error properties 38
3.3.4. Implementation note 40
3.4. Gradient Descent **40**
3.4.1. Preface 40
3.4.2. Definition 41
3.4.3. Full-batch, mini-batch and stochastic GD 42

3.1. Uni-variate LS

Goal: Rewrite univariate LS from Ch. 2 with linear algebra.

To improve the mathematical representation, vector notation can be used. This time, the points $\{x_k, y_k\}_{k=1}^M$ are organized into vectors, with a few additional ones, as follows,

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_M \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{bmatrix}, \quad \mathbf{1}_M = \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \in \mathbb{R}^M, \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} \tag{3.1}$$

The resulting model notation is

$$\hat{\mathbf{y}} = f(\mathbf{X}; \mathbf{w}) = \mathbf{1}_M w_0 + \mathbf{x} w_1 = \mathbf{X}\mathbf{w}, \tag{3.2}$$

where $\mathbf{X} = \begin{bmatrix} \mathbf{1}_M & \mathbf{x} \end{bmatrix} \in \mathbb{R}^{M \times 2}$ and $\mathbf{w} = \begin{bmatrix} w_0 & w_1 \end{bmatrix}^T$.

The model error is

$$\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}} \tag{3.3}$$

The corresponding SSE loss function (2.4) is

$$\begin{aligned} \mathcal{L}(\mathbf{w};\mathbf{X},\mathbf{y}) &= \mathbf{e}^T\mathbf{e} &&= \|\mathbf{e}\|^2 \\ &= (\mathbf{y}-\hat{\mathbf{y}})^T(\mathbf{y}-\hat{\mathbf{y}}) &&= \|\mathbf{y}-\hat{\mathbf{y}}\|^2 \\ &= (\mathbf{y}-\mathbf{X}\mathbf{w})^T(\mathbf{y}-\mathbf{X}\mathbf{w}) &&= \|\mathbf{y}-\mathbf{X}\mathbf{w}\|^2 \end{aligned} \tag{3.4}$$

The resulting normal equation solution in vector notation is given by

$$\mathbf{w}_{opt} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y} \tag{3.5}$$

Proof. The corresponding optimal minimum (Eq. (2.7)) results from the solution of normal equation (in matrix form)

$$\nabla_{\mathbf{w}}\mathcal{L}(\cdot) = -\mathbf{X}^T(\mathbf{y}-\mathbf{X}\mathbf{w}) = 0 \tag{3.6}$$

and is given by

$$\begin{aligned} \mathbf{X}^T(\mathbf{y}-\mathbf{X}\mathbf{w}) &= \mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{X}\mathbf{w} = 0 \\ \mathbf{X}^T\mathbf{y} &= \left(\mathbf{X}^T\mathbf{X}\right)\mathbf{w} \end{aligned}$$

Example 3.1: For centered ($\bar{x} = \bar{y} = 0$) variables with $w_0 = 0$,

$$w_1 = \frac{\sum_{i=1}^{M} x_i y_i}{\sum_{i=1}^{M} x_i^2}$$

3.2. Multivariate LS

Goal: Extend univariate results to multivariate ones.

3.2.1. Notation

For the multivariate N-dimensional formulation,

$$\mathbf{X} = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} & \cdots & x_{1,N} \\ 1 & x_{2,1} & x_{2,2} & \cdots & x_{2,N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{M,1} & x_{M,2} & \cdots & x_{M,N} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \vdots \\ \mathbf{x}_M^T \end{bmatrix} = \begin{bmatrix} \mathbf{1}_M & \tilde{\mathbf{x}}_1 & \tilde{\mathbf{x}}_2 & \cdots & \tilde{\mathbf{x}}_N \end{bmatrix} \in \mathbb{R}^{M\times(N+1)} \tag{3.7}$$

$$\mathbf{w} = \begin{bmatrix} w_0 & w_1 & \cdots & w_N \end{bmatrix}^T \in \mathbb{R}^{N+1} \tag{3.8}$$

where:

- each row $\mathbf{x}_i^T$ is termed *sample*, $\mathbf{x}_i = [1, x_{i,1}, \ldots, x_{i,N}]^T \in \mathbb{R}^{N+1}$, $i = 1, \ldots, M$.
- each column $\tilde{\mathbf{x}}_j \in \mathbb{R}^M$, $j = 1, \ldots, N$ is termed *feature vector*.

All the data rows in $(\mathbf{X}, \mathbf{y})$ are called dataset with M samples (or entries) and N features.

The normal equation solution (3.5) is the same for $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$, independently from the number of feature vectors.

3.3. Properties

Goal: Linear algebra properties of multivariate LS.

3.3.1. Moore–Penrose inverse (pseudo-inverse)

Goal: Inversion of non-rectangular matrices.

The Moore–Penrose inverse is the generalization of the ordinary matrix inverse for non-rectangular matrices. Assuming $\mathbf{X}$ has full column rank, its left pseudo-inverse is given by:

$$\mathbf{X}^{+} = \left(\mathbf{X}^{T}\mathbf{X}\right)^{-1}\mathbf{X}^{T} \in \mathbb{R}^{(N+1)\times M}, \tag{3.9}$$

such that multiplying it by $\mathbf{X}$ yields the identity matrix

$$\mathbf{X}^{+}\mathbf{X} = \left(\mathbf{X}^{T}\mathbf{X}\right)^{-1}\mathbf{X}^{T}\mathbf{X} = \mathbf{I}_{N+1}$$

Note that a direct implementation of $\mathbf{X}^{+}$ may have numerical stability issues due to inversion of $\left(\mathbf{X}^{T}\mathbf{X}\right)^{-1}$. All the modern programming languages have numerically-stable and efficient implementation of pseudo-inverse calculations.

Using this notation, the optimal weight vector in (3.5) can be compactly written as

$$\mathbf{w}_{opt} = \mathbf{X}^{+}\mathbf{y} \tag{3.10}$$

3.3.2. Projection matrix

Goal: Derive the projection matrix $\mathbf{P}$ that maps $\mathbf{y}$ to $\hat{\mathbf{y}}$, and state its key properties.

Using the pseudo-inverse notation, the model's output (the predicted values) can be written directly in terms of the target vector $\mathbf{y}$:

$$\begin{aligned}\hat{\mathbf{y}} &= \mathbf{X}\mathbf{w} = \mathbf{X}\left(\mathbf{X}^{T}\mathbf{X}\right)^{-1}\mathbf{X}^{T}\mathbf{y} \\ &= \mathbf{X}\mathbf{X}^{+}\mathbf{y} = \mathbf{P}\mathbf{y}\end{aligned} \tag{3.11}$$

where

$$\mathbf{P} = \mathbf{X}\left(\mathbf{X}^{T}\mathbf{X}\right)^{-1}\mathbf{X}^{T} \in \mathbb{R}^{M\times M} \tag{3.12}$$

is termed the *projection matrix*. Geometrically, $\mathbf{P}$ projects the target vector $\mathbf{y}$ onto the subspace spanned by the columns of $\mathbf{X}$ (the column space).

Important properties of the projection matrix $\mathbf{P}$ include:

- Symmetry, $\mathbf{P} = \mathbf{P}^{T}$,
- Idempotence, $\mathbf{P} = \mathbf{P}^{2}$,

Proof.

$$\begin{aligned}\mathbf{P}^2 &= \mathbf{X}\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{X}\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T \\ &= \mathbf{X}\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T = \mathbf{P}\end{aligned} \tag{3.13}$$

- Orthogonality: The projection $\mathbf{P}$ is orthogonal to its complement $(\mathbf{I}-\mathbf{P})$, i.e. $\mathbf{P} \perp (\mathbf{I}-\mathbf{P})$.

  Proof. $\mathbf{P}(\mathbf{I}-\mathbf{P}) = \mathbf{P}-\mathbf{P}^2 = \mathbf{0}$.
- Complementary projection, $\mathbf{I}-\mathbf{P}$ is also projection matrix.
- The residual error vector $\mathbf{e}$ is simply the projection of $\mathbf{y}$ onto the orthogonal complement of the column space of $\mathbf{X}$

$$\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}} = \mathbf{y} - \mathbf{P}\mathbf{y} = (\mathbf{I}-\mathbf{P})\,\mathbf{y}. \tag{3.14}$$

3.3.3. Error properties

Error and prediction orthogonality

$$\mathbf{e} \perp \hat{\mathbf{y}} \Rightarrow \hat{\mathbf{y}}^T\mathbf{e} = \mathbf{e}^T\hat{\mathbf{y}} = 0 \tag{3.15}$$

Proof. Using the properties of the projection matrix $\mathbf{P}$ (symmetry and idempotence):

$$\begin{aligned}\hat{\mathbf{y}}^T\mathbf{e} &= (\mathbf{P}\mathbf{y})^T(\mathbf{I}-\mathbf{P})\,\mathbf{y} \\ &= \mathbf{y}^T\mathbf{P}^T(\mathbf{I}-\mathbf{P})\,\mathbf{y} \\ &= \mathbf{y}^T\mathbf{P}\mathbf{y} - \mathbf{y}^T\mathbf{P}^2\mathbf{y} \\ &= \mathbf{y}^T\mathbf{P}\mathbf{y} - \mathbf{y}^T\mathbf{P}\mathbf{y} = 0\end{aligned}$$

The interesting outcome of this property is a relation between error and prediction,

$$\|\mathbf{y}\|^2 = \|\hat{\mathbf{y}}\|^2 + \|\mathbf{e}\|^2 \tag{3.16}$$

Proof.

$$\begin{aligned}\|\mathbf{y}\|^2 &= \|\hat{\mathbf{y}} + \mathbf{e}\|^2 \\ &= (\hat{\mathbf{y}} + \mathbf{e})^T(\hat{\mathbf{y}} + \mathbf{e}) \\ &= \hat{\mathbf{y}}^T\hat{\mathbf{y}} + 2\hat{\mathbf{y}}^T\mathbf{e} + \mathbf{e}^T\mathbf{e} \\ &= \|\hat{\mathbf{y}}\|^2 + 0 + \|\mathbf{e}\|^2 \qquad (\text{since } \hat{\mathbf{y}}^T\mathbf{e} = 0)\end{aligned}$$

Error and feature orthogonality

$$\mathbf{e} \perp \mathbf{X} \Rightarrow \mathbf{X}^T\mathbf{e} = \mathbf{0}_{N+1} \tag{3.17}$$

Proof. Substituting the projection matrix notation:

$$\begin{aligned}\mathbf{X}^T\mathbf{e} &= \mathbf{X}^T(\mathbf{y} - \hat{\mathbf{y}}) \\ &= \mathbf{X}^T\mathbf{y} - \mathbf{X}^T\mathbf{X}\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y} \\ &= \mathbf{X}^T\mathbf{y} - \mathbf{I}_{N+1}\mathbf{X}^T\mathbf{y} = \mathbf{0}_{N+1}\end{aligned}$$

Since $\mathbf{X}^T\mathbf{e} = \mathbf{0}_{N+1}$ expands row-by-row as

$$\mathbf{1}_M^T\mathbf{e} = 0, \quad \tilde{\mathbf{x}}_1^T\mathbf{e} = 0, \quad \ldots, \quad \tilde{\mathbf{x}}_N^T\mathbf{e} = 0,$$

the error vector is orthogonal to every feature vector (and to the bias column).

Average error

If the model includes a bias/intercept term (w_0), the mean of the residuals is exactly zero,

$$\bar{\mathbf{e}} = \frac{1}{M}\sum_{k=1}^{M} e_k = \frac{1}{M}\mathbf{1}^T\mathbf{e} = 0 \tag{3.18}$$

Proof. From Eq. (3.17), we know $\mathbf{X}^T\mathbf{e} = \mathbf{0}_{N+1}$. Because the first column of $\mathbf{X}$ is $\mathbf{1}_M$ (the bias feature),

$$\mathbf{X}^T\mathbf{e} = \begin{bmatrix}\mathbf{1}_M^T \\ \tilde{\mathbf{x}}_1^T \\ \vdots \\ \tilde{\mathbf{x}}_N^T\end{bmatrix}\mathbf{e} = \begin{bmatrix}\mathbf{1}_M^T\mathbf{e} \\ \vdots\end{bmatrix} = \mathbf{0}_{N+1} \implies \mathbf{1}_M^T\mathbf{e} = 0$$

An interesting consequence of this is that the mean of the true values equals the mean of the predicted values

$$\begin{aligned}\bar{\mathbf{y}} &= \bar{\hat{\mathbf{y}}} \\ &= w_0 + w_1\bar{\tilde{\mathbf{x}}}_1 + \cdots + w_N\bar{\tilde{\mathbf{x}}}_N\end{aligned} \tag{3.19}$$

Proof.

$$\begin{aligned}\bar{\mathbf{y}} &= \overline{\hat{\mathbf{y}} + \mathbf{e}} \\ &= \bar{\hat{\mathbf{y}}} + \bar{\mathbf{e}}\end{aligned} \tag{3.20}$$

Following this property, $\bar{\mathbf{y}} = 0$ implies $w_0 = 0$ and $\bar{\mathbf{e}} = \mathbf{0}$.

Error distribution

The values of the error vector $\mathbf{e}$ are assumed to be normally distributed, due to Central Limit Theorem (CLT). Typically, this assumption is not needed in ML, but it is important for statistical analysis for small values of M.

Minimum SSE

The reduced expression for the resulting minimal SSE loss is

$$\begin{aligned}\mathcal{L}_{min} &= \mathbf{y}^T\mathbf{y} - \mathbf{y}^T\mathbf{X}\mathbf{w} \\ &= \mathbf{y}^T\mathbf{y} - \mathbf{y}^T \begin{bmatrix} \mathbf{1}_M & \tilde{\mathbf{x}}_1 & \cdots & \tilde{\mathbf{x}}_N \end{bmatrix} \mathbf{w} \\ &= \|\mathbf{y}\|^2 - w_0 \mathbf{1}_M^T \mathbf{y} - \sum_{j=1}^{N} w_j \, \tilde{\mathbf{x}}_j^T \mathbf{y}\end{aligned} \tag{3.21}$$

Proof.

$$\begin{aligned}sse_{min} &= \mathbf{e}^T\mathbf{e} \\ &= (\mathbf{y} - \hat{\mathbf{y}})^T\mathbf{e} \\ &= \mathbf{y}^T\mathbf{e} - \hat{\mathbf{y}}^T\mathbf{e} \qquad \leftarrow (3.15) \\ &= \mathbf{y}^T\mathbf{e} \\ &= \mathbf{y}^T(\mathbf{y} - \mathbf{X}\mathbf{w})\end{aligned} \tag{3.22}$$

The MSE or RMSE evaluation from the loss is straightforward by dividing by M and taking the square root, respectively.

3.3.4. Implementation note

The matrix $\mathbf{X}$ is assumed *full-rank*, i.e. columns are linearly independent and the corresponding eigenvalues of $\mathbf{X}^T\mathbf{X}$ are sufficiently large. The straightforward approach is to use numerically optimized algorithms for $\mathbf{w}_{opt}$ solution given $\mathbf{X}$ and $\mathbf{y}$, such as:

1. Matlab: `lsqminnorm`
2. Python: `numpy.linalg.lstsq`
   and `scipy.linalg.lstsq`

3.4. Gradient Descent

Goal: Iterative algorithm to find the minimum of the function:

- First-order derivative based algorithm.
- Local minimum only.

3.4.1. Preface

Goal: Basic mechanism behind the algorithm.

Let's assume some continuous function $y = f(x)$, with $x, y \in \mathscr{R}$, differentiable with continuous $\dfrac{dy}{dx} = f'(x)$.

- The interpretation of $f'(x)$ is the slope of $f(x)$ at a point x.
- By the definition of the derivative, for some small ϵ,

$$f(x+\epsilon) \approx f(x) + \epsilon f'(x) \tag{3.23a}$$
$$f(x-\epsilon) \approx f(x) - \epsilon f'(x) \tag{3.23b}$$

- Given the sign of the derivative,

$$\begin{array}{lll} f(x-\epsilon) < f(x), & & f'(x) > 0 \\ f(x+\epsilon) < f(x), & f'(x) < 0 & f(x+\epsilon) > f(x), \quad f'(x) > 0 \\ f(x-\epsilon) > f(x), & & f'(x) < 0 \end{array} \tag{3.24}$$

- For sufficiently small ϵ,

$$f\left(x - \epsilon \operatorname{sign}\left(f'(x)\right)\right) \leq f(x) \tag{3.25}$$

The idea of the algorithm is to reduce $f(x)$ toward the minimum by moving in the direction opposite to the sign of the derivative $f'(x)$ according to (3.25).

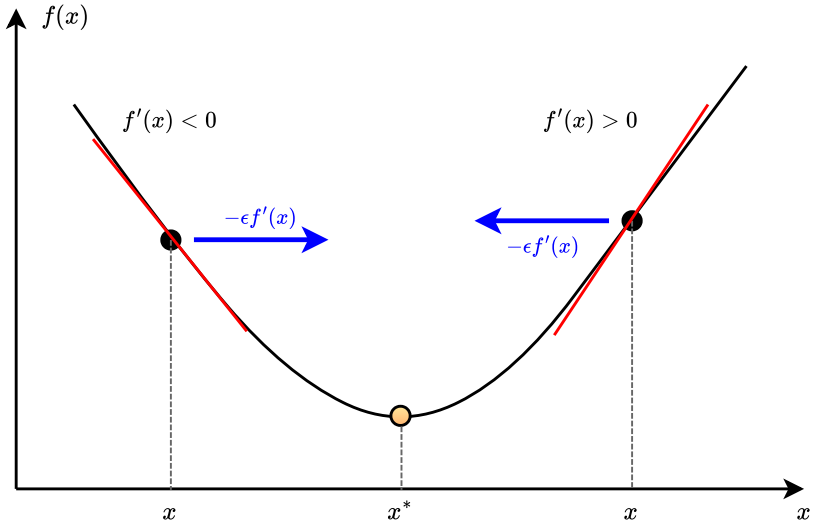


Figure 3.1.: Gradient descent intuition: moving in the direction $-\epsilon f'(x)$ reduces the function value toward the minimum.

3.4.2. Definition

Gradient descent (GD) - scalar function: For differentiable function $f(x)$, the iterative algorithm

$$x_{n+1} = x_n - \alpha f'(x_n) \tag{3.26}$$

converges to some local minimum of $f(x)$.

Required parameters are:

- Step-size $\alpha > 0$ is some positive constant or some function of n, i.e. α_n.
- x_0 is an initial guess.

Some of the most common stopping conditions are:

- Reaching the point of slow convergence, $\left|x_{n+1} - x_n\right| < \epsilon$.
- Limiting the number of iterations, $n \leq n_0$.

Gradient descent (GD) - vector function: For differentiable multivariate and multidimensional function $f(\mathbf{x}) : \mathscr{R}^N \to \mathscr{R}$, the iterative algorithm is

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \alpha \nabla_{\mathbf{x}} f(\mathbf{x}_n). \tag{3.27}$$

Each dimension is iteratively reduced according to its derivative.

Notes:

- Easy to implement.
- Requires analytical or numerical derivative.
- Non-trivial selection of the optimal value of α. Inappropriate selection of α results in slower convergence (or divergence in extreme cases). Typically, $\boldsymbol{\alpha}_n$ vector with non-equal values may be desirable.
- Useful only for the function with single (global) minimum, such as MSE minimization.

Minimization of loss function Optimal values of $\mathbf{w}$ may be found by substitution of the gradient of the loss function into (vector) GD,

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \alpha \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}) \tag{3.28}$$

GD for LS

Optimal values of $\mathbf{w}$ may be found by substitution of the gradient (Eq. (3.6)) into (3.28),

$$\begin{aligned} \mathbf{w}_{n+1} &= \mathbf{w}_n - \alpha \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}) \\ &= \mathbf{w}_n - \frac{\alpha}{M} \mathbf{X}^T \left(\mathbf{X}\mathbf{w}_n - \mathbf{y}\right) \\ &= \mathbf{w}_n \left(\mathbf{I} - \frac{\alpha}{M} \mathbf{X}^T \mathbf{X}\right) + \frac{\alpha}{M} \mathbf{X}^T \mathbf{y} \end{aligned} \tag{3.29}$$

The main differences between the normal equation (3.5) and GD solution are:

- Lower computational complexity, since it does not require inversion of $\mathbf{X}^T\mathbf{X}$.
- Option for reduced memory calculation with batch gradient descent.

3.4.3. Full-batch, mini-batch and stochastic GD

Goal: Adaptation of GD for big and very big datasets that does not fit into the memory all together.

The entire dataset is $\{(\mathbf{x}_k, y_k)\}_{k=1}^M$. Recall that for LS we can write the loss as a sum over samples,

$$\mathcal{L}(\mathbf{w}; \mathbf{X}, \mathbf{y}) = \sum_{k=1}^{M} \ell_k(\mathbf{w}; \mathbf{x}_k, y_k), \tag{3.30}$$

where $\ell_k(\cdot)$ is a single raw loss between y_k and $\hat{y}_k$.

For the SSE loss,

$$\ell_k(\mathbf{w}; \mathbf{x}_k, y_k) = e_k^2 = \left(y_k - \mathbf{x}_k^T \mathbf{w}\right)^2 \tag{3.31}$$

Full-batch

Full-batch GD uses the gradient of the *whole* loss elements, M. In this case, for LS, the loss gradient takes the form of (3.29).

Mini-batch GD

Goal: Split dataset into parts, typically due to memory constraints.

Instead of using all M samples, we select (usually at random) at iteration n a subset (mini-batch) of indices

$$\mathcal{B}_n \subset \{1, 2, \ldots, M\}, \qquad |\mathcal{B}_n| = B \ll M.$$

We define the corresponding sub-matrices containing rows indexed by $\mathcal{B}_n$,

$$\mathbf{X}_{\mathcal{B}_n} \in \mathscr{R}^{B\times(N+1)}, \qquad \mathbf{y}_{\mathcal{B}_n} \in \mathscr{R}^{B}, \tag{3.32}$$

The mini-batch loss is

$$\mathcal{L}_{\mathcal{B}_n}(\mathbf{w}) = \frac{1}{B}\sum_{k\in\mathcal{B}_n} \ell_k(\mathbf{w}; \mathbf{x}_k, y_k), \tag{3.33}$$

where B is called the *batch size*.

Mini-batch GD for LS For the SSE loss, the mini-batch loss becomes

$$\mathcal{L}_{\mathcal{B}_n}(\mathbf{w}) = \frac{1}{B}\left\|\mathbf{y}_{\mathcal{B}_n} - \mathbf{X}_{\mathcal{B}_n}\mathbf{w}\right\|^2 \tag{3.34}$$

and its gradient is

$$\nabla_{\mathbf{w}}\mathcal{L}_{\mathcal{B}_n}(\mathbf{w}) = \frac{1}{B}\mathbf{X}_{\mathcal{B}_n}^T\Big(\mathbf{X}_{\mathcal{B}_n}\mathbf{w} - \mathbf{y}_{\mathcal{B}_n}\Big). \tag{3.35}$$

The mini-batch GD update becomes

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \frac{\alpha}{B}\mathbf{X}_{\mathcal{B}_n}^T\Big(\mathbf{X}_{\mathcal{B}_n}\mathbf{w}_n - \mathbf{y}_{\mathcal{B}_n}\Big). \tag{3.36}$$

Stochastic gradient descent (SGD) The special extreme case of $B = 1$ is termed *stochastic gradient descent*.

Epoch

The term *epoch* denotes one full pass over the entire dataset. In full-batch GD, one iteration already uses all M samples, so one iteration is equivalent to one epoch. In mini-batch or stochastic GD, an epoch consists of several iterations, each using only a (small) part of the data.

Assume that we sweep once over all M samples without repetition:

- Full-batch GD: $B = M \Rightarrow L = 1$ update per epoch.

- Mini-batch GD: B samples per update results in

$$L = \left\lceil \frac{M}{B} \right\rceil \tag{3.37}$$

  batches (updates) per epoch.
- SGD: $B = 1 \Rightarrow L = M$ updates per epoch.

Theoretical relation between mini-batch and full-batch

Theoretically, mini-batches $\mathcal{B}_n$ are estimators of the full gradient,

$$\mathbb{E}\left[\nabla_{\mathbf{w}} \mathcal{L}_{\mathcal{B}_n}(\mathbf{w})\right] = \frac{1}{L} \sum_{n=1}^{L} \nabla_{\mathbf{w}} \mathcal{L}_{\mathcal{B}_n}(\mathbf{w}) = \frac{1}{M} \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}). \tag{3.38}$$

and follow (on average) the same direction as full-batch GD. Practically, since $\mathbf{w}$ is updated for each n, it results with an additional *noise* as illustrated in Fig. 3.2.

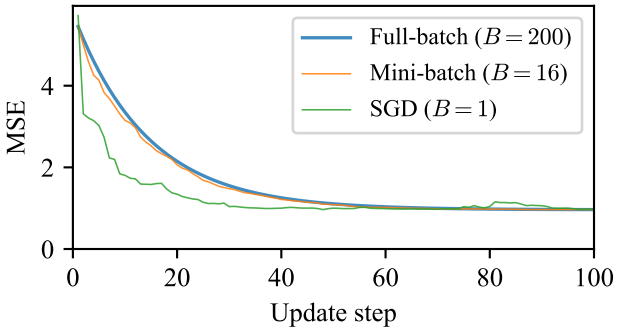


Figure 3.2.: Mini-batch gradient noise on a toy LS problem ($y = 2x + 1 + \epsilon$, $M = 100$, $\alpha = 0.04$): full-batch GD (B=M) yields a smooth loss trajectory, while mini-batch (B=16) and SGD (B=1) produce increasingly noisy updates.

Summary

- Mini-batch / SGD can handle huge datasets, since each update uses only B samples in memory.
- Updates are faster (less data per step), and typically allow efficient parallel computation (e.g., on GPUs).
- The noisy gradient may slow convergence and requires careful choice of step-size α (often smaller than in full-batch GD).
- In practice, mini-batch GD is the most commonly used compromise between speed, memory requirements and convergence stability.

Further Reading

- Gradient descent
  –
  – Let's play with gradient descent visualization (accessed 26-May-2026)

4. Model Characterization

Goal: Understand what makes a model perform well on new, unseen data:

- Why some models are too simple and others too complex.
- How to measure and compare model performance reliably.
- How to control model complexity and improve generalization.
- How the amount of data and its preprocessing affect results.

Contents

4.1. Uni-variate Polynomial Model **45**
4.2. Generalization **46**
4.3. Bayes-Optimal Error **48**
4.3.1. Basic explanation 48
4.3.2. Probabilistic formulation (*) 50
4.4. Bias-Variance Trade-Off **51**
4.4.1. Bias and Variance 51
4.4.2. Overfitting and underfitting 56
4.5. Cross-validation **57**
4.6. Hyper-parameter Search Strategies **60**
4.7. Normalization and Standardization **62**
4.8. Dataset Size **63**
4.9. Regularization **64**
4.9.1. Ridge Regression 65
4.10. Bias-Variance Decomposition (*) **67**
Further reading **69**
4.11. Summary **69**

4.1. Uni-variate Polynomial Model

Goal:
- Extend a linear model "engine" to polynomial models. The polynomial model is very flexible, e.g. due to the Taylor expansion theorem.
- Provide an example of a model with variable number of parameters.

The L-degree uni-variate polynomial regression model is

$$\begin{aligned}\hat{y} = f_{\mathbf{w}}(x) &= w_0 + w_1 x + w_2 x^2 + \cdots + w_L x^L \\ &= \sum_{j=0}^{L} w_j x^j\end{aligned} \tag{4.1}$$

This problem is linear by the change of variables, $z_j = x^j,\ j = 0, \ldots, L$,

$$\hat{y} = \sum_{j=0}^{L} w_j z_j \tag{4.2}$$

The corresponding prediction for the dataset $\{x_k, y_k\}_{k=1}^{M}$ can be easily written by using the matrix notation (also termed Vandermonde matrix),

$$\mathbf{X} = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^L \\ 1 & x_2 & x_2^2 & \cdots & x_2^L \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_M & x_M^2 & \cdots & x_M^L \end{bmatrix} \tag{4.3}$$

The weights vector values are straightforward by (3.5),

$$\mathbf{w}_{opt} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$$

Hyper-parameter: A hyper-parameter is a parameter that defines the model structure or the learning process, and is set before training rather than learned from the data. In contrast to the model parameters $\mathbf{w}$, which are optimized during training, hyper-parameters are chosen by the designer (further discussed in Sec. 7.3.1).

The value of L in the polynomial model is an example of a hyper-parameter, as it determines the model complexity.

4.2. Generalization

Examples of questions of the significant importance:

- Is a model $\hat{y} = f_{\mathbf{w}}(x)$ appropriate?
- How do we evaluate a model performance?
- How to select optimal hyper-parameter values, e.g. L of the polynomial model?

Goal: Understand how well a model performs beyond the data it was trained on:

- How to quantify model performance and what limits that estimate.
- Why training error alone is insufficient for evaluating a model.
- What determines a model's ability to predict on new, unseen data.

Let's assume that the dataset $(\mathbf{X}, \mathbf{y})$ are M samples drawn from some (unknown) joint probability distribution, $\mathcal{D}$. The theoretical model performance is given by some average model *metric* (not loss) over all (theoretically) possible points from $\mathcal{D}$,[1]

$$\bar{J}_{theory} = \lim_{M \to \infty} \frac{1}{M} \sum_{k=1}^{M} J(y_k, \hat{y}_k) \tag{4.4}$$

Generalization: The difference between:

- Performance metric over dataset that is used to train the model, $\bar{J}_M$.
- Theoretical performance metric $\bar{J}_{theory}$ from (4.4).

Better generalization means smaller difference between model performance and theoretical performance.

The problem is that the distribution $\mathcal{D}$ is unknown in most of the practical applications. Moreover, in practice, the value of M is (very) limited and $\bar{J}_M$ can significantly differ from $\bar{J}_{theory}$.

The generalization gap has two main sources:

- **Data-related**: limited dataset size M, sampling bias, noise in the data, and non-representative samples from $\mathcal{D}$.
- **Model-related**: model complexity mismatch (underfitting or overfitting), inappropriate model assumptions, and insufficient regularization.

In the following, methods to reduce both data-related and model-related gaps such that $\bar{J}_M \approx \bar{J}_{theory}$ will be discussed.

The generalization concept is illustrated in Fig. 4.1.

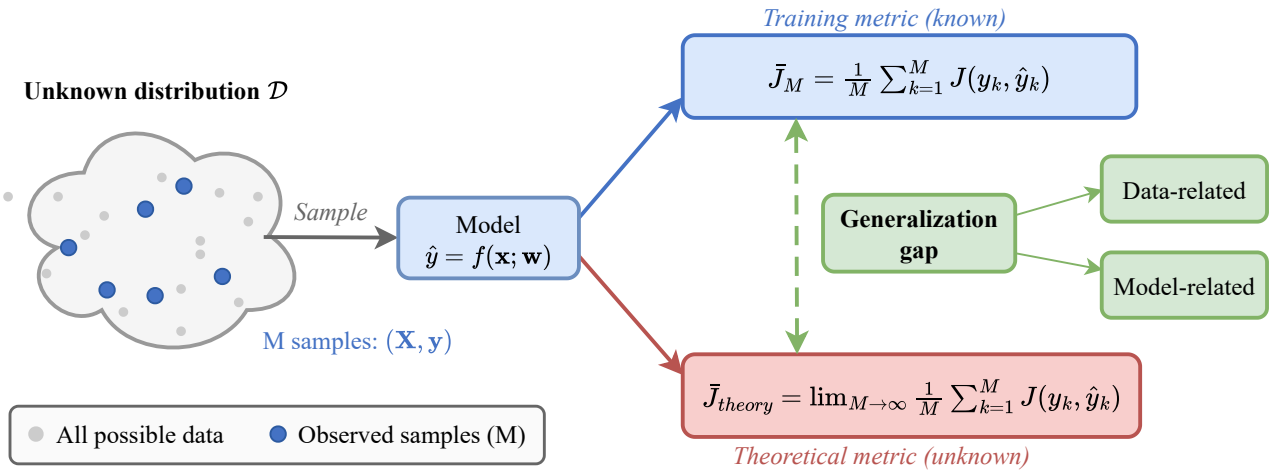


Figure 4.1.: Generalization: the gap between the training metric $\bar{J}_M$ (computed on M observed samples) and the theoretical metric $\bar{J}_{theory}$ (over the entire unknown distribution $\mathcal{D}$ of possible inputs). The gap has data-related and model-related sources.

Table 4.1 summarizes the methods discussed in this chapter and their influence on the generalization gap. Each method primarily addresses either data-related or model-related sources of the gap.

[1] Probabilistic formulation, $E_{\mathcal{D}}[J(y, \hat{y})]$

Table 4.1.: Chapter methods and their generalization influence.

Method	Data-related	Model-related
Cross-validation (Sec. 4.5)	✓	✓
Hyper-parameter tuning (Sec. 4.5)		✓
Normalization / Standardization (Sec. 4.7)	✓	
Dataset size increase (Sec. 4.8)	✓	
Regularization (Sec. 4.9)		✓

4.3. Bayes-Optimal Error

Goal:
- Define the theoretical lower bound on the error of any classifier.
- Use this bound as a yardstick to judge how close a trained model is to what is achievable on the task.

4.3.1. Basic explanation

Even with infinite training samples and an arbitrarily flexible model, some test examples will be misclassified. The reason is on the data side: two samples can have nearly identical features $\mathbf{x}$ but different labels y, so no decision boundary can separate them. The *Bayes error* R^* is the fraction of such unavoidable errors. It is the lowest test error any classifier could ever reach on the task: everything above R^* is a model limitation, everything at R^* is related to the data.

A concrete picture is shown in Fig. 4.2: two classes whose feature distributions overlap. The shaded tails are points that an optimal classifier still gets wrong, simply because the features there are equally compatible with either class.

Do not confuse *accuracy* (or its complement, test error) with the *Bayes error* R^*.
- Accuracy is a property of a *specific trained classifier*: the fraction of test examples it got right. It changes with architecture, hyper-parameters, training data, and random seed.
- R^* is a property of the *task itself*, the joint distribution $\mathcal{D}$ over $(\mathbf{x}, y)$. It is the test error of the optimal classifier and does not depend on any particular model.

Informally, any test error decomposes as

$$\underbrace{\text{test error}}_{1-\text{accuracy}} = \underbrace{R^*}_{\text{unavoidable, from the data}} + \underbrace{\text{excess error}}_{\text{model's fault: bias, variance, optimization}} .$$

Maximum achievable accuracy on the task is therefore $1 - R^*$. A classifier at 95% accuracy on a task with $R^* = 4\%$ is one point shy of optimal; the same accuracy on a task with $R^* = 0.1\%$ leaves most of the error available to be removed by a better model.

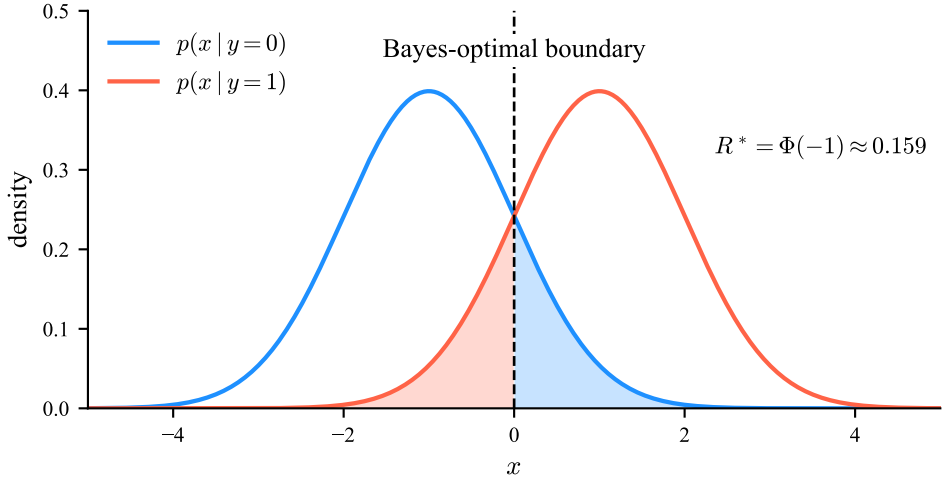


Figure 4.2.: Bayes-optimal classifier for two Gaussian classes with equal priors, means ± 1, and unit variance. The dashed line is the optimal decision boundary at $x = 0$; the shaded tails are the irreducible error regions whose total area is the Bayes error $R^* \approx 0.159$.

Theoretical range For a classification task with C possible labels, the Bayes error always lies in

$$0 \leq R^* \leq \frac{C-1}{C}. \tag{4.5}$$

The two extremes are intuitive:

- $R^* = 0$: the classes do not overlap in feature space, every $\mathbf{x}$ unambiguously identifies its label, and the optimal classifier never errs.
- $R^* = (C-1)/C$: the features carry no information about the label, so the best a classifier can do is guess uniformly at random; the optimal classifier reduces to guessing, with two classes giving $R^* = 1/2$.

Any real problem sits strictly inside this interval, and the position of R^* inside it is a property of the data, not of the algorithm.

Since the true data distribution is unknown, R^* cannot be computed exactly. Empirically it is estimated from *human-level performance* on the task, from the disagreement rate between expert annotators, or from the best published model's error after years of community effort. Representative estimates:

- Handwritten digit recognition (MNIST-style): $R^* \lesssim 0.5\%$.
- Natural-image classification (ImageNet, 1000 classes, top-5): $R^* \approx 3$ to 5%, close to trained-human error.
- Conversational speech recognition (clean audio, word error rate): $R^* \approx 4$ to 6%.
- Medical screening from images : $R^* \approx 0.5$ to 15%, depending by task.

A test error far above these reference levels indicates a model problem (bias, capacity, optimization). A test error close to them indicates that further accuracy gains will require not a better model but better labels, additional features, or a redefinition of the task.

Once the test error approaches R^*, further model tuning offers diminishing improvements. Subsequent effort is better spent on the data side: cleaner labels, additional features, or a

redefinition of the task.

4.3.2. Probabilistic formulation (*)

Bayes-optimal classifier: For a *known* joint distribution $\mathcal{D}$ over $(\mathbf{x}, y)$ with class labels $y \in \{1, \dots, C\}$, denote the *posterior* probability of class c at $\mathbf{x}$ by

$$\eta_c(\mathbf{x}) \equiv P(y = c \mid \mathbf{x}), \qquad \mathbf{p}(\mathbf{x}) = \big[\eta_1(\mathbf{x}), \dots, \eta_C(\mathbf{x})\big]^T, \tag{4.6}$$

where $\eta_c(\mathbf{x})$ is the scalar posterior for class c and $\mathbf{p}(\mathbf{x})$ collects all C posteriors into a probability vector with $\sum_c \eta_c(\mathbf{x}) = 1$. The Bayes-optimal classifier predicts the most likely class at $\mathbf{x}$,

$$\hat{y}^*(\mathbf{x}) = \arg\max_c \eta_c(\mathbf{x}), \tag{4.7}$$

and minimizes the probability of misclassification among all possible classifiers.

The *posterior* $P(y = c \mid \mathbf{x})$ answers "after observing the features $\mathbf{x}$, how likely is class c?". It is the Bayesian update of the *prior* $P(y = c)$ (class frequency before seeing $\mathbf{x}$) by the *likelihood* $P(\mathbf{x} \mid y = c)$ (how features distribute within class c), via Bayes' rule

$$\eta_c(\mathbf{x}) = P(y = c \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid y = c)\, P(y = c)}{P(\mathbf{x})}.$$

Maximizing $\eta_c(\mathbf{x})$ over c therefore picks the class that is most probable given the evidence.

At each $\mathbf{x}$, the best a classifier can do is pick the largest component of $\mathbf{p}(\mathbf{x})$; the residual per-point error is $1 - \max_c \eta_c(\mathbf{x}) = 1 - \big\|\mathbf{p}(\mathbf{x})\big\|_\infty$. Averaging over the data distribution gives the formal *Bayes error*,

$$R^* = \mathbb{E}\left[1 - \max_c \eta_c(\mathbf{x})\right]. \tag{4.8}$$

R^* is *irreducible*: no classifier can do better, regardless of complexity or training-set size. It is the classification analogue of the irreducible noise term σ^2 in the regression bias-variance decomposition (4.13) (Sec. 4.4). A classifier is called *Bayes-consistent* if its error converges to R^* as the training-set size grows, $M \to \infty$.

Example 4.1 (Two overlapping Gaussian classes): Consider a binary problem with equal priors and class-conditional densities $p(x \mid y{=}0) = \mathcal{N}(-1, 1)$ and $p(x \mid y{=}1) = \mathcal{N}(+1, 1)$. The two posteriors are equal at $x = 0$, so the Bayes-optimal classifier predicts $\hat{y}^* = 0$ for $x < 0$ and $\hat{y}^* = 1$ for $x > 0$. The Bayes error is the total probability mass each class places on the wrong side of the boundary,

$$R^* = \frac{1}{2} P(x > 0 \mid y{=}0) + \frac{1}{2} P(x < 0 \mid y{=}1) = \Phi(-1) \approx 0.159.$$

Figure 4.2 shows the densities, the optimal boundary, and the shaded tail regions whose total area equals R^*. No classifier trained on samples from this distribution can drop below $\approx 16\%$ error, however much data it sees.

4.4. Bias-Variance Trade-Off

4.4.1. Bias and Variance

Noise deterministic function model The general model beneath the dataset model is

$$y_i = h(\mathbf{x}_i) + \epsilon_i, \tag{4.9}$$

where $h(\mathbf{x})$ is some unknown function and ϵ is some random noise with unknown distribution, zero-mean ($\bar{\epsilon} = 0$) and variance of σ^2 ($\mathrm{Var}[\epsilon] = \sigma^2$).

A model predicts outputs via $\hat{y}_i = f_{\mathbf{w}}(\mathbf{x}_i)$ with error $e_i = \hat{y}_i - y_i$.

Bias The bias of the model is the expected difference between predictions and the true function,

$$\mathrm{Bias}[\hat{y}] = \mathbb{E}[\hat{y}] - \mathbb{E}[y] = \mathbb{E}\big[f_{\mathbf{w}}(\mathbf{x})\big] - h(\mathbf{x}) \tag{4.10}$$

where the expectation is taken over all possible training samples.

For a specific dataset, the empirical bias can be estimated as:

$$\begin{aligned} \mathrm{Bias}_{\text{empirical}} = \bar{\mathbf{e}} &= \frac{1}{M}\sum_{i=1}^{M} e_i \\ &= \frac{1}{M}\sum_{i=1}^{M} \hat{y}_i - \frac{1}{M}\sum_{i=1}^{M} \big(h(\mathbf{x}_i) + \cancel{\epsilon_i}\big) \end{aligned} \tag{4.11}$$

Variance The variance of the model measures the variability of predictions across different training sets,

$$\mathrm{Var}[\hat{y}] = \mathbb{E}\Big[\big(\hat{y} - \mathbb{E}[\hat{y}]\big)^2\Big] \tag{4.12}$$

The bias and variance are illustrated conceptually in Fig. 4.3.

The Explicit Bias-Variance Trade-off Formula The core of model evaluation relates to understanding the total expected prediction error (MSE). For a new unseen point $\mathbf{x}$, the expected error is explicitly given by the sum of three fundamental components:

$$\mathbb{E}\Big[(y - \hat{y})^2\Big] = \big(\mathrm{Bias}[\hat{y}]\big)^2 + \mathrm{Var}[\hat{y}] + \sigma^2 \tag{4.13}$$

where:

- *Squared Bias* $\big(\mathrm{Bias}[\hat{y}]\big)^2$: Error introduced by approximating a real-world problem with a simplified model.
- *Variance* $\mathrm{Var}[\hat{y}]$: Error introduced by the model's sensitivity to small fluctuations or noise in the training set.
- *Irreducible Noise* σ^2: The inherent variance of the data itself that no model can capture.

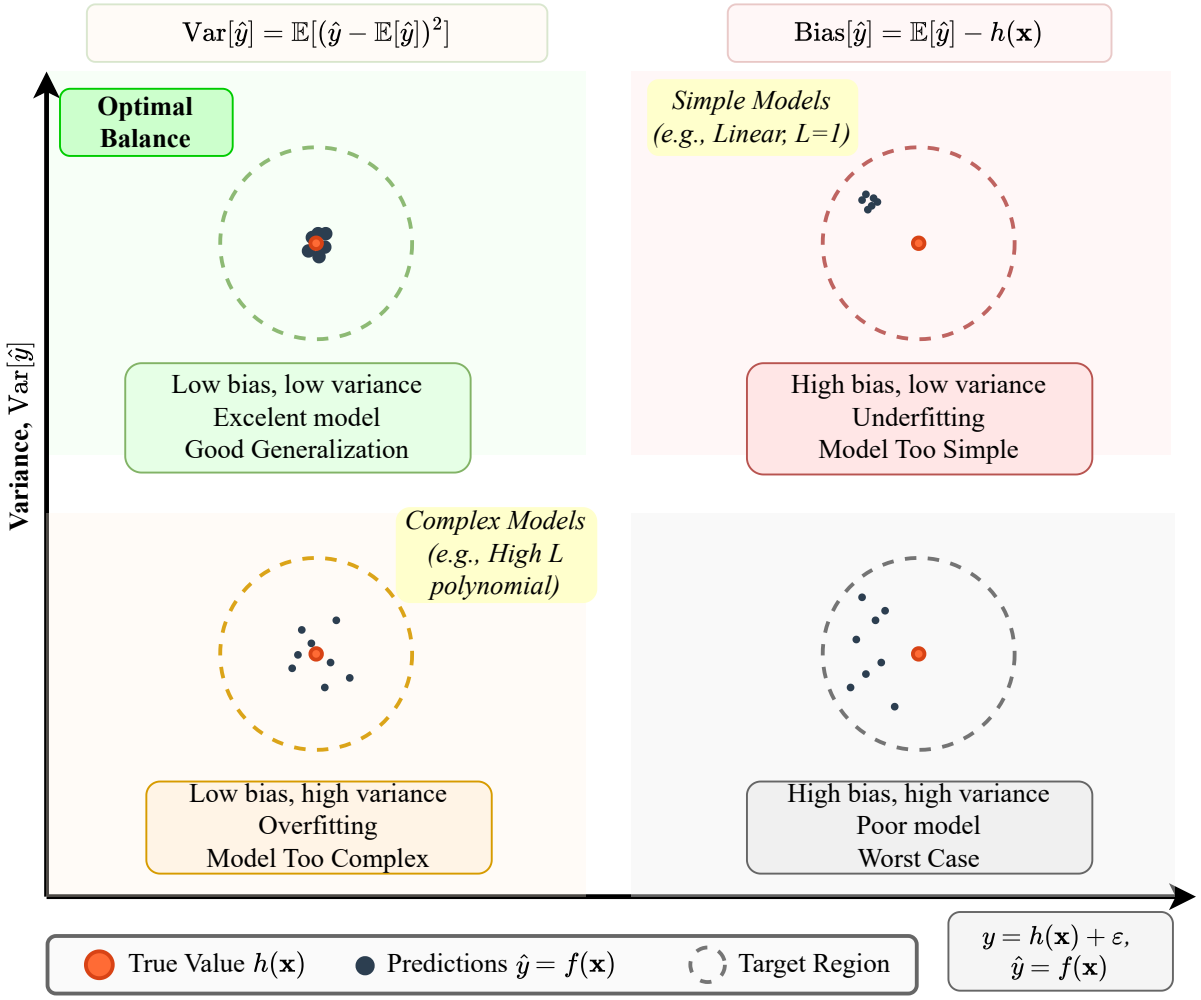


Figure 4.3.: Illustration of bias and variance.

Do not confuse *model bias* with *zero-mean error*.

- In least-squares regression with an intercept, the training residuals always satisfy $\frac{1}{n}\sum_{i=1}^{n} e_i = 0$ (Sec. 3.3.3). This is an algebraic property of the normal equations, not a statement about prediction quality.
- The bias term $\mathrm{Bias}[\hat{y}] = \mathbb{E}\left[\hat{y}(\mathbf{x})\right] - h(\mathbf{x})$ measures how far the *expected prediction averaged over all possible training sets* deviates from the true function at *unseen* points.
- A misspecified model, for example, fitting a straight line to cubic data, will have zero-mean training error in every single run, yet its expected prediction systematically misses the true curve, producing nonzero bias.

The total prediction error consists of these three components (formally derived in Sec. 4.10). The inherent bias-variance trade-off is presented in Fig. 4.4. Underfitting is characterized by low variance and high bias, whereas overfitting is characterized by high variance and low bias. The best generalized model performance balances this inherent bias-variance trade-off to minimize the overall expected error. However, note that some poorly chosen models can simultaneously exhibit both high bias and high variance.

Example 4.2 (Bias-Variance Trade-Off in Sample Variance Estimation): Recall the two

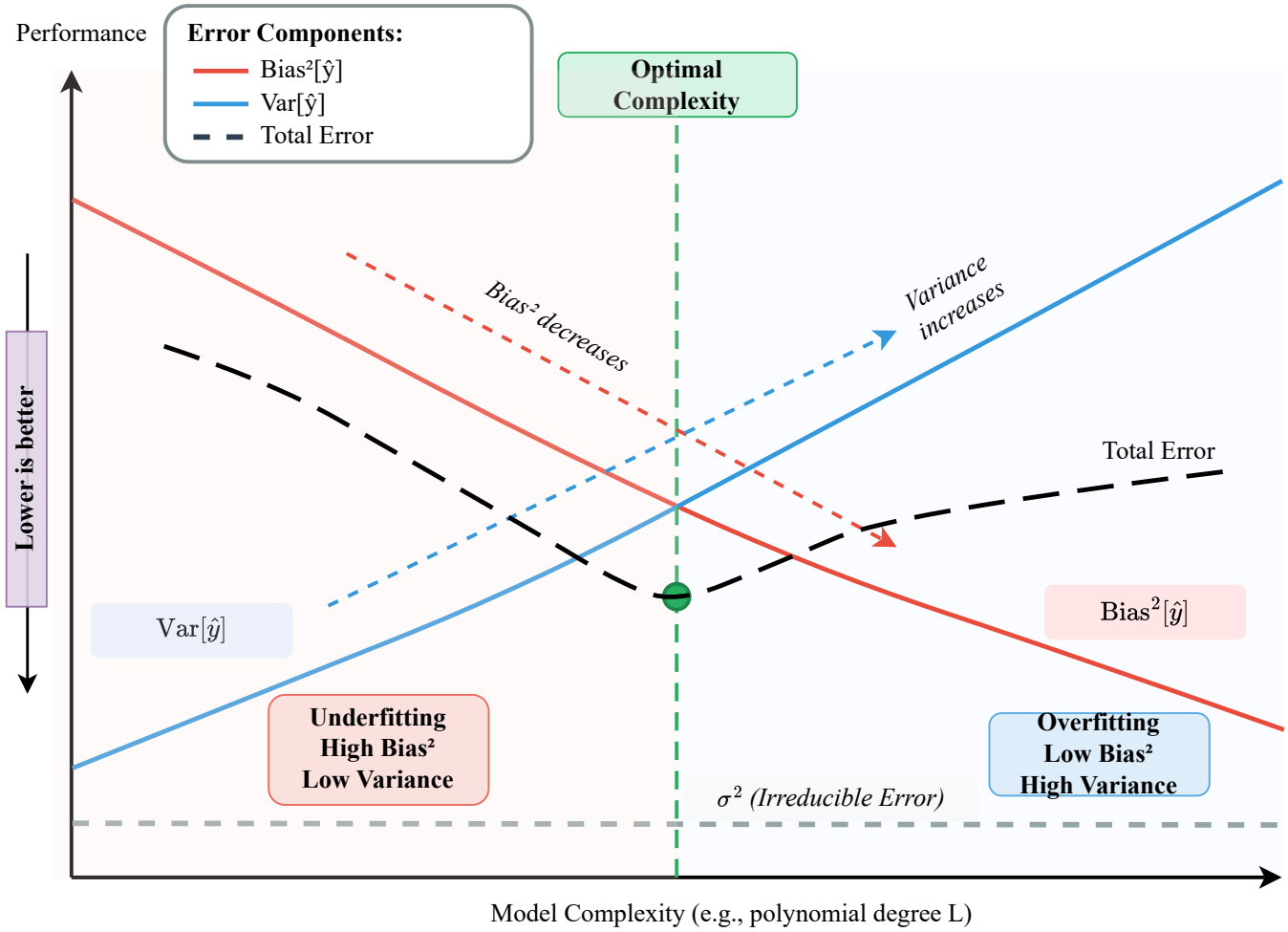


Figure 4.4.: Illustration of bias-variance trade-off.

sample variance estimators from Sec. 1.1: the biased estimator $s^2_{\text{biased}} = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$ and the unbiased estimator $s^2_{\text{unbiased}} = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$, where the true variance is σ^2. Their MSE decomposition is:

Estimator	Bias^2	Var	$\text{MSE} = \text{Bias}^2 + \text{Var}$
s^2_{biased} $\left(\frac{1}{n}\right)$	$\frac{\sigma^4}{n^2}$	$\frac{2(n-1)}{n^2}\sigma^4$	$\frac{2n-1}{n^2}\sigma^4$
s^2_{unbiased} $\left(\frac{1}{n-1}\right)$	0	$\frac{2}{n-1}\sigma^4$	$\frac{2}{n-1}\sigma^4$

Comparing the total MSE:

$$\text{MSE}_{\text{biased}} = \frac{2n-1}{n^2} < \frac{2}{n-1} = \text{MSE}_{\text{unbiased}} \quad \text{for all } n \geq 2$$

The biased estimator achieves a *lower* total error despite its nonzero bias, because eliminating the bias (by using $\frac{1}{n-1}$) increases the variance by more than the bias reduction saves. This is the bias-variance trade-off in action: a small amount of bias can be beneficial if it substantially reduces variance.

Example 4.3 (Bias-Variance Simulation with Polynomial Regression): The bias-variance decomposition (4.13) is verified numerically using a Monte-Carlo simulation (Sec. 1.6). The true data-generating function is a cubic polynomial $h(x) = 0.2 + x + 3x^2 + 0.1x^3$, corrupted by additive Gaussian noise $\epsilon \sim \mathcal{N}(0, \sigma^2)$ with $\sigma = 0.1$. For each polynomial degree $N = 1, \ldots, 8$, a total of $K = 200$ independent training sets of size $M = 25$ are sampled, and a polynomial model $\hat{y} = \mathbf{w}^\top \mathbf{x}$ is fitted to each.

Figure 4.5 illustrates the effect visually. For $N = 1$ (underfitting), the fitted curves cluster tightly but miss the true function shape — low variance, high bias. For $N = 3$ (matching the true model order), the fits are both accurate and consistent. For $N = 7$ (overfitting), individual fits vary wildly across training sets — low bias in the mean, but high variance.

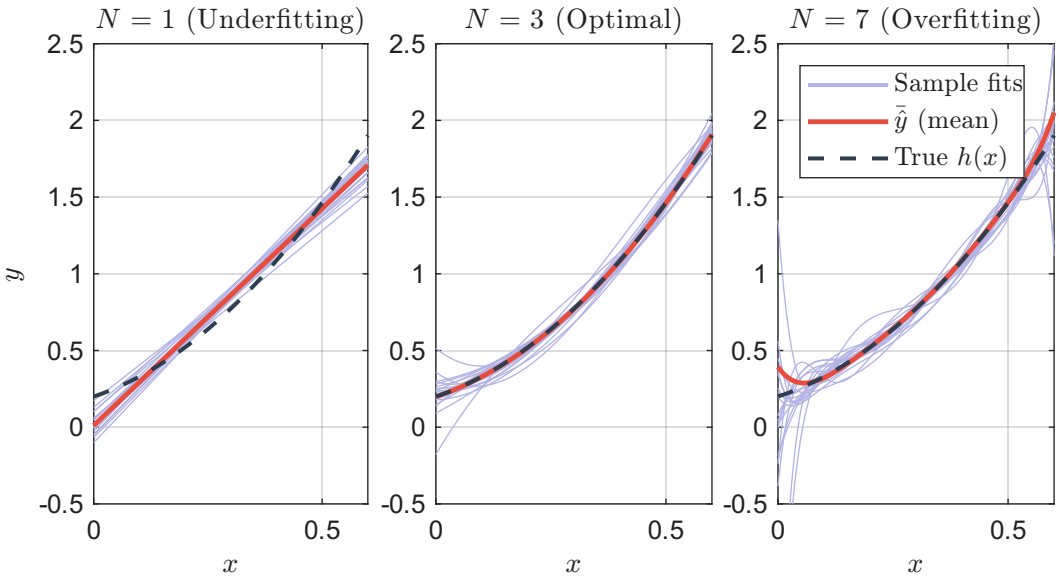


Figure 4.5.: Overlaid polynomial fits from 20 independent training sets at three complexity levels. Red: mean prediction $\bar{\hat{y}}$; dashed black: true function $h(x)$; blue: individual sample fits.

Figure 4.6 shows the squared bias, variance, and their sum (plus irreducible noise σ^2) as a function of polynomial degree. The empirical MSE (green crosses) closely matches the theoretical decomposition (black diamonds), confirming (4.13). As the model complexity increases, bias decreases while variance increases, producing the characteristic U-shaped total error curve.

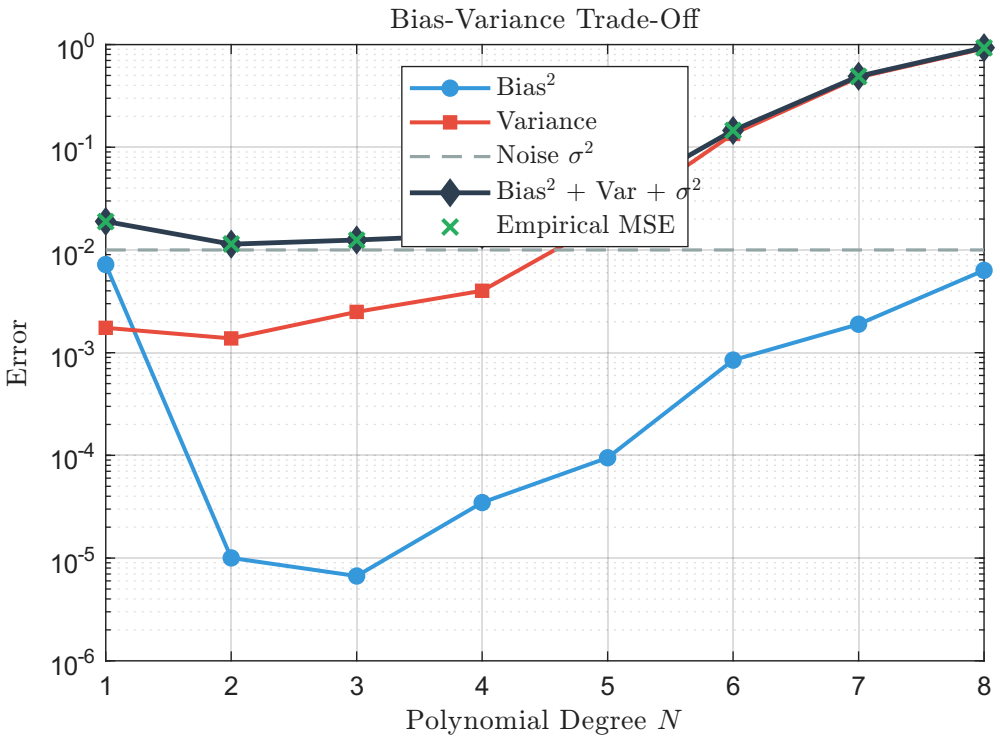


Figure 4.6.: Bias-variance trade-off: squared bias, variance, irreducible noise, and total MSE as a function of polynomial degree N, estimated from $K = 200$ Monte-Carlo runs.

4.4.2. Overfitting and underfitting

Goal: Understand underfitting and overfitting as two extremes of model complexity.

The effect of model complexity is illustrated with polynomial regression (Sec. 4.1) in Fig. 4.7. A degree-1 polynomial (top left) underfits and it cannot capture the cubic shape (see Fig. 4.5) of the data. A degree-3 model (top right) matches the true data-generating order and generalizes well to test points. Degree-6 and degree-10 models (bottom row) overfit, failing on unseen test data (blue). The annotated weight w_i in each panel shows that the largest coefficient magnitude grows with the polynomial degree, a mark of overfitting.

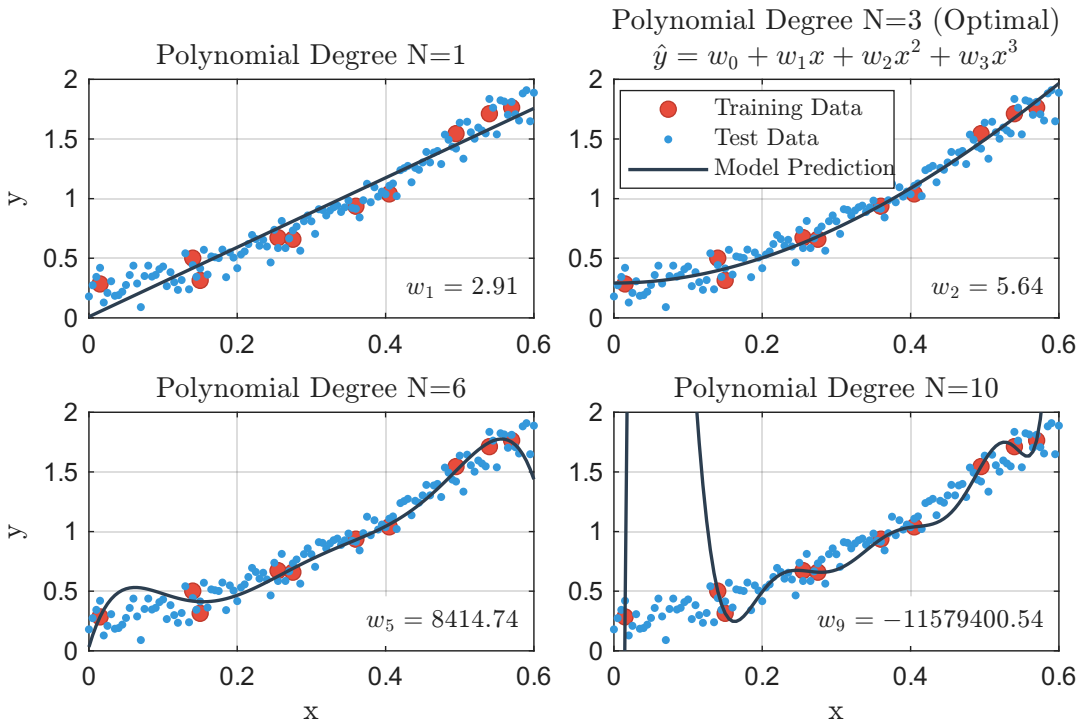


Figure 4.7.: Polynomial regression at degrees $N = 1, 3, 6, 10$. Red circles: training data ($M = 10$); blue circles: test data; black line: fitted model. The annotation shows the largest weight coefficient in each model.

To evaluate a model, the available dataset is split into a *training* subset (used to fit the model) and a *test* subset (used to evaluate it on unseen data). The formal splitting procedures are discussed in Sec. 4.5. The relationship between model complexity and performance on these two subsets reveals two failure modes.

Overfitting is when the model is too complex, i.e. have too many parameters.

- Too many parameters relative to the number of observations. Theoretically, we expect at least an order of magnitude more observations than parameters.
- Follow the training data very closely.
- Fail to generalize well to unseen data.

Significant performance difference between train and test data.

Underfitting happens when a model is too simple.

- Unable to capture the underlying pattern of the data and hence misses the trends in the data.
- Performs poorly on the training data and fail to generalize.

Poor performance on both train and test datasets.

Overfitting and underfitting are complementary and balancing between them is key to building robust machine learning models that perform well on new, unseen data, i.e. generalize well. This principle is illustrated in Fig. 4.8

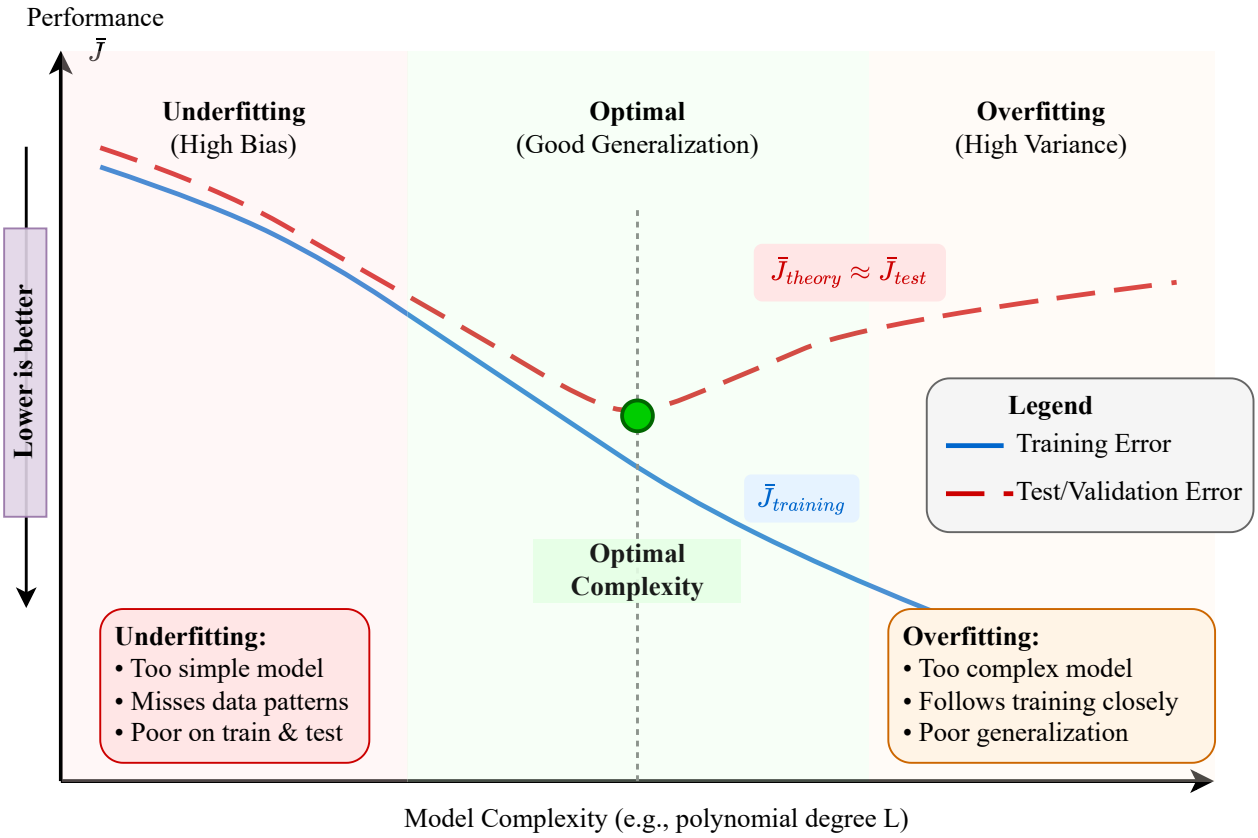


Figure 4.8.: Increasing model complexity improves the prediction error for training data, but from a certain point of transitioning from underfitting to overfitting, it increases the error for test data.

4.5. Cross-validation

Goal: Trial and error approach to quantify generalization performance. The cross-validation is also termed performance assessment. Outcomes:

- Approximated generalization performance, $\bar{J}_M \approx \bar{J}_{theory}$.
- Hyper-parameters selection guideline.
- Clear separation between train and test datasets.
- Special (validation) dataset for hyper-parameter selection.

The cross-validation methods are illustrated in Fig. 4.9.

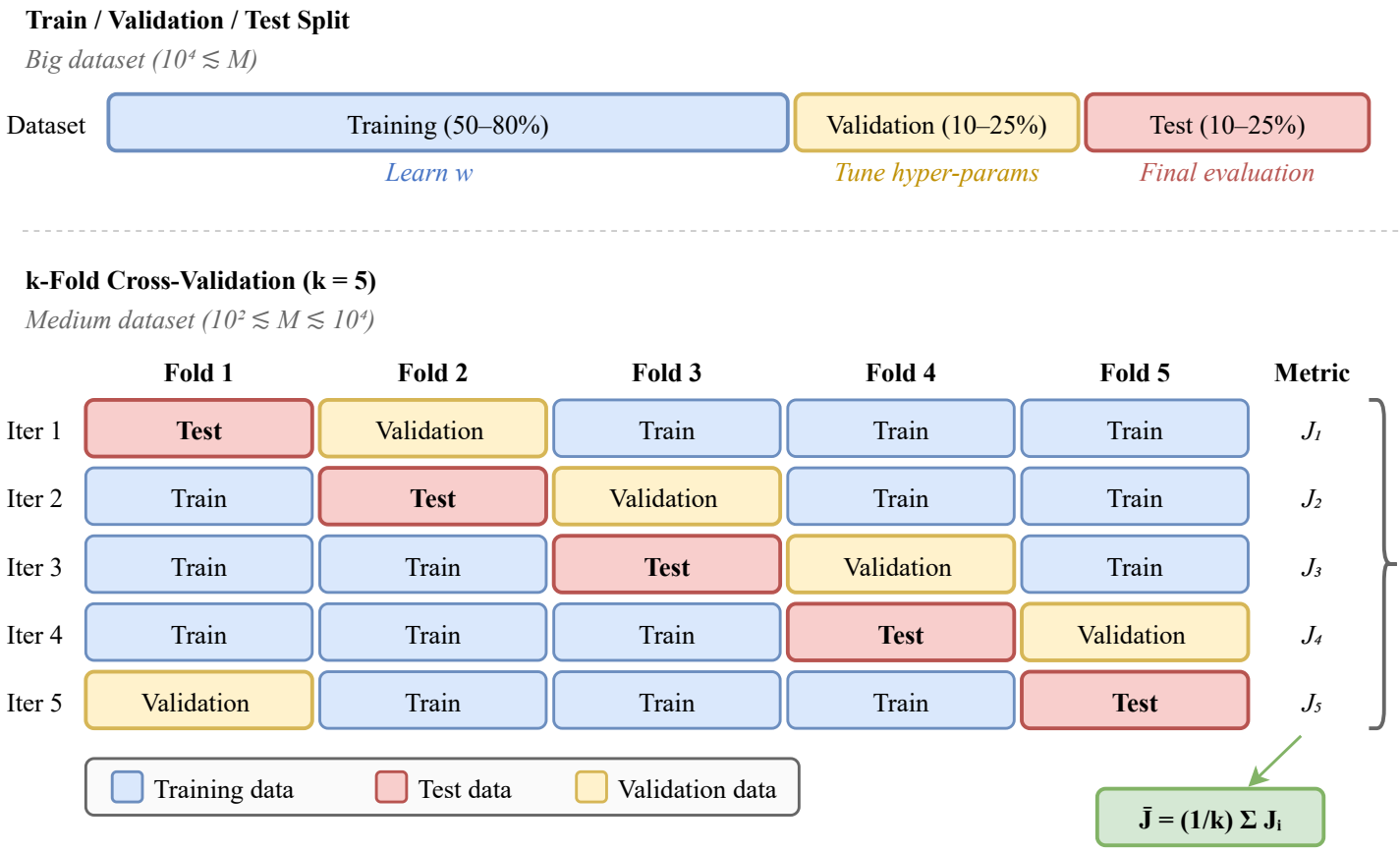


Figure 4.9.: Cross-validation illustration: train/validation/test split for big datasets (top) and nested k-fold cross-validation for medium datasets (bottom).

Big dataset $(10^4 \lesssim M)$: train/validation/test

First step is to resample the dataset into the **random order**. Then, split it into three *distinctive* datasets:

- *Training dataset* (50-80%): used for learning of model parameters, e.g. weights $\mathbf{w}$.
- *Validation dataset* (10-25%): used for assessment of model hyper-parameters influence.
- *Test dataset* (10-25%): performance assessment that is supposed to be sufficiently close to $\bar{J}_{theory}$.

Hyper-parameter selection proceeds as follows:

1. For each candidate hyper-parameter value (e.g., polynomial degree L, regularization λ), train the model on the *training* set and evaluate the metric on the *validation* set.
2. Select the hyper-parameter value that yields the best validation metric.
3. Retrain the model on the combined training and validation sets using the selected hyper-parameter, and report the final performance on the held-out *test* set.

Medium datasets $(10^2 \lesssim M \lesssim 10^4)$: Nested k-fold Cross-Validation

Goal: Simultaneously tune hyper-parameters (when applicable) and obtain an unbiased performance estimate.

The first step is to resample the dataset into a **random order**. After randomizing, we apply a nested loop procedure to separate two main objectives: hyper-parameter selection and performance assessment. Conflating these objectives into a single loop leads to optimistic performance estimates because the test data would indirectly influence model selection.

Nested cross-validation separates these objectives into two loops (Fig. 4.9):

- *Outer loop* (k_{out} folds for Performance Assessment): The available dataset is divided into k_{out} subsets (folds) of approximately equal size. In each of the k_{out} iterations, one fold (red in Fig. 4.9) is held out as the test set for unbiased performance evaluation. The remaining $k_{\text{out}} - 1$ folds form the outer training set.
- *Inner loop* (k_{in} folds for Hyper-parameter Tuning): Within each outer training set, a further k_{in}-fold split creates validation folds (yellow in Fig. 4.9, Fig. 4.10). For each candidate hyper-parameter value (e.g., polynomial degree L, regularization parameter λ), the model is trained on the inner training folds (blue) and evaluated on the inner validation fold. The hyper-parameter value that maximizes the average inner validation metric is selected.

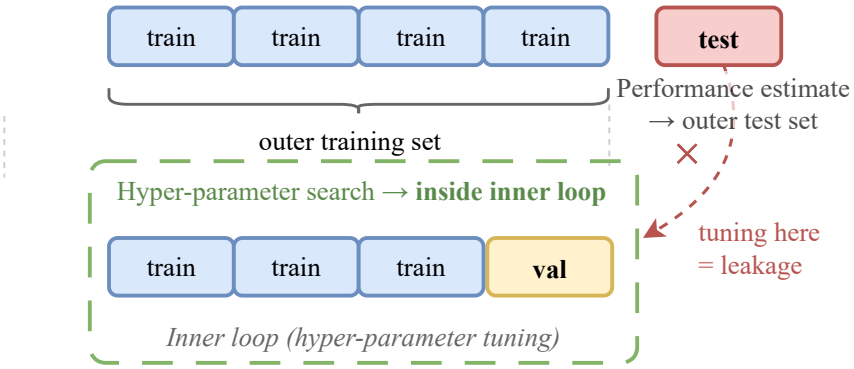


Figure 4.10.: The hyper-parameter search lives strictly inside the inner loop (green box): it sees only the inner train/validation folds carved out of the outer training set. Letting the held-out outer test fold influence the search (red) reintroduces data leakage and inflates the reported performance.

After the inner loop selects the best hyper-parameter, the model is retrained on the entire outer training set using that hyper-parameter, and its performance is evaluated on the outer test fold. The final performance estimate is the average over the k_{out} outer test metrics:

$$\bar{J} = \frac{1}{k_{\text{out}}} \sum_{i=1}^{k_{\text{out}}} J_i \tag{4.14}$$

Usually, k_{out} defaults to 5 or 10.

The hyper-parameter selected may differ across outer folds. This is expected — each outer fold sees a slightly different training distribution. The purpose of nested CV is to produce an unbiased performance estimate, not a single set of hyper-parameters.

When hyper-parameter optimization is not required the inner loop is completely omitted. The procedure simply collapses into a standard, single-loop k-fold cross-validation. The model is trained on the $k_{\text{out}}-1$ training folds and evaluated directly on the held-out test fold. This process is repeated k_{out} times, and the performance is averaged.

Grouped Cross-Validation

When data is collected from distinct domains or groups (e.g., multiple patients, different sensors, separate locations), standard random splitting can place samples from the same group in both training and test folds. This allows the model to learn domain-specific signatures rather than generalizable patterns. Grouped cross-validation ensures that all samples from a given group are assigned to the same fold, providing a realistic estimate of performance on unseen domains. Further CV splitting strategies are discussed in Sec. 13.1.1.

Example 4.4 (Grouped Cross-Validation): Consider $M = 12$ EEG recordings drawn from $G = 3$ patients, with 4 recordings each:

Recording	Patient	Standard 3-fold	Grouped 3-fold
x_1–x_4	Patient A	Folds 1, 2, 3 (split)	Fold 1 (all)
x_5–x_8	Patient B	Folds 1, 2, 3 (split)	Fold 2 (all)
x_9–x_{12}	Patient C	Folds 1, 2, 3 (split)	Fold 3 (all)

With standard 3-fold CV, each fold contains recordings from all three patients. A model tested on fold 1 has already seen other recordings from Patients A, B, and C during training, so it can exploit patient-specific signatures (e.g., baseline amplitude, electrode impedance) to improve its predictions. The resulting accuracy is optimistic.

With grouped 3-fold CV, when the model is tested on fold 1 (Patient A), it has never seen any recording from Patient A during training. This forces the model to rely on patterns that generalize across patients and provides an unbiased estimate of performance on unseen patients.

Very small datasets $\left(M \lesssim 10^2\right)$: Leave-One-Out Cross-Validation (LOOCV)

Uses k-fold with $k = M$, which means that each fold will contain only one data point.

> LOOCV is generally recommended only for very small datasets where k-fold would leave too few samples per fold. For medium and large datasets, 5- or 10-fold cross-validation typically provides a better bias-variance trade-off in the performance estimate itself.

4.6. Hyper-parameter Search Strategies

Goal: Decide *which* candidate hyper-parameter values the cross-validation loop should evaluate, given a limited search budget.

The nested cross-validation of Sec. 4.5 evaluates the metric "for each candidate hyper-parameter value", but it leaves open how that set of candidates is constructed. This is the role of a *search strategy* over the hyper-parameter space.

Search space: The set of candidate hyper-parameter configurations considered during optimization. Each axis corresponds to one hyper-parameter, either continuous (e.g. the regularization parameter λ or a learning rate, usually searched on a logarithmic scale) or discrete/categorical (e.g. the polynomial degree L or the kernel type).

The binding constraint is cost: every candidate configuration requires a full (nested) cross-validation run, so the strategy must spend a fixed budget of evaluations wisely. The three standard strategies are compared in Table 4.2.

Grid search Form the Cartesian product of a finite list of values for each hyper-parameter and evaluate every combination. The approach is exhaustive within its resolution, deterministic, and trivially parallel. Its weakness is the curse of dimensionality: with D hyper-parameters and n values each, the cost is n^D, growing exponentially with D. Worse, the resolution along each axis is fixed in advance, so budget is wasted re-evaluating hyper-parameters that barely affect the metric.

Random search Draw a fixed number T of configurations by sampling each hyper-parameter independently from a chosen distribution (e.g. log-uniform for λ). The key observation (Bergstra and Bengio, see Sec. 4.10) is that in practice only a few hyper-parameters strongly influence performance. For the same budget, a grid of $T = n^2$ points probes only n distinct values along each of two axes, whereas T random points probe up to T distinct values along *every* axis (Fig. 4.11). Random search therefore explores the few important dimensions much more finely at equal cost, and tends to find good configurations faster.

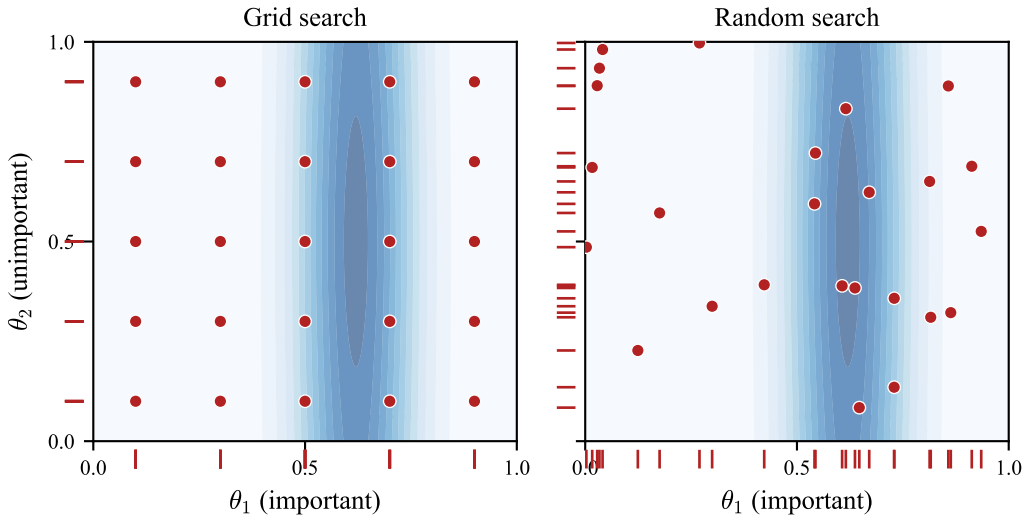


Figure 4.11.: Grid versus random search at equal budget over two hyper-parameters, where performance depends mostly on the horizontal axis (background shading). Grid search (left) probes only a few distinct values along the important axis; random search (right) probes a distinct value at every trial, covering the important axis far more finely.

Bayesian optimization Also called sequential model-based optimization (SMBO). Instead of fixing all configurations in advance, it builds a probabilistic *surrogate* model of the validation metric as a function of the hyper-parameters, and uses an *acquisition function* to choose the next configuration to evaluate. The acquisition function balances *exploitation* (regions the surrogate predicts to be good) against *exploration* (regions where the surrogate is uncertain). After each evaluation the surrogate is updated, so the search concentrates around promising configurations. This is the most sample-efficient strategy, but it is inherently sequential, each trial depends on the outcome of previous ones, so it parallelizes less readily than grid or random search. A widely used variant is the Tree-structured Parzen Estimator (TPE).

Table 4.2.: Comparison of hyper-parameter search strategies.

Strategy	Cost scaling	Parallel	Best when
Grid	n^D (exponential in D)	Fully	Few hyper-parameters, low resolution acceptable
Random	T (fixed budget)	Fully	Many dimensions of which only a few matter
Bayesian	T (fixed budget)	Limited	Evaluations expensive and budget small

Searching multiplies the cross-validation cost. With T candidate configurations and nested CV with k_{out} outer folds and k_{in} inner folds (Sec. 4.5), the total is $T \cdot k_{\text{out}} \cdot k_{\text{in}}$ model fits.

4.7. Normalization and Standardization

Goal: Data pre-processing to reduce possible overfitting.

Values in different columns in $\mathbf{X}$ (feature columns $\tilde{\mathbf{x}}_j$) may be different by orders of magnitudes, i.e. $\|\tilde{\mathbf{x}}_i\| \gg \left\|\tilde{\mathbf{x}}_j\right\|$. This results in:

- Some columns have significantly higher influence on $\hat{\mathbf{y}}$.
- Numerical instabilities.

Standardization (z-score normalization)

Goal: Mapping all the input values such that they follow a distribution with zero mean and unit variance

$$\mathbf{z} = \frac{\mathbf{x} - \bar{\mathbf{x}}}{s_{\mathbf{x}}}, \tag{4.15}$$

such that $\bar{\mathbf{z}} = 0, s_{\mathbf{z}} = 1$.

Implementation steps:[2]

1. On **train** dataset, evaluate $\bar{\mathbf{x}}$ and $\sigma_{\mathbf{x}}$.
2. Apply normalization on **train** dataset, using $\bar{\mathbf{x}}$ and $\sigma_{\mathbf{x}}$.
3. Apply normalization on **test** dataset, using **same** $\bar{\mathbf{x}}$ and $\sigma_{\mathbf{x}}$ (no recalculation).

When normalization is applied to $\mathbf{y}$, the output of the model is transformed back, $\hat{\mathbf{y}} = \hat{\mathbf{y}}_{std}\sigma_{\mathbf{y}} + \bar{\mathbf{y}}$.

Normalization

Mapping all values of a feature to be in the range $[0, 1]$ by the transformation[3]

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{4.16}$$

Implementation steps for normalization are similar to standardization.

Beware, normalization and standardization are used interchangeably.

Pros:

- Improves convergence speed of gradient-based optimization.
- Prevents features with large magnitudes from dominating distance-based models.
- Standardization of both $\mathbf{X}$ and $\mathbf{y}$ results in $w_0 = 0$ and removes the requirement for $\mathbf{1}$ column in $\mathbf{X}$.

Cons:

- Min-max normalization is sensitive to outliers.
- Scaling parameters must be computed on the training set and applied consistently to the test set.

4.8. Dataset Size

Goal: Dataset size influence on underfitting-overfitting trade-off. Understanding when more data helps and when it does not.

By the law of large numbers, the sample average $\bar{J}_M = \frac{1}{M}\sum_{k=1}^{M} J(y_k, \hat{y}_k)$ converges to the theoretical performance (4.4), so the generalization gap $\bar{J}_M - \bar{J}_{theory}$ shrinks as the number of samples M grows. However, the benefit of additional data depends on whether the model is overfitting or underfitting, as summarized in Table 4.3 and illustrated in Fig. 4.12.

Beyond a certain dataset size, performance improvement may be almost negligible or non-existent for a constraint model complexity.

[2] In Python: `sklearn.preprocessing.StandardScaler`, in Matlab zscore.
[3] In Python: `sklearn.preprocessing.MinMaxScaler`

Table 4.3.: Effect of increasing dataset size M on model performance.

	Overfitting (complex)	**Underfitting** (simple)
Training error	Rises	Stays high
Test error	Drops	Stays high
Gap (test − train)	Shrinks	Already small
More data helps?	Yes (reduces variance)	No (model lacks capacity)

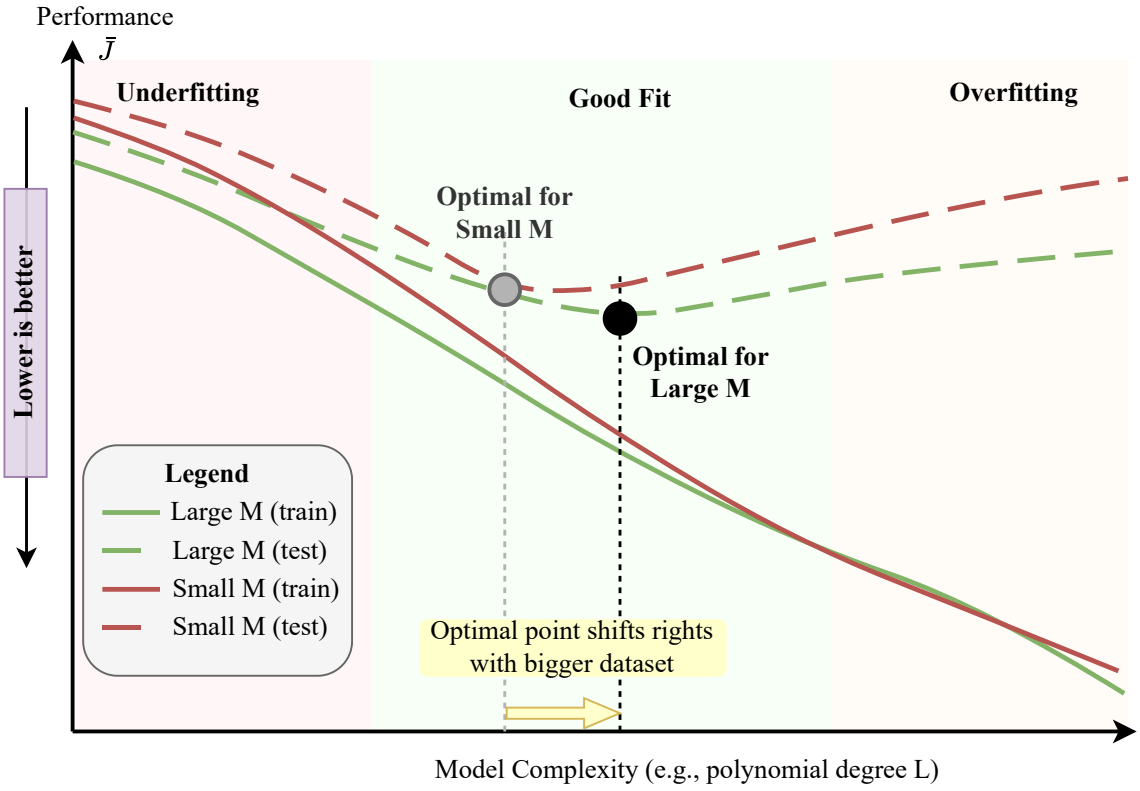


Figure 4.12.: Increase in dataset size improves the over-complex model performance.

4.9. Regularization

Goal: Loss function tweak that influences on underfitting-overfitting trade-off.

Regularization: Penalty to the loss function

$$\mathcal{L}_{\text{new}} = \mathcal{L} + \lambda g(\mathbf{w}) \tag{4.17}$$

where λ is termed *regularization parameter*.

L_2-regularization: Special case of

$$g(\mathbf{w}) = \frac{1}{2M} \|\mathbf{w}\|^2 = \frac{1}{2M} \sum_{i=1}^{N} w_i^2 \tag{4.18}$$

where vector of weights $\mathbf{w}$ does not include w_0 weight. Moreover, when normalization is used, no w_0 weight is needed. Two special cases are:

- $\lambda \to 0$ gets the original loss function (overfitting).
- $\lambda \to \infty$ makes loss function independent on $\mathbf{w}$ (underfitting).

Polynomial regression example The resulting coefficients w_i will be significantly higher for higher i (Fig. 4.7). Reducing them results in smooth $\hat{y}$ prediction. The illustration of λ influence is presented in Fig. 4.13.

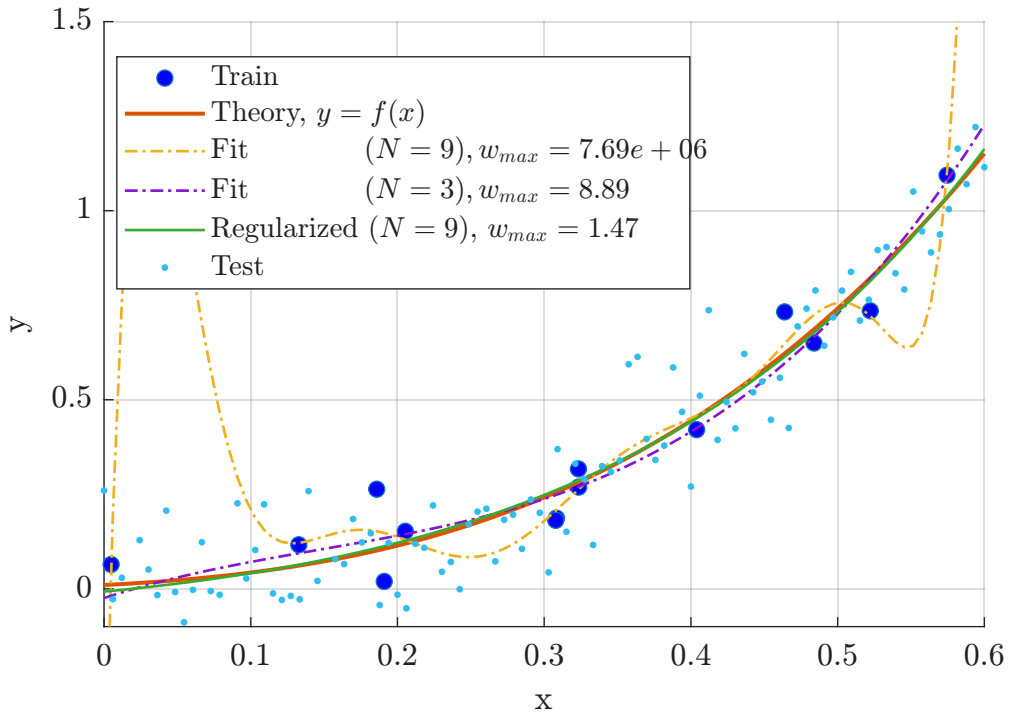


Figure 4.13.: Illustration of regularization influence on the polynomial model.

The corresponding underfitting-overfitting λ-related trade-off is presented in Fig. 4.14.

4.9.1. Ridge Regression

Ridge regression: L_2-regularized linear regression.

$$\begin{aligned}\mathcal{L}(\mathbf{w}) &= \frac{1}{2M}\left\|\mathbf{y}-\mathbf{X}\mathbf{w}\right\|^2 + \frac{\lambda}{2M}\underbrace{\left\|\mathbf{w}\right\|^2}_{\sum_{i=1}^{N} w_i^2} \\ &= \frac{1}{2M}\left(\mathbf{y}-\mathbf{X}\mathbf{w}\right)^T\left(\mathbf{y}-\mathbf{X}\mathbf{w}\right) + \frac{\lambda}{2M}\mathbf{w}^T\mathbf{w}\end{aligned} \tag{4.19}$$

Derivative

$$\nabla\mathcal{L}(\mathbf{w}) = \frac{1}{M}\left(-\mathbf{X}^T\left(\mathbf{y}-\mathbf{X}\mathbf{w}\right) + \lambda\mathbf{w}\right) = 0 \tag{4.20}$$

Solution

$$-\mathbf{X}^T\left(\mathbf{y}-\mathbf{X}\mathbf{w}\right) + \lambda\mathbf{w} = -\mathbf{X}^T\mathbf{y} + \mathbf{X}^T\mathbf{X}\mathbf{w} + \lambda\mathbf{w} = 0$$

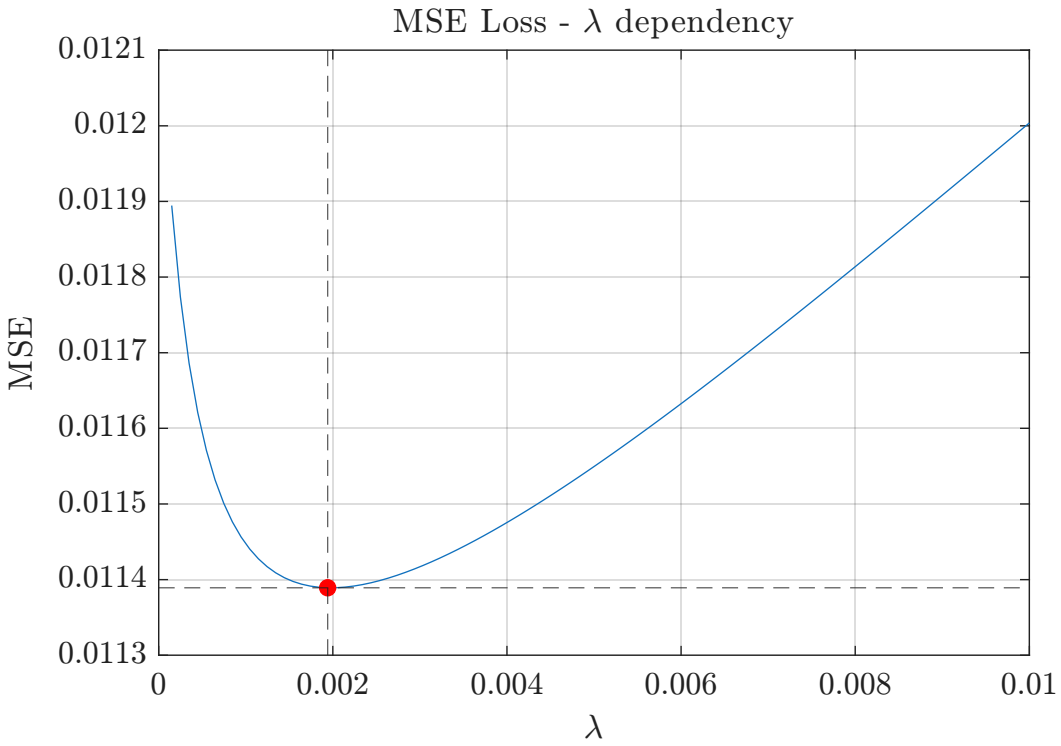


Figure 4.14.: Dependence of MSE on λ for regularization in Fig. 4.13 (polynomial regression). Overfitting is on the left ($\lambda \to 0$) and undefitting on the right ($\lambda \to \infty$).

$$\mathbf{X}^T\mathbf{y} = \mathbf{X}^T\mathbf{X}\mathbf{w} + \lambda\mathbf{w}$$

Finally, the regularized weights are given by

$$\mathbf{w} = \left(\lambda\mathbf{I} + \mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y} \tag{4.21}$$

Can be viewed as special case of linear regression, of the form

$$\begin{aligned} \tilde{y} &= \tilde{\mathbf{X}}\mathbf{w} \\ \begin{bmatrix} \mathbf{y} \\ 0 \\ \vdots \\ 0 \end{bmatrix} &= \begin{bmatrix} - & \mathbf{X} & - \\ \sqrt{\lambda} & & 0 \\ & \ddots & \\ 0 & & \sqrt{\lambda} \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_N \end{bmatrix} + w_0 \end{aligned} \tag{4.22}$$

Slope interpretation Higher value of λ reduces (mostly) the highest slopes (weights w_i) $\Rightarrow$ the dependency on the parameters with these weights is reduced.

Eigenvalues interpretation This calculation limits the smallest eigenvalues to $> \frac{1}{\lambda}$ and thus improve the numerical stability.

Bias-Variance Trade-off The bias-variance trade-off for ridge regression is illustrated in Fig. 4.15.

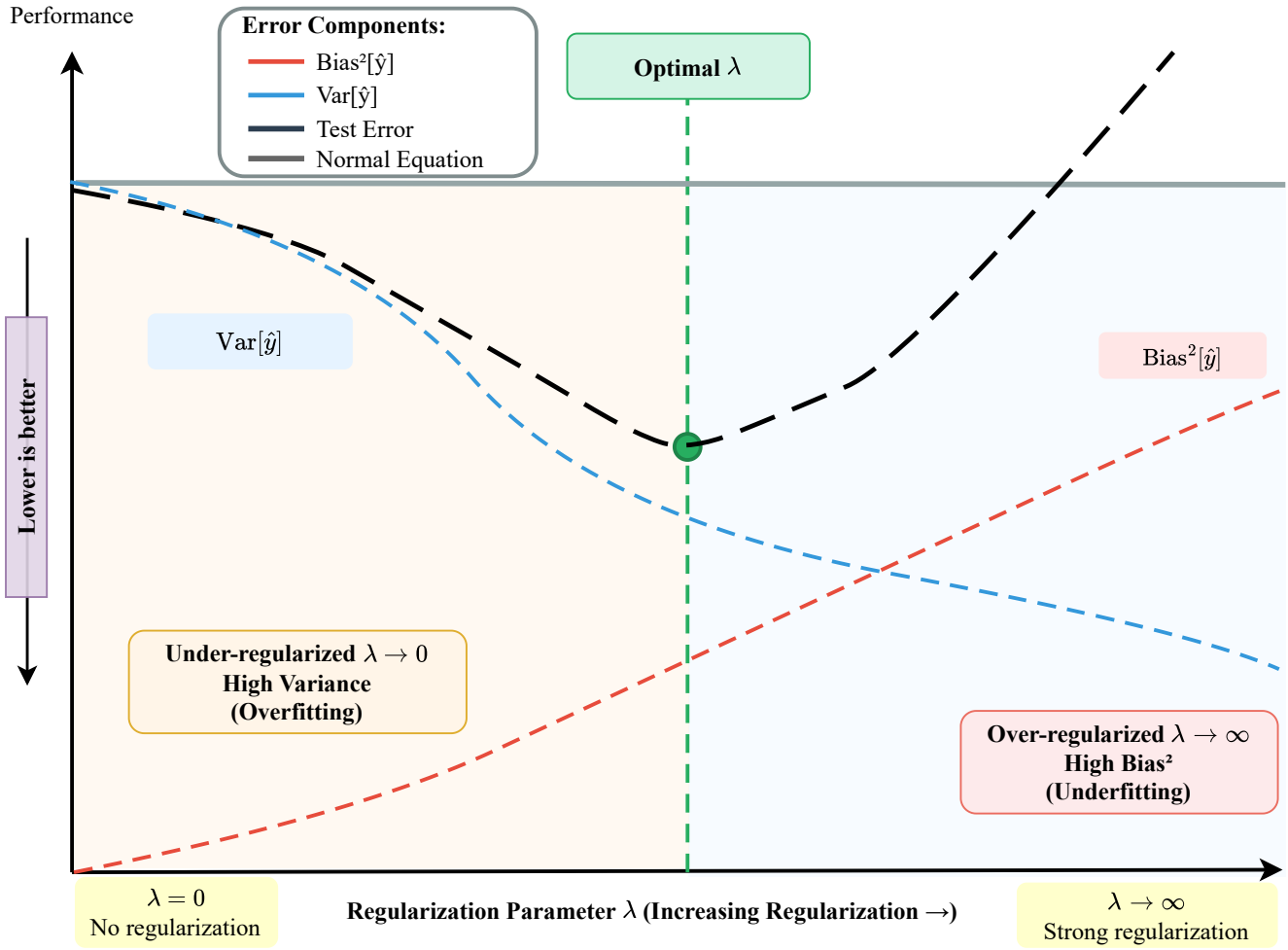


Figure 4.15.: Regularization trade-off for ridge regression: Test error $\bar{J}_{\text{test}}$ is minimized at optimal λ. Note, the trends are opposite to Fig. 4.4.

GD Solution

Tailored GD (Sec. 3.4) solution is

$$\begin{aligned} \mathbf{w}_{n+1} &= \mathbf{w}_n - \alpha \nabla \mathcal{L}(\mathbf{w}) \\ &= \mathbf{w}_n - \frac{\alpha}{M}\left[\mathbf{X}^T(\mathbf{X}\mathbf{w}_n - \mathbf{y}) + \lambda \mathbf{w}_n\right] \\ &= \mathbf{w}_n\left(1 - \frac{\alpha}{M}\lambda\right) - \frac{\alpha}{M}\mathbf{X}^T(\mathbf{X}\mathbf{w}_n - \mathbf{y}) \end{aligned} \tag{4.23}$$

The α value is typically chosen such that

$$0.99 \gtrsim \left(1 - \frac{\alpha}{M}\lambda\right) \gtrsim 0.95 \tag{4.24}$$

4.10. Bias-Variance Decomposition (*)

Goal: Discussion of inherent underfitting and overfitting trade-off in (4.13).

The expected prediction error (MSE), taken over all possible training sets and noise realizations, decomposes into three irreducible components:

$$\mathbb{E}\Big[(\hat{y} - y)^2\Big] = \big(\mathbb{E}[\hat{y}] - h(\mathbf{x})\big)^2 + \mathrm{Var}[\hat{y}] + \sigma^2 \tag{4.25}$$

where $\big(\mathrm{Bias}[\hat{y}]\big)^2 = \big(\mathbb{E}[\hat{y}] - h(\mathbf{x})\big)^2$.

Proof. Starting from:

$$\mathrm{MSE} = \mathbb{E}\Big[(\hat{y} - y)^2\Big] = \mathbb{E}\Big[\hat{y}^2\Big] - 2\mathbb{E}[y\hat{y}] + \mathbb{E}\Big[y^2\Big] \tag{4.26}$$

We evaluate each term separately. For the first term, by the variance identity:

$$\mathbb{E}\Big[\hat{y}^2\Big] = \mathrm{Var}[\hat{y}] + \mathbb{E}^2[\hat{y}] \tag{4.27}$$

For the cross-term, since $y = h(\mathbf{x}) + \epsilon$ and ϵ is independent of $\hat{y}$:

$$\mathbb{E}[y\hat{y}] = \mathbb{E}\big[(h(\mathbf{x}) + \epsilon)\hat{y}\big] = h(\mathbf{x})\mathbb{E}[\hat{y}] + \cancelto{0}{\mathbb{E}[\epsilon]}\mathbb{E}[\hat{y}] = h(\mathbf{x})\mathbb{E}[\hat{y}] \tag{4.28}$$

For the last term:

$$\mathbb{E}\Big[y^2\Big] = \mathbb{E}\Big[(h(\mathbf{x}) + \epsilon)^2\Big] = h^2(\mathbf{x}) + 2h(\mathbf{x})\cancelto{0}{\mathbb{E}[\epsilon]} + \mathbb{E}\Big[\epsilon^2\Big] = h^2(\mathbf{x}) + \sigma^2 \tag{4.29}$$

Combining all three terms:

$$\begin{aligned}\mathbb{E}\Big[(\hat{y} - y)^2\Big] &= \mathrm{Var}[\hat{y}] + \mathbb{E}^2[\hat{y}] - 2h(\mathbf{x})\mathbb{E}[\hat{y}] + h^2(\mathbf{x}) + \sigma^2 \\ &= \big(\mathbb{E}[\hat{y}] - h(\mathbf{x})\big)^2 + \mathrm{Var}[\hat{y}] + \sigma^2\end{aligned} \tag{4.30}$$

Empirical estimate For a specific dataset of M samples, the MSE can be similarly decomposed:

$$\mathrm{MSE} = \frac{1}{M}\sum_{i=1}^{M}(\hat{y}_i - y_i)^2 = \mathrm{bias}^2_{\mathrm{emp}} + \mathrm{variance}_{\mathrm{emp}} + \mathrm{noise}_{\mathrm{emp}} \tag{4.31}$$

where

$$\mathrm{bias}^2_{\mathrm{emp}} = \Big(\bar{\hat{y}} - \bar{h}\Big)^2, \qquad \bar{\hat{y}} = \frac{1}{M}\sum_{i=1}^{M}\hat{y}_i, \quad \bar{h} = \frac{1}{M}\sum_{i=1}^{M}h(\mathbf{x}_i) \tag{4.32}$$

$$\mathrm{variance}_{\mathrm{emp}} = \frac{1}{M}\sum_{i=1}^{M}(\hat{y}_i - \bar{\hat{y}})^2 \tag{4.33}$$

$$\mathrm{noise}_{\mathrm{emp}} = \frac{1}{M}\sum_{i=1}^{M}\epsilon_i^2 \tag{4.34}$$

Further reading

- Optuna: an open-source hyper-parameter optimization framework that implements random search, TPE-based Bayesian optimization, and trial pruning (optuna.org).
- J. Bergstra and Y. Bengio, *Random Search for Hyper-Parameter Optimization*, Journal of Machine Learning Research, 2012: the source of the random-versus-grid result of Sec. 4.6.

4.11. Summary

- *Overfitting and underfitting* are two extremes of model complexity: overfitting follows training data too closely, underfitting fails to capture the underlying pattern.
- *Bias-variance trade-off* provides the theoretical framework explaining why overfitting (high variance, low bias) and underfitting (high bias, low variance) occur.
- *Cross-validation* mitigates data-related issues by evaluating model performance across different data splits, and guides model-related decisions such as hyper-parameter selection.
- *Hyper-parameter tuning* (e.g., polynomial degree L, regularization λ) adjusts model complexity to balance underfitting and overfitting.
- *Normalization and standardization* are data pre-processing steps that reduce the influence of feature scale imbalance and improve numerical stability.
- *Increasing dataset size* directly reduces the data-related gap by making $\bar{J}_M$ closer to $\bar{J}_{theory}$, and reduce overfitting in complex models.
- *Regularization* constrains model complexity via the penalty parameter λ, controlling the bias-variance trade-off from the model side.

5. Kernels

The goal is to effectively apply non-linear models.

Contents

5.1. Uni-variate Polynomial Model **70**
5.2. Mapping function **71**
5.3. Kernel Trick **72**
5.3.1. LS Identity 72
5.3.2. Kernel Matrix 73
5.3.3. Handling Overfitting 73
5.3.4. Kernel-Based Model 73
5.4. Kernel Types **74**
5.4.1. Polynomial Kernel 74
5.4.2. Gaussian Radial Basis Kernel (RBK) 75
5.4.3. Summary 76
5.5. Kernel Smoothing **76**

5.1. Uni-variate Polynomial Model

The L-degree uni-variate polynomial regression model[1] is

$$\begin{aligned}\hat{y} = f_{\mathbf{w}}(x) &= w_0 + w_1 x + w_2 x^2 + \cdots + w_L x^L \\ &= \sum_{j=0}^{L} w_j x^j\end{aligned} \tag{5.1}$$

This problem is linear under the change of variables $z_j = x^j,\ j = 0, \ldots, L$,

$$\hat{y} = \sum_{j=0}^{L} w_j z_j \tag{5.2}$$

[1]The model is also presented in Sec. 4.1. The selection of L is discussed in Ch. 4.4.2.

The corresponding prediction for the dataset $\{x_k, y_k\}_{k=1}^M$ can be easily written by using the matrix notation (also termed Vandermonde matrix),

$$\mathbf{X} = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^L \\ 1 & x_2 & x_2^2 & \cdots & x_2^L \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_M & x_M^2 & \cdots & x_M^L \end{bmatrix} \tag{5.3}$$

The weights vector $\mathbf{w}$ then follows from the standard least-squares formula.

5.2. Mapping function

The data-linearization results (e.g., in (5.3)) can be formulated by a *mapping function*.

Univariate polynomial mapping The polynomial model uses a uni-variate to multi-variate mapping, $\phi(\cdot) : \mathbb{R} \to \mathbb{R}^L$,

$$\phi(x) = \begin{bmatrix} 1 & x & x^2 & \cdots & x^L \end{bmatrix} \tag{5.4}$$

Applying ϕ to each value in $\{x_k\}_{k=1}^M$ builds the matrix $\mathbf{X}$ in (5.3).

Multivariate mapping example More general multi-variate to multi-variate mappings $\phi(\cdot) : \mathbb{R}^N \to \mathbb{R}^L$ can be applied. For the regression model

$$f(\mathbf{x}; \mathbf{w}) = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2 + w_4 x_1^2 + w_5 x_2^2$$

the corresponding $N = 2$ and $L = 6$ mapping function (ellipse equation elements) is

$$\phi(\mathbf{x}) = \phi(x_1, x_2) = \begin{bmatrix} 1, x_1, x_2, x_1 x_2, x_1^2, x_2^2 \end{bmatrix} \tag{5.5}$$

that for each value in $\{x_{1_k}, x_{2_k}\}_{k=1}^M$ produces the corresponding matrix

$$\phi(\mathbf{X}) = \mathbf{X}_\phi = \begin{bmatrix} 1 & x_{1_1} & x_{2_1} & x_{1_1} x_{2_1} & x_{1_1}^2 & x_{2_1}^2 \\ 1 & x_{1_2} & x_{2_2} & x_{1_2} x_{2_2} & x_{1_2}^2 & x_{2_2}^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & x_{1_M} & x_{2_M} & x_{1_M} x_{2_M} & x_{1_M}^2 & x_{2_M}^2 \end{bmatrix} \tag{5.6}$$

Problem

Even for a relatively small N, multivariate mapping produces numerous feature vectors (large L) that are less convenient to implement. Moreover, the matrix $\mathbf{X} \in \mathbb{R}^{M \times L}$ produces memory and computationally prohibitive $\mathbf{X}^T \mathbf{X} \in \mathbb{R}^{L \times L}$ matrix.

Example of $\mathbf{X}^T\mathbf{X}$ For $L = 3$ and $\mathbf{X} = \begin{bmatrix} \tilde{\mathbf{x}}_1 & \tilde{\mathbf{x}}_2 & \tilde{\mathbf{x}}_3 \end{bmatrix} \in \mathbb{R}^{M\times 3}$ (column decomposition) the resulting $\mathbf{X}^T\mathbf{X} \in \mathbb{R}^{3\times 3}$,

$$\mathbf{X}^T\mathbf{X} = \begin{bmatrix} \tilde{\mathbf{x}}_1^T \\ \tilde{\mathbf{x}}_2^T \\ \tilde{\mathbf{x}}_3^T \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{x}}_1 & \tilde{\mathbf{x}}_2 & \tilde{\mathbf{x}}_3 \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{x}}_1^T\tilde{\mathbf{x}}_1 & \tilde{\mathbf{x}}_1^T\tilde{\mathbf{x}}_2 & \tilde{\mathbf{x}}_1^T\tilde{\mathbf{x}}_3 \\ \tilde{\mathbf{x}}_2^T\tilde{\mathbf{x}}_1 & \tilde{\mathbf{x}}_2^T\tilde{\mathbf{x}}_2 & \tilde{\mathbf{x}}_2^T\tilde{\mathbf{x}}_3 \\ \tilde{\mathbf{x}}_3^T\tilde{\mathbf{x}}_1 & \tilde{\mathbf{x}}_3^T\tilde{\mathbf{x}}_2 & \tilde{\mathbf{x}}_3^T\tilde{\mathbf{x}}_3 \end{bmatrix}, \tag{5.7}$$

where $\tilde{\mathbf{x}}_j^T\tilde{\mathbf{x}}_k \in \mathbb{R}$ is a scalar.

5.3. Kernel Trick

The goal is:

- To replace $\mathbf{X}^T\mathbf{X} \in \mathbb{R}^{L\times L}$ with $\mathbf{X}\mathbf{X}^T \in \mathbb{R}^{M\times M}$, a matrix useful in the $L > M$ case.
- Use arbitrary high L.
- Provide vector-based and more general expressions for $\phi(\mathbf{x})$.

5.3.1. LS Identity

The base of the method is an identity

$$\begin{aligned} \mathbf{w} &= \mathbf{X}^T\left(\mathbf{X}\mathbf{X}^T\right)^{-1}\mathbf{y} \\ &= \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y} \end{aligned} \tag{5.8}$$

The key property of this representation is that the $M \times M$ dimensions are independent of the number of original features N and mapped features L. The prediction

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$$

is independent of the way $\mathbf{w}$ is evaluated.

Example of $\mathbf{X}\mathbf{X}^T$ for $N = 3$ For $\mathbf{X} = \begin{bmatrix} \tilde{\mathbf{x}}_1 & \tilde{\mathbf{x}}_2 & \tilde{\mathbf{x}}_3 \end{bmatrix} \in \mathbb{R}^{M\times 3}$ (column decomposition) the resulting $\mathbf{X}\mathbf{X}^T \in \mathbb{R}^{M\times M}$,

$$\mathbf{X}\mathbf{X}^T = \begin{bmatrix} \tilde{\mathbf{x}}_1 & \tilde{\mathbf{x}}_2 & \tilde{\mathbf{x}}_3 \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{x}}_1^T \\ \tilde{\mathbf{x}}_2^T \\ \tilde{\mathbf{x}}_3^T \end{bmatrix} = \tilde{\mathbf{x}}_1\tilde{\mathbf{x}}_1^T + \tilde{\mathbf{x}}_2\tilde{\mathbf{x}}_2^T + \tilde{\mathbf{x}}_3\tilde{\mathbf{x}}_3^T \tag{5.9}$$

where each term $\tilde{\mathbf{x}}_j\tilde{\mathbf{x}}_j^T \in \mathbb{R}^{M\times M}$ is an outer product.

5.3.2. Kernel Matrix

The mapping function $\phi(\cdot) : \mathbb{R}^{M\times N} \to \mathbb{R}^{M\times L}$ is applied to the full data matrix $\mathbf{X}$ (as in (5.6))

$$\mathbf{X}_\phi = \phi(\mathbf{X}) \tag{5.10}$$

By using the identity

$$\phi\left(\mathbf{X}^T\right) = \phi\left(\mathbf{X}\right)^T \tag{5.11}$$

the resulting kernel matrix $\mathbf{K} \in \mathbb{R}^{M\times M}$ is defined as

$$\mathbf{K}(\mathbf{X},\mathbf{X}) = \phi\left(\mathbf{X}\right)\phi\left(\mathbf{X}\right)^T \tag{5.12}$$

Kernel matrix properties:

- Symmetry, $\mathbf{K} = \mathbf{K}^T$
- Positive semi-definite, $\mathbf{x}\mathbf{K}\mathbf{x}^T \geq 0$, with all eigenvalues non-negative.

5.3.3. Handling Overfitting

Large L values, e.g. in the polynomial model, result in a high potential for overfitting. Therefore, ridge regression (4.21) is applied together with a kernel trick,

$$\mathbf{w} = \mathbf{X}^T\left(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I}\right)^{-1}\mathbf{y} \tag{5.13}$$

5.3.4. Kernel-Based Model

The calculation of the kernel trick relation in Eq. (5.13) is performed by the following steps:

1. Evaluate the kernel matrix $\mathbf{K}$ using (5.12).
2. Calculate $\boldsymbol{\alpha}$,
$$\boldsymbol{\alpha} = (\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{y} \tag{5.14}$$
where $\mathbf{K}$ replaces $\mathbf{X}\mathbf{X}^T$, yielding $\boldsymbol{\alpha} \in \mathbb{R}^M$. After this step, $\mathbf{K}$ no longer needs to be stored. The weight vector $\mathbf{w} \in \mathbb{R}^L$ can then (theoretically) be recovered as
$$\mathbf{w} = \varphi\left(\mathbf{X}\right)^T\boldsymbol{\alpha} = \sum_{j=1}^{M}\alpha_j\varphi\left(\tilde{\mathbf{x}}_j\right) \tag{5.15}$$
but is not required for prediction.
3. **Model inference:** The regression for a new test point $\tilde{\mathbf{x}}_0$ is evaluated by
$$\begin{aligned}\hat{y}_0 &= \varphi(\tilde{\mathbf{x}}_0)^T\mathbf{w}\\ &= \sum_{j=1}^{M}\alpha_j\varphi(\tilde{\mathbf{x}}_0)^T\varphi\left(\tilde{\mathbf{x}}_j\right)\\ &= \varphi(\tilde{\mathbf{x}}_0)^T\varphi\left(\mathbf{X}\right)\boldsymbol{\alpha}\\ &= K\left(\tilde{\mathbf{x}}_0,\mathbf{X}\right)\boldsymbol{\alpha}\end{aligned} \tag{5.16}$$

where

$$\begin{aligned} \varphi(\tilde{\mathbf{x}}_0)^T \in \mathbb{R}^{1\times L}, \varphi(\tilde{\mathbf{x}}_0) \in \mathbb{R}^{L\times 1} \\ \varphi\left(\mathbf{X}\right)^T \in \mathbb{R}^{L\times M}, \varphi\left(\mathbf{X}\right) \in \mathbb{R}^{M\times L} \\ K\left(\tilde{\mathbf{x}}_0, \mathbf{X}\right) \in \mathbb{R}^{1\times M} \end{aligned} \tag{5.17}$$

No L-dimensional calculations are required, and $\mathbf{w}$ need not be evaluated explicitly.

In summary, the order is:

1. Kernel matrix, (5.12).
2. $\boldsymbol{\alpha}$, (5.14).
3. For test data $\tilde{\mathbf{X}}_0$, the prediction (5.16) becomes

$$\hat{\mathbf{y}}_0 = K\left(\tilde{\mathbf{X}}_0, \mathbf{X}\right)\boldsymbol{\alpha} \tag{5.18}$$

5.4. Kernel Types

5.4.1. Polynomial Kernel

$$K(\mathbf{x}, \mathbf{y}) = \left(\mathbf{x}^\mathsf{T}\mathbf{y} + c\right)^d \tag{5.19}$$

where c is some constant (optionally $c = 0$) and d is the power.

The number of mapped dimensions is $L = \binom{d+N-1}{d}$. For example, for $N = 10, d = 4, L = \binom{13}{4} = 715$.

Example for $c = 0, d = 3, L = 2$:

$$\begin{aligned} K(\mathbf{a}, \mathbf{b}) = \left(\mathbf{a}^T\mathbf{b}\right)^3 &= \left([a_1, a_2]\begin{bmatrix} b_1 \\ b_2 \end{bmatrix}\right)^3 \\ &= \left(a_1 b_1 + a_2 b_2\right)^3 \\ &= a_1^3 b_1^3 + 3a_1^2 b_1^2 a_2 b_2 + 3a_1 b_1 a_2^2 b_2^2 + a_2^3 b_2^3 \\ &= \left[a_1^3, \sqrt{3}a_1^2 a_2, \sqrt{3}a_1 a_2^2, a_2^3\right] \cdot \left[b_1^3, \sqrt{3}b_1^2 b_2, \sqrt{3}b_1 b_2^2, b_2^3\right]^T \\ &= \varphi(\mathbf{a})^T \varphi(\mathbf{b}) \end{aligned}$$

Example for $c = 0, d = 2$ and arbitrary L:

$$\begin{aligned} K(\mathbf{a}, \mathbf{b}) &= \left(\mathbf{a}^T\mathbf{b} + 1\right)^2 \\ &= \left(\mathbf{a}^T\mathbf{b}\right)^2 + 2\mathbf{a}^T\mathbf{b} + 1 \end{aligned} \tag{5.20}$$

$$
\begin{aligned}
&= \left(\sum_{i=1}^{L} a_i b_i\right)^2 + 2\sum_{i=1}^{L} a_i b_i + 1 \\
&= \sum_{i=1}^{L} a_i b_i \sum_{j=1}^{L} a_j b_j + 2\sum_{i=1}^{L} a_i b_i + 1 \\
&= \sum_{i=1}^{L} a_i^2 b_i^2 + 2\sum_{i=1}^{L}\sum_{j=i+1}^{L} a_i b_i a_j b_j + 2\sum_{i=1}^{L} a_i b_i + 1 \\
&= \varphi(\mathbf{a})^T \varphi(\mathbf{b})
\end{aligned}
$$

That corresponds to the mapping functions

$$
\begin{aligned}
\varphi(\mathbf{a}) = \langle & 1, \sqrt{2}a_1, \sqrt{2}a_2, \ldots, \sqrt{2}a_L, a_1^2, a_2^2, \ldots, a_L^2, \\
& \sqrt{2}a_1a_2, \sqrt{2}a_1a_3, \ldots, \sqrt{2}a_1a_L, \sqrt{2}a_2a_3, \ldots, \sqrt{2}a_{L-1}a_L\rangle \\
\varphi(\mathbf{b}) = \langle & \underbrace{1}_{1}, \underbrace{\sqrt{2}b_1, \sqrt{2}b_2, \ldots, \sqrt{2}b_L}_{2\sum_{i=1}^{L} a_i b_i}, \underbrace{b_1^2, b_2^2, \ldots, b_L^2}_{\sum_{i=1}^{L} a_i^2 b_i^2}, \\
& \underbrace{\sqrt{2}b_1b_2, \sqrt{2}b_1b_3, \ldots, \sqrt{2}b_1b_L, \sqrt{2}b_2b_3, \ldots, \sqrt{2}b_{L-1}b_L}_{2\sum_{i=1}^{L}\sum_{j=i+1}^{L} a_i b_i a_j b_j}\rangle
\end{aligned}
$$

with all the combinations above.

5.4.2. Gaussian Radial Basis Kernel (RBK)

The kernel definition is

$$
K(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|^2}{2b}\right) \tag{5.21}
$$

Due to Taylor expansion,

$$
\exp(x) \cong 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \cdots
$$

this kernel has $L \to \infty$.

Typically,

- Too low b, when $b \to 0$
  - Kernel becomes δ-function-like.
  - Each training point only affects neighbors
  - Overfitting (high variance, low bias).
- Too high b, when $b \to \infty$
  - Kernel becomes constant, $\mathbf{K}_{ij} \approx 1$.
  - All training points affect everywhere
  - Underfitting (low variance, high bias)

The numerical example (same dataset as in Fig. 4.13) is presented in Fig. 5.1.

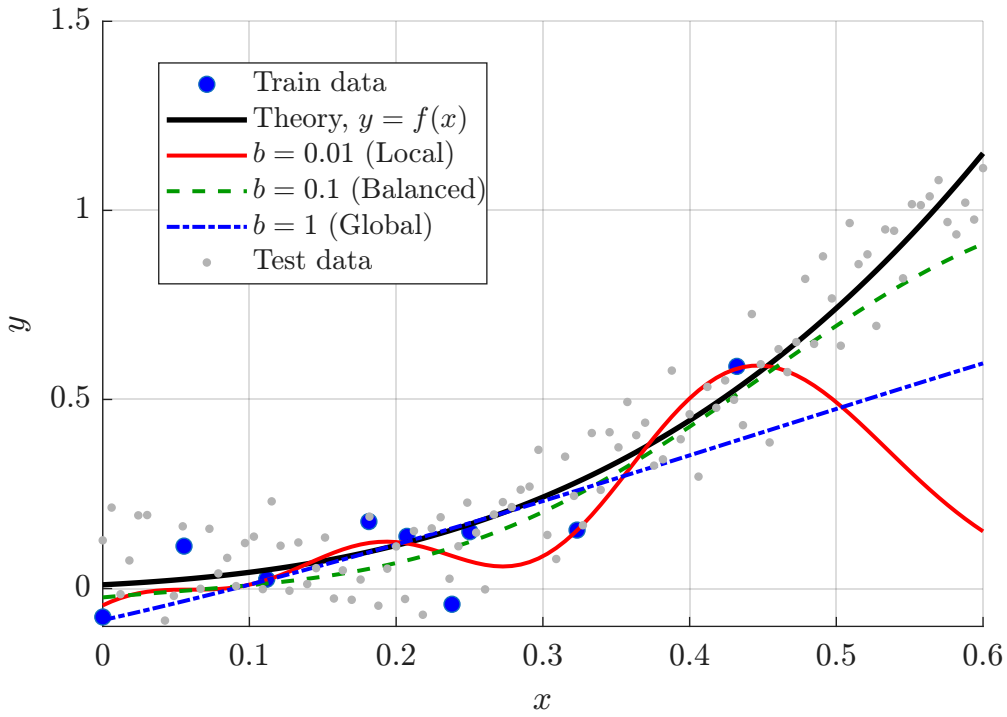


Figure 5.1.: Kernel regression example with polynomial kernel. Direct comparison of RBF kernel regression with three bandwidth values. The medium bandwidth ($b = 0.1$, green) achieves the best balance between fitting the data and maintaining smoothness.

5.4.3. Summary

- Using virtually high L.
- Require regularization parameter (λ) tuning.
- Kernel type is hyper-parameter, and each kernel has its own hyper-parameters.
- Somewhat limited by $M \times M$ kernel matrix size.
- Since the values of $\mathbf{w}$ are not evaluated explicitly, the influence of each $\tilde{\mathbf{x}}_j$ and inter-data $\tilde{\mathbf{x}}_j, \tilde{\mathbf{x}}_k$ relations are unknown.

5.5. Kernel Smoothing

Nadaraya–Watson smoothing For the $\tilde{\mathbf{x}}_0$ of interest, the kernel smoothing is weighted averaging given by

$$\hat{y}_0 = \sum_{j=1}^{M} w_j y_j \tag{5.22}$$

where the weights are

$$\tilde{\mathbf{w}} = K(\tilde{\mathbf{x}}_0, \mathbf{X}) \tag{5.23}$$

with normalization

$$w_j = \frac{\tilde{w}_j}{\sum_{j=1}^{M} \tilde{w}_j} \tag{5.24}$$

Finally, for a test dataset, $\mathbf{X}_{\text{test}}$ (vector notation)

$$\tilde{\mathbf{W}} = K(\mathbf{X}_{\text{test}}, \mathbf{X}_{\text{train}}) \qquad \in \mathbb{R}^{M_{\text{test}} \times M_{\text{train}}} \tag{5.25}$$

$$\mathbf{W} = \tilde{\mathbf{W}} / \tilde{\mathbf{W}} \mathbf{1}_{M_{\text{train}}} \tag{5.26}$$

$$\hat{\mathbf{y}}_{\text{test}} = \mathbf{W} \mathbf{y}_{\text{train}} \tag{5.27}$$

The numerical example is presented in Fig. 5.2.

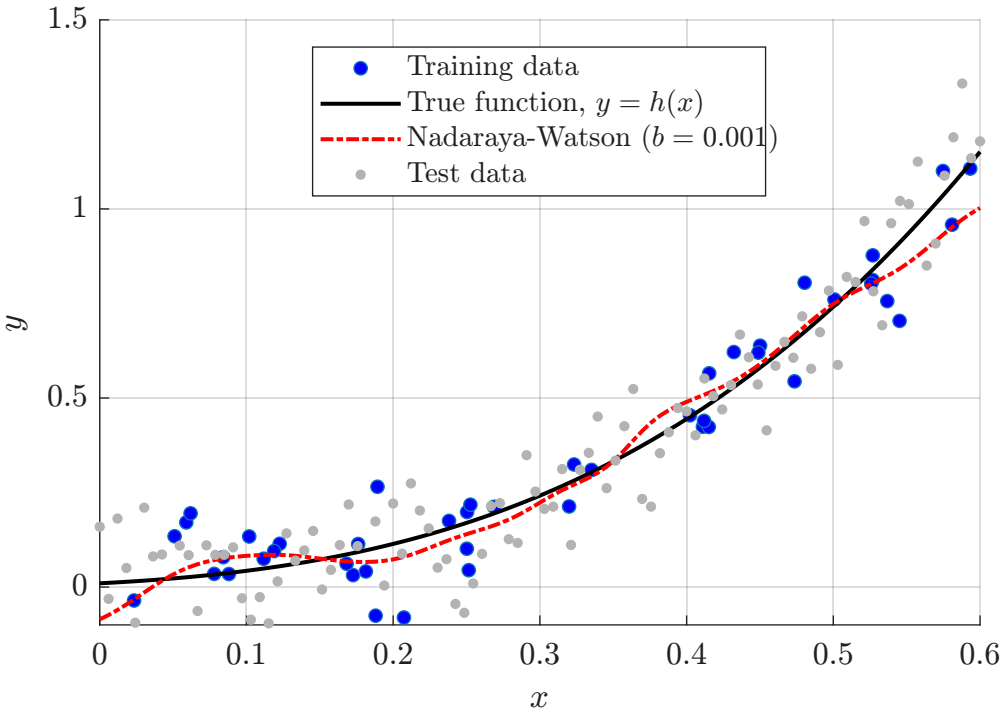


Figure 5.2.: Kernel smoothing example with exponential kernel. Incorrect selection of b parameters results in overfitting. The kernel smoother provides a non-parametric estimate by computing locally weighted averages of nearby training points, where weights decay exponentially with distance.

Summary

- Low numerical complexity.
- Kernel type is hyper-parameter, and each kernel has its own hyper-parameters. No additional method hyper-parameters are required.
- Non-trivial hyper-parameter optimization on multivariate datasets.
- Require sufficiently dense dataset.
- Capture nonlinear relationships without assuming a specific parametric form.

6. Regression Losses and Metrics

Goal: Quantify the difference between predicted and target values.

For regression, loss and metrics are often the same quantity.

Contents

6.1. Preface **78**
6.2. Loss Function Properties **79**
6.3. Losses **80**
6.3.1. Mean-squared error (MSE) or L_2-loss 80
6.3.2. RMSE 81
6.3.3. Mean absolute error (MAE) 81
6.3.4. Special Losses (*) 84
6.3.5. Mean Squared Logarithmic Error (MSLE) 85
6.4. Relative Metrics **85**
6.5. Information Criteria **87**
6.5.1. Akaike's Final Prediction Error (FPE) 87
6.5.2. Akaike's Information Criterion (AIC) 87
6.5.3. Bayesian Information Criterion (BIC) 87
6.6. Summary **88**
Further Reading **88**

6.1. Preface

The predicted range of $\hat{y}_i = f(\mathbf{x}_i)$ results from the particular applied model.

Metric A performance metric is a real-valued function $J : \mathbf{y}, \hat{\mathbf{y}} \to \mathscr{R}$ that is used to evaluate and quantify the effectiveness of a model. These metrics provide insight into how well a model is performing according to various aspects of the data it predicts.

Loss function The loss function is a metric of the form $L : \mathbf{y}, \hat{\mathbf{y}} \to \mathscr{R}$ that is calculated over the test set, such that optimal parameters that correspond to the minimum loss

$$\mathbf{w} = \arg\min_{\mathbf{w}} \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) \tag{6.1}$$

can be evaluated, e.g., by GD (Sec. 3.4).

A metric is not necessarily a loss function, and a loss function is not necessarily a metric.

For example cross-entropy loss in classification is not a metric and R-square is not a loss.

Summary Metrics are for communication; losses are for training. A *loss* is the objective minimized during training, while a *metric* is reported to summarize performance. They may coincide (e.g., MSE as both loss and metric), but need not.

6.2. Loss Function Properties

A loss function has a few desired properties that are presented below. While not obligatory, these properties may significantly ease the evaluation of the parameters $\mathbf{w}$. In the following, only the basic description of these properties is provided, sacrificing mathematical rigor for brevity.

Continuity: Single unbroken (without jumps) curve for all possible input values, i.e., without discontinuities.

Lipschitz continuity: A formal stricter continuity requirement that limits the pace of function changes. A real-valued function $f(\cdot) : \mathcal{R} \to \mathcal{R}$ is called Lipschitz continuous if there exists a positive real constant K such that, for all real x_1 and x_2,

$$\left|f(x_1) - f(x_2)\right| \leq K|x_1 - x_2|. \tag{6.2}$$

This feature formally limits the maximum gradient values of a loss function.

Differentiability: A differentiable function of one real variable is a function whose derivative exists at each point in its domain.

- If the function is differentiable, it is also continuous.
- If the derivative is a continuous function, it is also locally Lipschitz continuous.

Convexity and strict convexity: Each segment lies above the graph between two points (Fig. 6.1). Formally, it means that the line segment between $\big(x_1, f(x_1)\big)$ and $\big(x_2, f(x_2)\big)$ is always above or just meeting the graph of the function $f(x), x_1 \leq x \leq x_2$. Mathematically, for all $0 \leq t \leq 1$ and $\forall x_1, x_2 \in \mathbb{R}$,

$$f\left(tx_1 + (1-t)x_2\right) \leq tf(x_1) + (1-t)f(x_2) \tag{6.3}$$

A twice-differentiable function of a single variable is convex if and only if its *second derivative is nonnegative* on its entire domain. For a strict convexity, the second derivative is always positive. In the context of loss function properties, convexity guarantees that any local minimum is also a global minimum. A strictly convex function has at most one global minimum.

All these properties are particularly important in GD-related methods.

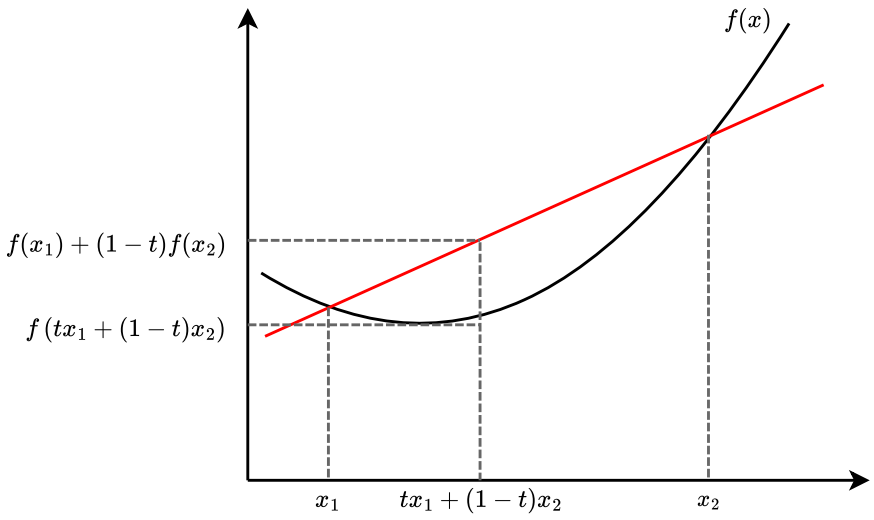


Figure 6.1.: Convexity property visualization.

6.3. Losses

Per-sample and vector notation error notation are,

$$e_i = y_i - \hat{y}_i \quad i = 1, \ldots, M$$
$$\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}}$$

Note, the order of subtraction is sometimes important in the context of this section.

Some of the following losses are also used as metrics.

6.3.1. Mean-squared error (MSE) or L_2-loss

- Continuous, differentiable, convex

MSE loss (also termed L_2-loss) and metric is ((2.5))

$$\begin{aligned} J(\mathbf{y}, \hat{\mathbf{y}}) &= \frac{1}{M} \|\mathbf{y} - \hat{\mathbf{y}}\|^2 = \frac{1}{M} \mathbf{e}^T \mathbf{e} \\ &= \frac{1}{M} \sum_{i=1}^{M} e_i^2 \end{aligned} \tag{6.4}$$

When used as loss, sometimes sum-of-squared error (SSE) ((2.4)) or the version with factor $\frac{1}{2M}$ is applied to "compensate" for the derivative are also used (see also Sec. 2.1.1).

Important properties:

- Popular regression loss.
- Popular metric.
- Analytical gradient that is error-dependent.
- Sometimes, analytical solution is available, e.g. normal equation.
- The main drawback is inherent outlier sensitivity.

6.3.2. RMSE

- Continuous, differentiable, convex

Complementary convenient metric for MSE is root-MSE (RMSE).

$$\begin{aligned} J(\mathbf{y}, \hat{\mathbf{y}}) &= \frac{1}{\sqrt{M}} \|\mathbf{y} - \hat{\mathbf{y}}\| \\ &= \sqrt{\frac{1}{M} \sum_{i=1}^{M} e_i^2} \end{aligned} \tag{6.5}$$

- Theoretically, RMSE can also be used as loss, but it is very similar to MSE with only a difference in gradients.
- Easier human interpretation.

6.3.3. Mean absolute error (MAE)

MAE is used both as loss and metric.

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{M} \sum_{i=1}^{M} |e_i| \tag{6.6}$$

Important properties:

- Popular loss and metric.
- All errors are similarly important and, therefore, less sensitive to outliers than MSE.
- Error-independent gradient that may result in *slower convergence* under certain conditions. In particular, the gradient is high even for very small errors. To fix this, a dynamic learning rate that decreases as we move closer to the minima is required. MSE behaves nicely in this case and will converge even with a fixed learning rate. The gradient of MSE loss is high for larger loss values and decreases as loss approaches 0, making it more precise at the end of training.
- Non-differentiable at $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = 0$, also without dramatic influence on most learning algorithms.

A brief numerical example of MSE, RMSE and MAE is presented in Table 6.1 and in Fig. 6.2.

Table 6.1.: Example of MSE, RMSE and MAE. Small change in error results in significant change in MSE and less dramatic change in MAE.

(y_1, y_2)	$(\hat{y}_1, \hat{y}_2)$	MSE	RMSE	MAE
(**40**,30)	(30,25)	$\frac{1}{2}\left[(40-30)^2+(30-25)^2\right]=62.5$	$\sqrt{62.5} = 7.91$	$\frac{1}{2}\left[\lvert 40-30\rvert+\lvert 30-25\rvert\right]=7.5$
(**50**,30)	(30,25)	$\frac{1}{2}\left[(50-30)^2+(30-25)^2\right]=212.5$	$\sqrt{212.5} = 14.6$	$\frac{1}{2}\left[\lvert 50-30\rvert+\lvert 30-25\rvert\right]=12.5$

The visual comparison of a few regression losses is presented in Fig. 6.3 and detailed in Sec. 6.3.4.

Figure 6.2.: The model is $N = 4$ polynomial with 40 inliers and three outliers at significantly higher values. Both MSE and MAE were used with λ-optimized L_2-regularization (Sec. 4.9). MSE regression is pulled toward outliers due to squaring large errors, while MAE regression provides more robust fitting.

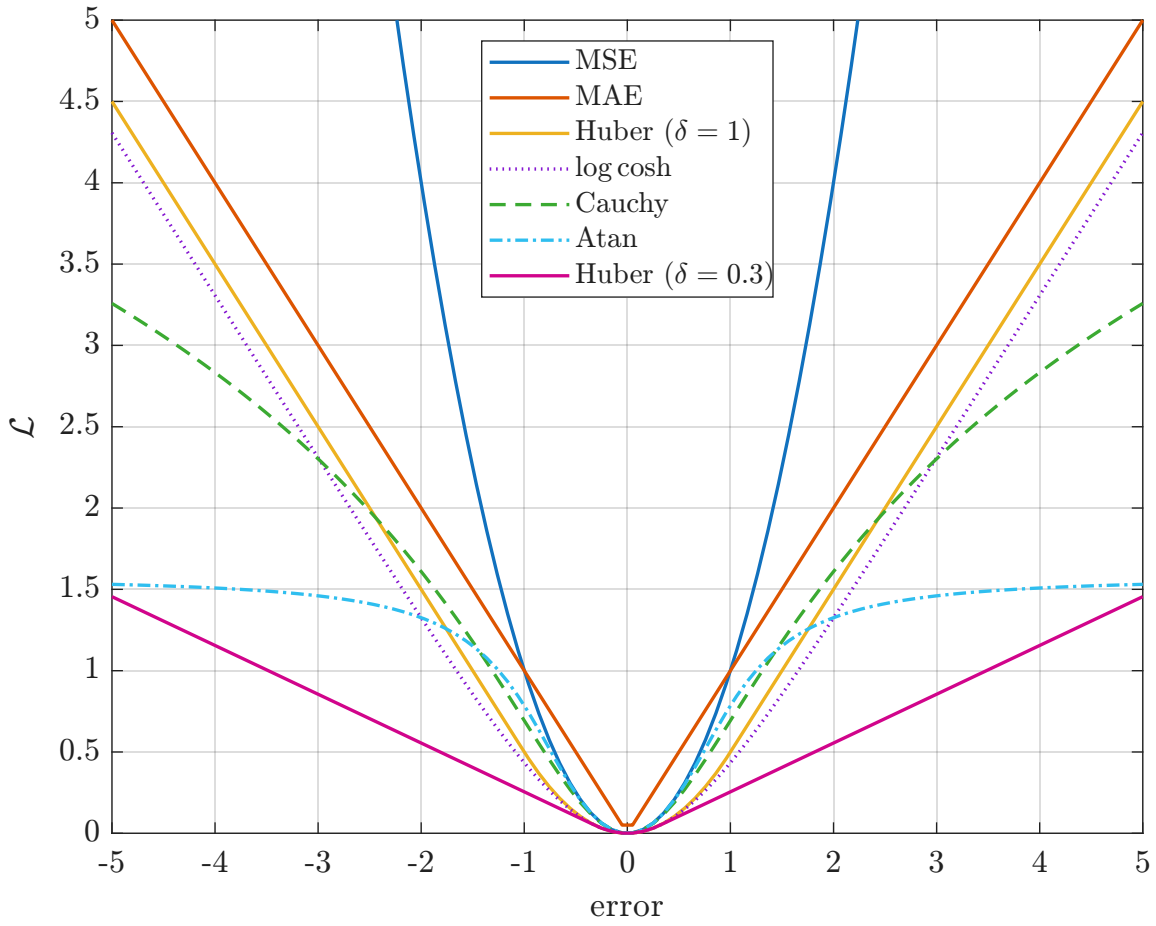


Figure 6.3.: Visualization of different loss functions. MSE is the steepest.

6.3.4. Special Losses (*)

Huber loss

- Lipschitz continuous, differentiable, convex

Goal: Hybrid between MAE and MSE.

$$\mathcal{L}(e_i) = \begin{cases} \frac{1}{2}e_i^2 & |e_i| \leq \delta \\ \delta\left(|e_i| - \frac{1}{2}\delta\right) & \text{otherwise} \end{cases} \tag{6.7}$$

For small e_i it behaves like MSE and for larger e_i like MAE.

The problem with Huber loss is the need to tune the hyperparameter δ, which is a non-trivial process.

Log-cosh loss

Goal: Hybrid between MAE and MSE without hyper-parameters.

$$\mathcal{L}(e_i) = \log\cosh e_i \tag{6.8}$$

Properties:

- Twice differentiable everywhere, $\frac{\partial}{\partial e_i} L(e_i) = \tanh e_i$.
- Approximation:

$$L(e_i) \approx \begin{cases} \frac{e_i^2}{2} & e_i \text{ small} \\ |e_i| - \log(2) & e_i \text{ large} \end{cases} \tag{6.9}$$

- No hyper-parameters.
- Similar to Huber loss with $\delta = 1$.
- Requires non-trivial error handling, otherwise may get stuck in either of two regions.
- Explicit hyper-parameter optimization of Huber loss is recommended.

Cauchy

Goal: More robust to outliers than MAE.

$$\mathcal{L}(e_i) = \log\left(1 + \left(\frac{e_i}{d}\right)^2\right) \tag{6.10}$$

- d is "sharpness" parameter
- Robust against outliers more than MAE but less than `atan` loss.

Atan

Goal: More robust to outliers than Cauchy.

$$\mathcal{L}(e_i) = \arctan\left(e_i^2\right) \tag{6.11}$$

- Atan tends to $\pi/2$ as input tends to infinity, and its derivatives tend to 0.
- This means extreme outliers will have a negligible effect on the search direction compared to non-outliers.
- It is important to ensure the data is well scaled and that the starting point is a reasonable guess at the true solution, if possible (similar to `Log-cosh` loss).

6.3.5. Mean Squared Logarithmic Error (MSLE)

Goal: Relative loss for high-dynamic range data.

MSLE is the relative difference between the log-transformed actual and predicted values.

$$\begin{aligned}\mathcal{L}(y_i, \hat{y}_i) &= \frac{1}{M}\sum_{i=1}^{M}\left(\log(y_i+1) - \log(\hat{y}_i+1)\right)^2 \\ &= \frac{1}{M}\sum_{i=1}^{M}\log\left(\frac{y_i+1}{\hat{y}_i+1}\right)^2\end{aligned} \tag{6.12}$$

'1' is added to both y and $\hat{y}$ for mathematical convenience since $\log(0)$ is not defined but both y and $\hat{y}$ can be 0.

- Addresses y with high dynamic range, i.e. addresses relative error. Less useful for low dynamic range.
- MSLE tries to treat small and large differences between the actual and predicted values similarly, e.g. in Table 6.2.
- Penalizes underestimated values more than overestimated values.
- Root MSLE (RMSLE) is also used, e.g. in `scikit`.

Table 6.2.: Example of MSLE

y	$\hat{y}$	MSE Loss	MSLE Loss
40	30	100	0.0782
4000	3000	$100 \times \mathbf{10{,}000}$	0.0827
20	10	100	**0.4181**
20	30	100	**0.1517**

6.4. Relative Metrics

Goal:
- Add human tractability to the standard metrics.

- Used to compare different models.
- Most of the metrics are normalized to the range of $[0, 1]$.

Relative squared error (RSE)

Normalized MSE loss.

$$\begin{aligned} J(y_i, \hat{y}_i) &= \frac{\text{MSE}}{s_{\mathbf{y}}^2} \\ &= \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2} = \frac{\|\mathbf{y} - \hat{\mathbf{y}}\|^2}{\|\mathbf{y} - \bar{\mathbf{y}}\|^2} \end{aligned} \tag{6.13}$$

Recall, that $\hat{y}_i = \bar{y}$ results MSE $= s_{\mathbf{y}}^2$. The loss shows the fraction of the explained variance, with closer to 0 is better and > 1 is abnormal.

R^2

The common metric in social sciences,

$$R^2 = 1 - RSE \tag{6.14}$$

Opposite to RSE, i.e., R^2 close to 1 is better. In highly undesirable cases, it may be negative.

For example, R^2 of 40% indicates that your model has reduced the mean squared error by 40% compared to the baseline, which is the mean model. This is the same as RSE of 60%.

For additional aspects, see also Sec. 2.3.1.

Normalized Root Mean Squared Error

Expressed as a percentage, defined as:

$$J(y_i, \hat{y}_i) = 100 \left(1 - \frac{\text{RMSE}}{s_{\mathbf{y}}}\right) = 100 \left(1 - \sqrt{RSE}\right) = 100 \left(1 - \frac{\|\mathbf{y} - \hat{\mathbf{y}}\|}{\|\mathbf{y} - \bar{\mathbf{y}}\|}\right) \tag{6.15}$$

Relative absolute error (RAE)

Normalized MAE loss.

$$J(y_i, \hat{y}_i) = \frac{\sum_i |y_i - \hat{y}_i|}{\sum_i |y_i - \overline{y}|} = \frac{MAE}{\frac{1}{M} \sum_i |y_i - \overline{y}|} \tag{6.16}$$

Recall, that $\hat{y}_i = \bar{y}$ results in $MAE = \frac{1}{M} \sum_i |y_i - \overline{y}|$. Closer to 0 is better and > 1 is abnormal.

Mean Absolute Percentage Error (MAPE)

Scaled error metric.

$$J = \frac{1}{M} \sum_{i=1}^{M} \frac{|y_i - \hat{y}_i|}{y_i} \times 100\% \tag{6.17}$$

- Beware of small denominator!
- Can exceed 100%.
- Asymmetric, as described in Table 6.3.

Table 6.3.: Example of MAPE

y	$\hat{y}$	Absolute Error	MAPE
100	60	40	40%
20	60	40	**300%**

6.5. Information Criteria

Goal: Compare models of different complexity by penalizing the number of parameters, using MSE as the goodness-of-fit term.

Adding parameters always improves the training MSE, which encourages overfitting. Information criteria replace the raw MSE with a penalized score that balances goodness-of-fit against the *number of parameters* N, given M data points. All criteria below assume that the residuals are approximately Gaussian.

6.5.1. Akaike's Final Prediction Error (FPE)

FPE estimates the expected prediction error on new data by scaling the training MSE with a bias-correction factor:

$$FPE = \text{MSE}\frac{1 + N/M}{1 - N/M} = \text{MSE}\frac{M + N}{M - N} \tag{6.18}$$

6.5.2. Akaike's Information Criterion (AIC)

AIC trades off goodness-of-fit against complexity. Lower AIC indicates a better model:

$$\Delta AIC = 2N + M \ln(\text{MSE}) \tag{6.19}$$

AICc adds a small-sample correction, recommended when $M/N < 40$:

$$AICc = AIC + 2N\frac{N + 1}{M - N - 1} \tag{6.20}$$

6.5.3. Bayesian Information Criterion (BIC)

BIC applies a stronger penalty on the number of parameters, especially as the sample size M grows, so it tends to favor simpler models than AIC:

$$BIC = M \ln(\text{MSE}) + N \ln(M) \tag{6.21}$$

These criteria are computed directly from residuals and parameter count without any additional model fitting, and they enable principled comparison between models of different complexity. Their main limitation is the Gaussian-residual assumption, which can be misleading when violated; AIC and BIC can also disagree about which model is best, so no single criterion is universally preferred. These criteria are applied to time-series model-order selection in Sec. 24.3.

6.6. Summary

Common Pitfalls:

- **Scale confusion**: MSE/RMSE/MAE are scale-dependent; compare across datasets only after normalization or via dimensionless metrics.
- **MAPE near zero**: Avoid MAPE when y can be small or cross zero.

Further Reading

- Alexei Botchkarev, "Performance metrics (error measures) in machine learning regression, forecasting and prognostics: Properties and typology" *arXiv preprint arXiv:1809.03006*, 2018. https://arxiv.org/abs/1809.03006.
- Ciampiconi, L., et al. "A survey and taxonomy of loss functions in machine learning; A survey and taxonomy of loss functions in machine learning." 10.3390/ai7040128.

7. Learning Systems (Mid-Summary)

Contents

7.1. Basic Workflow **89**
7.2. Goal Types **90**
7.3. Model and Parameters **92**
7.3.1. Parameters vs Hyper-parameters 92
7.3.2. Parametric vs Non-parametric Models 92
7.4. Loss and Metrics: Optimization Approaches **92**
7.5. Project Steps **93**
7.6. MLOps **94**

Goal: Consolidate the concepts introduced in the previous chapters into a single end-to-end view of an ML/DL system: workflow, goal types, model and loss recap, project process (CRISP-DM), and operational deployment (MLOps).

7.1. Basic Workflow

The basic ML/DL workflow is presented in Fig. 7.1. Following the data-flow in the figure, its building blocks are:

- *Goal definition*: what the system is intended to predict or decide.
- *Raw data*: the available observations before any treatment.
- *Pre-processing*: preliminary exploration and validation of dataset integrity (e.g., consistent physical units across all values of the same feature), producing the dataset $(\mathbf{X}, \mathbf{y})$.
- *Model*: assumptions about the hidden pattern within the data, $\hat{\mathbf{y}} = f(\mathbf{X}; \mathbf{w})$, parameterized by $\mathbf{w}$.
- *Loss*: the quantity $\mathcal{L}(\hat{\mathbf{y}}, \mathbf{y})$ to be minimized.
- *Solver/optimizer*: the training procedure that adjusts $\mathbf{w}$ so as to minimize the loss.
- *Hyper-parameter optimization*: outer loop that tunes the model sub-type, typically via cross-validation.
- *Performance metrics*: human-facing indicators used for assessment.
- *Generalization performance*: the final estimate of model quality on unseen data.

Baseline: The basic end-to-end implementation of the workflow is called the *baseline*.

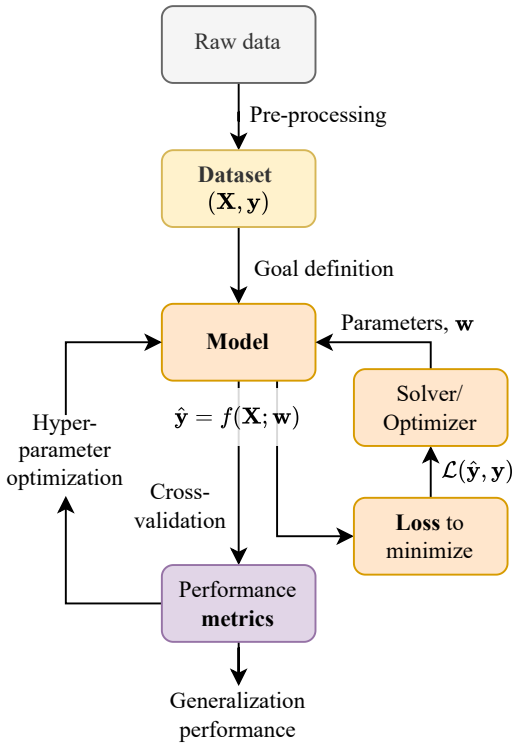


Figure 7.1.: Basic workflow of an ML/DL solution.

7.2. Goal Types

The goal determines both the form of the target $\mathbf{y}$ and the family of admissible models, losses, and metrics. The most common ML goal types are listed below and illustrated in Fig. 7.2.

Supervised The dataset provides paired inputs and targets $(\mathbf{X}, \mathbf{y})$.

- *Regression*: $\mathbf{y}$ is quantitative; the model predicts a continuous value (Fig. 7.2a).
- *Classification*: $\mathbf{y}$ is categorical; the model assigns each sample to one of finitely many classes (Fig. 7.2b).
- *Segmentation*: classification or regression performed per pixel, per sample, or per time index, producing a structured output of the same shape as the input.

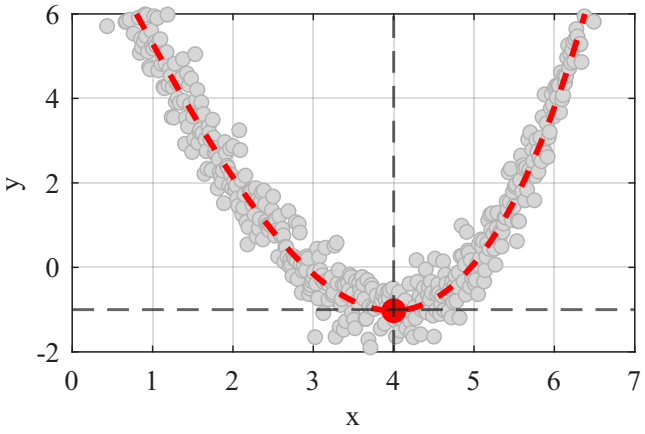
Unsupervised No target $\mathbf{y}$ is provided; structure is inferred from $\mathbf{X}$ alone.

- *Clustering*: group samples by similarity (Fig. 7.2c).
- *Anomaly detection*: identify samples that deviate from the typical pattern; situated between classification and clustering (Fig. 7.2e).
- *Simulation*: learn the underlying distribution to generate new, statistically similar samples.

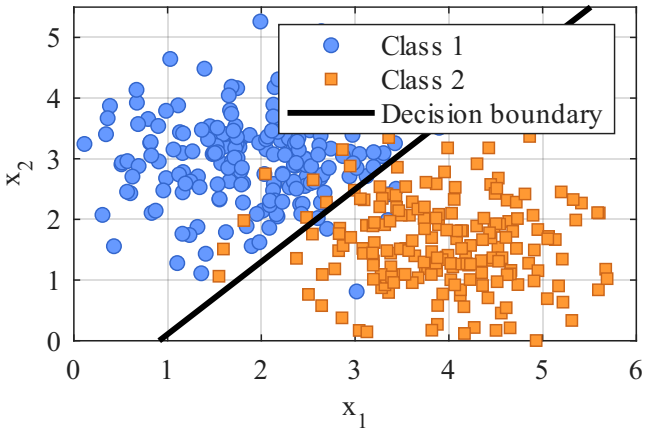
Mixed and signal-oriented

- *Semi-supervised learning*: a small set of labeled samples guides learning from a much larger pool of unlabeled ones (Fig. 7.2d).

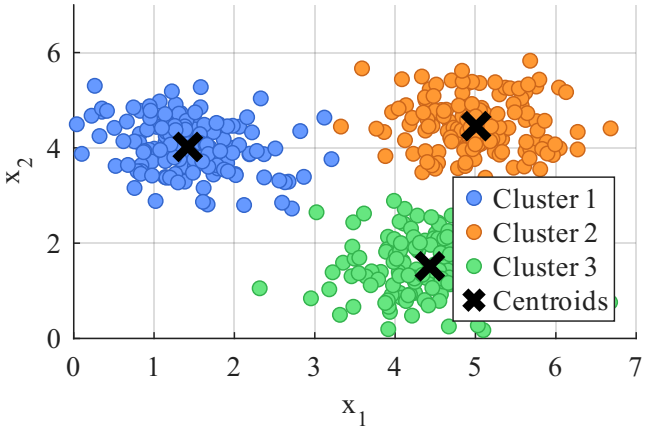
- *Signal processing tasks*: noise removal, smoothing or filling missing values, and event/condition detection.



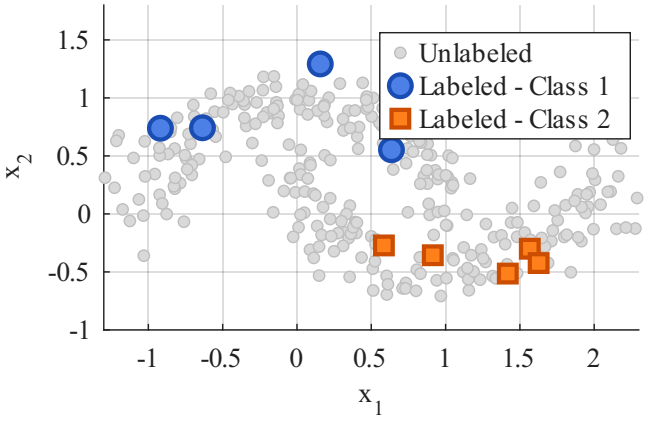
(a) Regression: what is the value of y for given x?



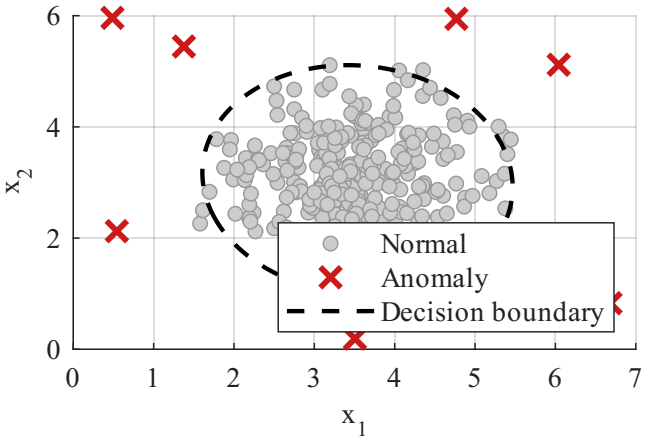
(b) Classification: what is the class of a new sample?



(c) Clustering: which samples belong together?



(d) Semi-supervised: few labeled samples guide learning from many unlabeled ones.



(e) Anomaly detection: which samples deviate from the normal pattern?

Figure 7.2.: Examples of common ML goal types.

A single dataset may serve two or more goals simultaneously.

7.3. Model and Parameters

Recall, see 2 and 4.2, that the assumed data-generating model is

$$y = h(\mathbf{x}) + \epsilon, \tag{7.1}$$

where $h(\mathbf{x})$ is the unknown true function and ϵ is irreducible (often zero-mean) noise. The model $f(\cdot; \mathbf{w})$, parameterized by $\mathbf{w}$, approximates $h(\mathbf{x})$ and produces predictions $\hat{y}_0 = f(\mathbf{x}_0; \mathbf{w})$ for new inputs $\mathbf{x}_0$. The process of finding $\mathbf{w}$ from the dataset is called *learning* or *training*.

7.3.1. Parameters vs Hyper-parameters

Model parameters: Model parameters $\mathbf{w}$ are learned directly from the dataset by minimizing the loss function.

Hyper-parameters: Hyper-parameters are model parameters that are not learned directly from the dataset; they are set by the designer or chosen indirectly via cross-validation (see 4.2).

Hyper-parameter optimization Selecting the most appropriate hyper-parameter values is called hyper-parameter optimization.

7.3.2. Parametric vs Non-parametric Models

There are two main classes of models, parametric and non-parametric, summarized in Table 7.1. The modern trend is to bridge the gap between interpretable and non-parametric modeling.

7.4. Loss and Metrics: Optimization Approaches

The loss (or cost) function $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$ relates dataset outputs $\mathbf{y}$ to model outputs $\hat{\mathbf{y}}$, and the parameters $\mathbf{w}$ are obtained as

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}), \tag{7.2}$$

a process termed *training* (see 2.1.1 for definitions and examples such as SSE, MSE, RMSE). A *metric* is the corresponding human-facing performance indicator; sometimes the loss minimum and the reported metric coincide (e.g. MSE).

Closed-form solution A closed-form solution for $\mathbf{w}$ is expressible through standard mathematical operations. For example, the "normal equation" provides such a solution for linear regression.

Table 7.1.: Comparison of parametric and non-parametric models.

Aspect	Parametric	Non-parametric
Dependence of the number of parameters on dataset size	Fixed	Flexible
Interpretability	Yes	No
Underlying data assumptions	Yes	No
Risk	Underfitting due to rigid structure	Overfitting due to high flexibility
Dataset size	Smaller	Best for larger
Complexity	Often fast	Often complex
Examples	Linear regression	k-NN, trees

Local-minimum gradient-based iterative algorithms This family of algorithms is applicable only for convex (preferably strictly convex) loss functions. Examples include gradient descent (GD) and its modifications (e.g., stochastic GD), used to evaluate NN parameters, and the Newton-Raphson algorithm.

- Some advanced algorithms in this category employ (or require) the second-order derivative $\frac{\partial^2}{\partial \mathbf{w}} \mathcal{L}$ for faster convergence.
- If a derivative is not available in closed form, it is evaluated numerically.

Global optimizers Global optimizers aim to find a global minimum of a non-convex function. They may be gradient-free, first-derivative, or second-derivative based. Their complexity is significantly higher than that of local optimizers and can be prohibitive for more than a few hundred variables in $\mathbf{X}$.

7.5. Project Steps

The workflow of Fig. 7.1 captures the technical core of an ML/DL solution but says little about the surrounding project. A widely adopted process framework that does is the Cross-Industry Standard Process for Data Mining (CRISP-DM), which organizes a project into six *iterative* phases: findings in any phase may require revisiting earlier ones. Each phase is described below together with practical guidelines for executing it well.

1. **Business understanding**: define the problem, success criteria, and project plan. Domain knowledge matters most in this phase.
    - Translate the business need into a concrete ML goal type and a single, measurable success criterion.
    - Choose the performance metric and the numerical performance goal in advance, before looking at any model output.
    - Estimate human-level performance, if relevant, as a reference point for what is achievable.

- Identify constraints early: data access, latency, interpretability, and regulatory or safety requirements.

2. **Data understanding**: collect initial data and perform exploratory analysis to learn what the dataset actually contains.
   - Compute basic statistics (mean, variance, ranges, missingness) and create visualizations for every feature and the target.
   - Verify that the sample is representative of the deployment distribution and that the dataset size is sufficient for the chosen model family.
   - Flag outliers, label noise, and class imbalance early; these shape later decisions on loss and metric.
3. **Data preparation**: clean, transform, and construct the final dataset. This is typically the most time-consuming phase.
   - Validate dataset integrity (consistent units, deduplication, handling of missing values).
   - Apply normalization or standardization where appropriate (Sec. 4.2).
   - Construct features and split the data into training, validation, and test sets in a way that reflects the deployment scenario (e.g., temporal split for time-series).
4. **Modeling**: select and train candidate models.
   - Start with a standard, pre-implemented model; avoid overly complex baselines.
   - Build an end-to-end pipeline data $\rightarrow$ model $\rightarrow$ loss $\rightarrow$ metrics, and debug it until the metric value is sane on a known-easy slice of data.
   - Only after the baseline is healthy, introduce more expressive models or richer features.
5. **Evaluation**: assess model performance against the business objectives, not only against the loss.
   - Compare the metric to the goal set in phase 1 and to the human-level reference.
   - Diagnose overfitting and underfitting via bias/variance analysis (Sec. 4.2).
   - Investigate the largest errors qualitatively; they often reveal dataset-integrity problems or unmodeled regimes.
   - Improve performance iteratively by tuning hyper-parameters, adding data, or refining the model. If the goal remains out of reach, revisit phase 1 rather than over-tuning.
6. **Deployment**: integrate the model into the target application. From this point on, the system is governed by the MLOps practices of the next section: serving, monitoring, retraining triggers, and versioning of both data and model.

7.6. MLOps

Goal: Automate the ML lifecycle from development to production.

ML projects differ from traditional software: outputs are probabilistic, development is experimental, and up to 80% of the effort may go into data preparation. ML operations (MLOps) extends software development operations (DevOps) principles to address these challenges, bridging model development and production deployment (Fig. 7.3).

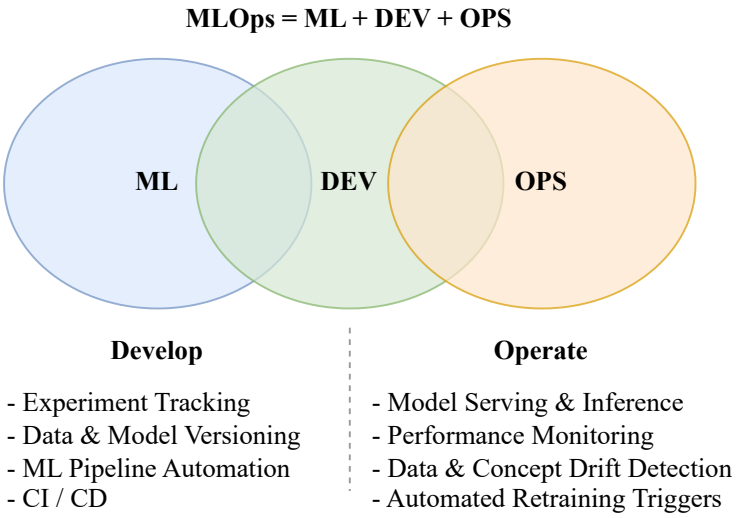


Figure 7.3.: MLOps: integrating ML, software development (DEV), and operations (OPS) into a unified workflow.

CI/CD Continuous Integration (CI) automatically builds and tests code changes upon each commit. Continuous Delivery (CD) extends CI by automating the release of validated changes to production. In MLOps, CI/CD pipelines additionally handle data validation, model training, and model deployment.

Key practices

- *Experiment tracking*: logging parameters, metrics, and artifacts for reproducibility.
- *Data and model versioning*: tracking dataset and model changes over time.
- *Automated testing*: data and model validation before deployment.
- *ML pipeline automation*: automating the training workflow via CI/CD pipelines.
- *Model serving and inference*: deploying trained models for real-time or batch predictions.
- *Performance monitoring*: tracking model accuracy, data drift, and concept drift in production.

MLOps maturity levels

- **Level 0, Manual**: all steps (data preparation, training, deployment) are performed manually.
- **Level 1, ML pipeline automation**: model training and validation are automated via pipelines.
- **Level 2, CI/CD automation**: automated testing, deployment, and monitoring close the feedback loop.

Concept drift: A shift in the data distribution over time. It is especially relevant for time-series models, where the underlying process may evolve. Monitoring for drift and triggering model retraining are essential for maintaining prediction quality.

Example 7.1: A demand-forecasting model for a grocery chain is trained on two years of pre-pandemic sales. After deployment, consumer behavior shifts: online orders surge, certain staple items are stockpiled, and the seasonality of fresh-produce sales changes. The mapping from features (day-of-week, promotion, weather) to demand y is no longer the same as in the training data, and

forecast errors grow even though the model itself is unchanged. Monitoring the rolling prediction error and the input-feature distribution detects the drift and triggers retraining on recent data.

8. Logistic Regression

Goal: Binary (two-class) classification with linear decision boundary.

For binary classification, each entry of vector $\mathbf{y}$ is $y_j \in \{0, 1\}$.

Contents

8.1. Generalized Linear Classification Models **97**
8.2. Basic Linear Model **98**
8.3. Logistic Model **99**
8.4. Cross-Entropy Loss **101**
8.4.1. Entropy 102
8.4.2. Cross-Entropy 103
8.4.3. Binary Cross-Entropy (BCE) Loss 103
8.5. BCE Loss for Logistic Regression **104**
8.5.1. Maximum Likelihood Estimation (MLE) (*) 107
8.6. Odd and Logit **108**
8.7. Summary **109**

8.1. Generalized Linear Classification Models

Goal: Extend linear models to classification by applying a non-linear activation function.

Generalized linear model: A generalized linear model applies a non-linear function $g(\cdot) : \mathbb{R} \to \mathbb{R}$ to the linear combination $\mathbf{w}^T\mathbf{x}_i$:

$$\hat{y}_i = g(\mathbf{w}^T\mathbf{x}_i) \tag{8.1}$$

The choice of $g(\cdot)$ determines the model type. The decision boundary $\{\mathbf{x} : \mathbf{w}^T\mathbf{x} \lessgtr thr\}$ is linear (hyperplane), regardless of the choice of $g(\cdot)$ (Fig. 8.1).

Two important questions:

- Selection of a loss function.
- Minimization of a selected loss function.

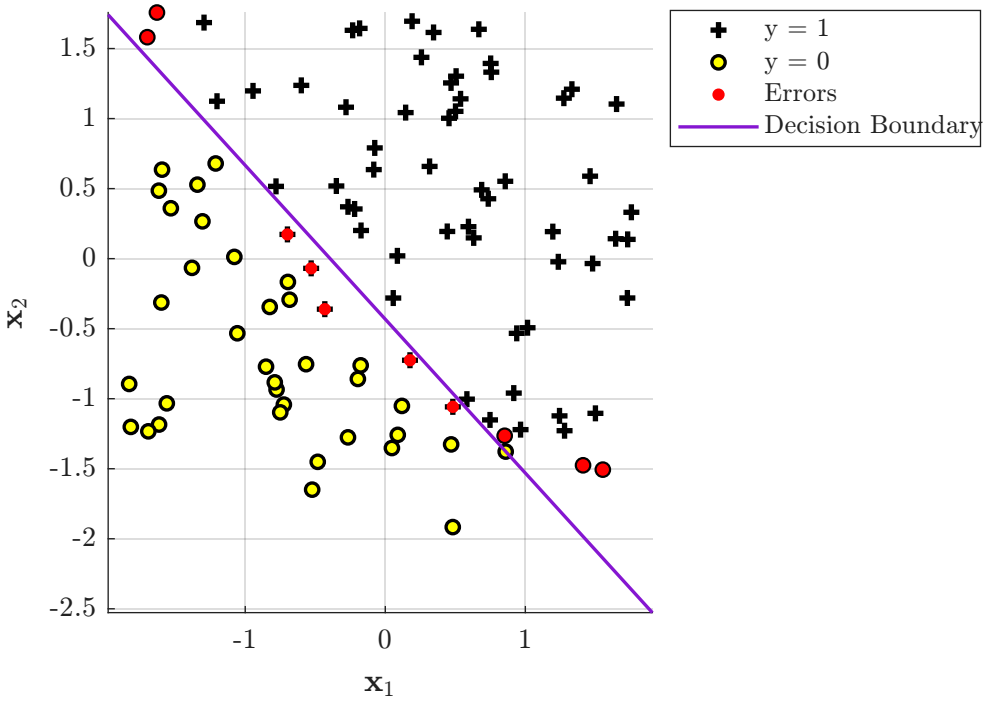


Figure 8.1.: An example of linear classification boundary.

8.2. Basic Linear Model

Goal: Use linear regression for classification as a baseline approach.

Linear model is with

$$g(x) = x \tag{8.2}$$

and MSE loss. It has unbounded and continuous output.

Linear regression can be applied to classification by thresholding the output, as follows:

1. Compute regression weights $\mathbf{w}$ according to data $\mathbf{X}$ and the binary vector $\mathbf{y}$ (e.g., by (3.5)).
2. Compute regression output according to (8.1) with (8.2) substituted. Then, apply threshold 0.5 to obtain class labels:

$$\hat{y}_i = \begin{cases} 1 & \mathbf{w}^T\mathbf{x}_i > 0.5 \\ 0 & \mathbf{w}^T\mathbf{x}_i \leqslant 0.5 \end{cases} \tag{8.3}$$

Example 8.1: Consider $M = 20$ samples with a single feature $x_1 \in [10, 29]$: the first ten labeled $y = 1$ and the last ten $y = 0$. The design matrix $\mathbf{X} = [\mathbf{1}_M \;\; \mathbf{x}] \in \mathbb{R}^{M\times 2}$ includes a column of ones for the intercept (Sec. 3.1). Fitting $\mathbf{w}$ by least squares and thresholding $\hat{y} = \mathbf{w}^T\mathbf{x} \lessgtr 0.5$ classifies all points correctly (Fig. 8.2(a)).

Adding ten class-0 outliers at $x_1 \in [60, 69]$ pulls the regression line toward them, shifting the decision boundary and causing misclassification near the original boundary (Fig. 8.2(b)).

Limitations

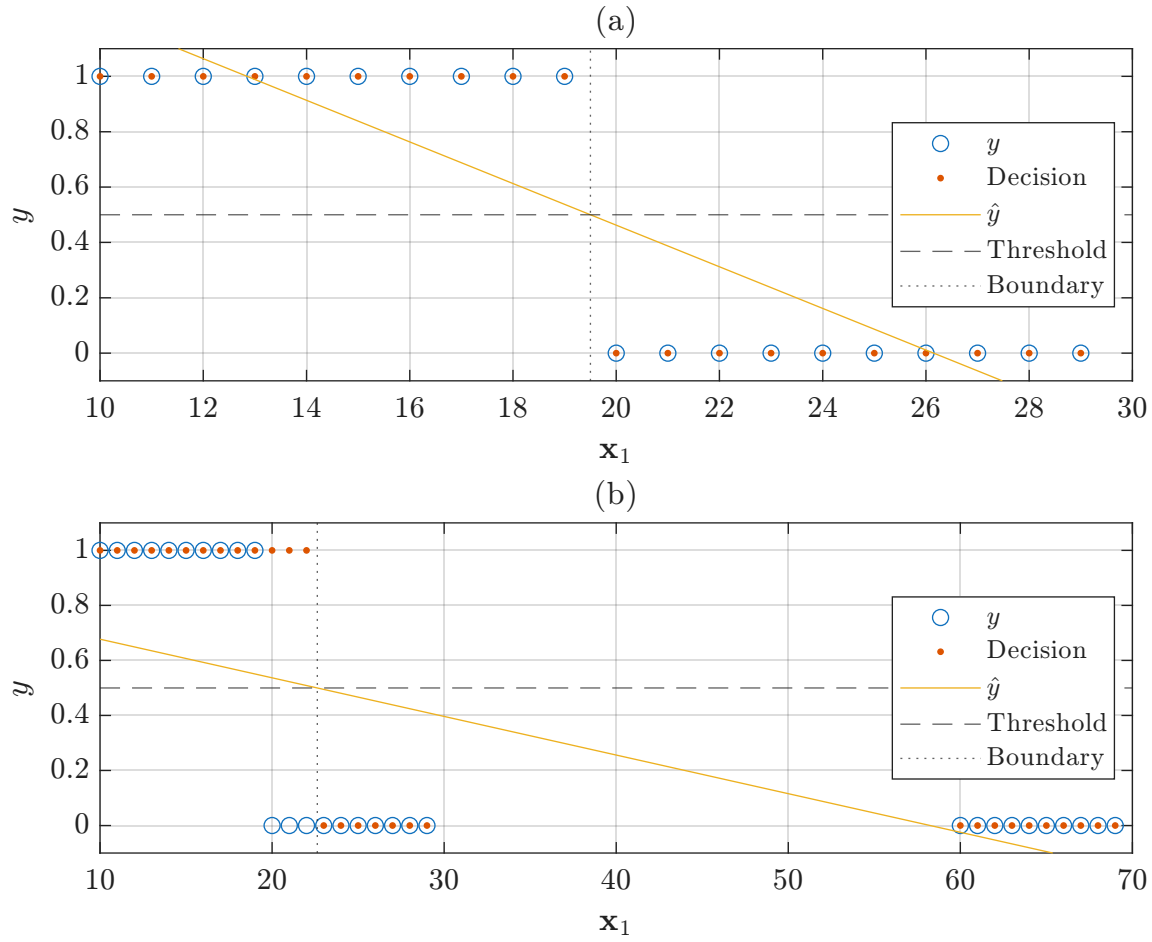


Figure 8.2.: Linear regression for 1D binary classification: (a) least-squares fit with threshold at 0.5 correctly separates the two classes; (b) adding class-0 outliers shifts the decision boundary and causing misclassification.

- **Unbounded output**: $\tilde{y}$ can be significantly larger than 1 or smaller than 0, with no probabilistic interpretation.
- **Outlier sensitivity**: Distant points disproportionately influence the regression line, shifting the decision boundary (Fig. 8.2(b)).

These limitations motivate the logistic model.

8.3. Logistic Model

Goal: Binary classification model with:

- Generalized linear model
- Outlier robustness
- Probabilistic interpretation

The logistic model addresses the limitations of the basic linear model by applying a sigmoid function (Fig. 8.3):

$$\sigma(x) = \frac{\exp(x)}{1+\exp(x)} = \frac{1}{1+\exp(-x)} \tag{8.4}$$

Because $\sigma(x) \in [0, 1]$, the output is bounded and can be interpreted as a probability. The saturating tails reduce the influence of distant points, improving robustness to outliers.

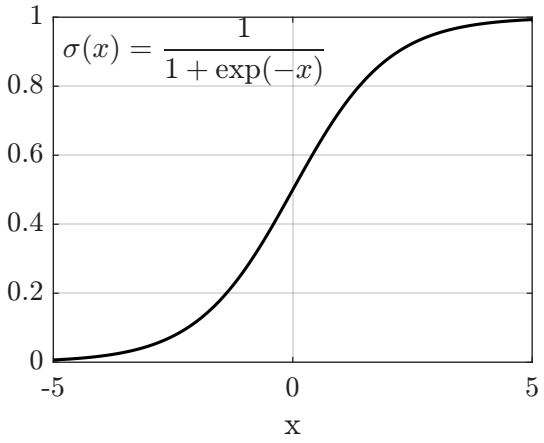


Figure 8.3.: Sigmoid function: bounded output $\sigma(x) \in [0, 1]$ with saturating tails.

Logistic regression model Substituting $g(x) = \sigma(x)$ into (8.1) gives the logistic model. The decision rule with threshold thr is

$$\hat{y} = \begin{cases} 1 & \sigma(\mathbf{w}^T\mathbf{x}) > thr \\ 0 & \sigma(\mathbf{w}^T\mathbf{x}) \leq thr \end{cases} \tag{8.5}$$

With the default $thr = 0.5$, the rule simplifies to

$$\hat{y} = \begin{cases} 1 & \mathbf{w}^T\mathbf{x} > 0 \\ 0 & \mathbf{w}^T\mathbf{x} \leq 0 \end{cases} \tag{8.6}$$

since $\sigma(0) = 0.5$.

Example 8.2: Using the same 1D dataset from Sec. 8.2, the logistic model correctly separates the two classes (Fig. 8.4(a)). Unlike linear regression, adding ten class-0 outliers at $x_1 \in [60, 69]$ does not significantly shift the decision boundary (Fig. 8.4(b)), because the sigmoid saturates for large $|\mathbf{w}^T\mathbf{x}|$.

Why not MSE? Applying MSE loss to the logistic model,

$$\mathcal{L}(\cdot) = \frac{1}{2M}\|\hat{\mathbf{y}} - \mathbf{y}\|^2 \tag{8.7}$$

yields a non-convex optimization problem with multiple local minima and no closed-form solution. This motivates the cross-entropy loss in the following section (Sec. 8.4).

A loss function is not necessarily a metric.

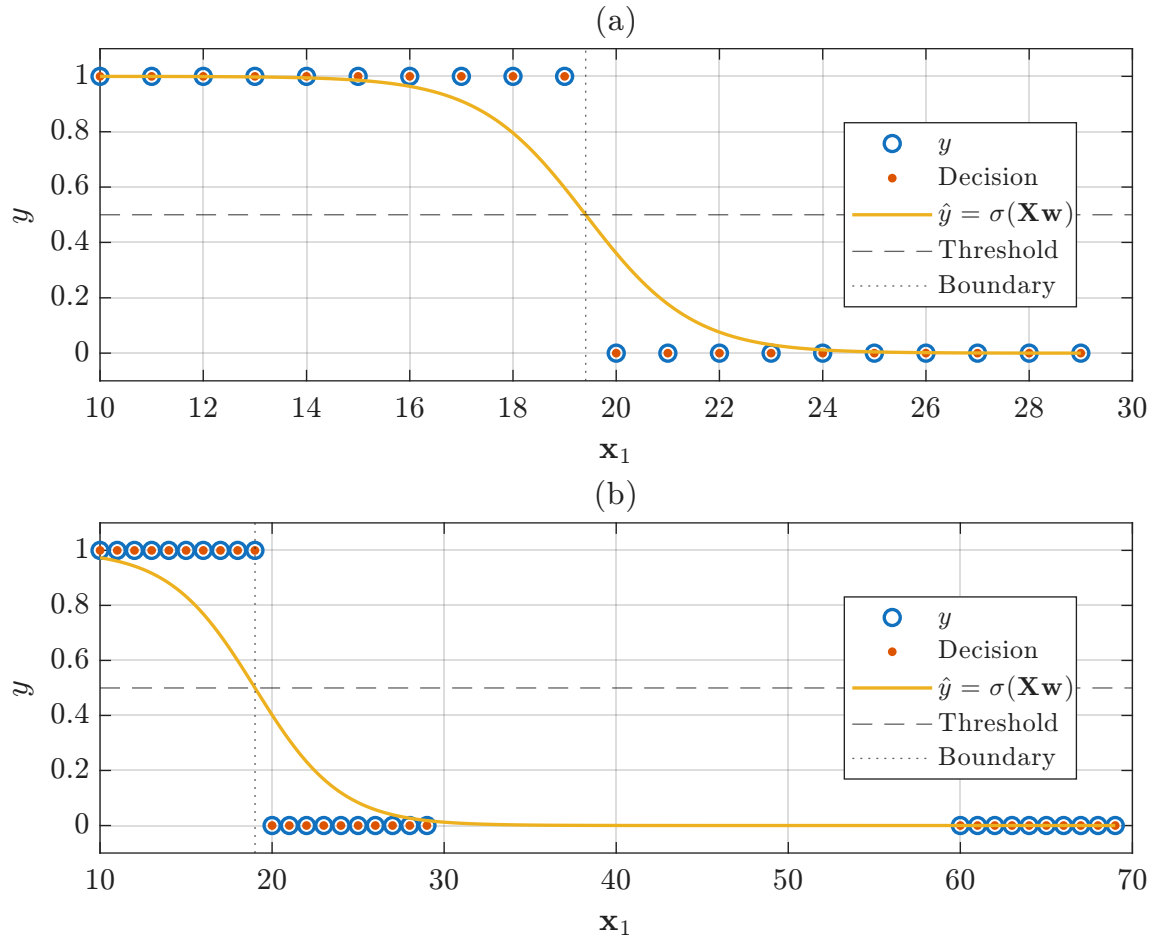


Figure 8.4.: Logistic regression for 1D binary classification: (a) sigmoid fit correctly separates the two classes; (b) adding class-0 outliers does not shift the decision boundary, unlike linear regression (Fig. 8.2).

8.4. Cross-Entropy Loss

Goal: Probabilistic loss that quantifies the distance between target and predicted distributions.

The logistic model output $\hat{y} = \sigma(\mathbf{w}^T\mathbf{x}) \in [0, 1]$ can be interpreted as the probability $\Pr\big(y = 1 \mid \mathbf{x}, \mathbf{w}\big)$. For each sample, both the binary label $y \in \{0, 1\}$ and the scalar prediction $\hat{y} \in [0, 1]$ thus define a Bernoulli distribution over the two outcomes $\{y = 0,\ y = 1\}$: the *target* distribution p is one-hot, concentrated on the true class, and the *predicted* distribution q is $(1-\hat{y},\ \hat{y})$ produced by the sigmoid (Table 8.1).

Table 8.1.: Mapping of label y and prediction $\hat{y}$ to two-outcome distributions p (target) and q (predicted).

	Pr(y=0)	Pr(y=1)
$y = 0$ (target p)	$p_0 = 1$	$p_1 = 0$
$y = 1$ (target p)	$p_0 = 0$	$p_1 = 1$
$\hat{y} = \sigma(\mathbf{w}^T\mathbf{x})$ (predicted q)	$q_0 = 1 - \hat{y}$	$q_1 = \hat{y}$

This reframes classification as *distribution matching*: training should push the predicted distribution q toward the target distribution p. A suitable loss must therefore:

(i) reach its minimum only when $q = p$,
(ii) grow as q moves away from p, and
(iii) remain well-defined when p is one-hot.

Cross-entropy loss, rooted in information theory, satisfies all three properties.

8.4.1. Entropy

Entropy: For a discrete distribution $P = \{p_i = \Pr[X = x_i]\}$, the entropy is

$$H(P) = -\sum_i p_i \log(p_i) \tag{8.8}$$

Entropy measures the uncertainty of a distribution P. It is maximized when all outcomes are equally likely ($p_i = p_j \; \forall \, i, j$) and decreases as the distribution becomes more concentrated.

Coding interpretation: With base-2 logarithm, entropy gives the theoretical minimum average number of bits needed to encode outcomes drawn from P.

Example 8.3: For a binary distribution:

$$p_1 = p_2 = \frac{1}{2} \Rightarrow H(P) = -2 \cdot \tfrac{1}{2} \log_2\left(\tfrac{1}{2}\right) = 1 \text{ bit}$$
$$p_1 = \frac{1}{10},\; p_2 = \frac{9}{10} \Rightarrow H(P) = -\tfrac{1}{10} \log_2\left(\tfrac{1}{10}\right) - \tfrac{9}{10} \log_2\left(\tfrac{9}{10}\right) \approx 0.469 \text{ bits}$$

The more concentrated distribution has lower entropy and *theoretically* requires fewer bits on average.

Example 8.4: Consider transmitting letters $\{A, B, C, D\}$ over a binary channel.

Equal probabilities. If all four letters are equally likely ($p_i = 0.25$), an optimal code is $\{00, 01, 10, 11\}$ that are two bits per letter. The entropy confirms: $H(P) = -4 \cdot \frac{1}{4} \log_2\left(\frac{1}{4}\right) = 2$ bits.

Unequal probabilities. With the distribution in Table 8.2, shorter codewords are assigned to more probable letters. The average code length is

$$\sum_i \text{length}_i \cdot p_i = 1 \cdot 0.7 + 2 \cdot 0.26 + 3 \cdot 0.02 + 3 \cdot 0.02 = 1.34 \text{ bits}$$

while the entropy gives the theoretical minimum:

$$H(P) = -0.7 \log_2(0.7) - 0.26 \log_2(0.26) - 2 \cdot 0.02 \log_2(0.02) \approx 1.091 \text{ bits}$$

Table 8.2.: Variable-length coding *example* for an unequal distribution.

Letter	Probability, p_i	Codeword	Length
A	0.70	0	1
B	0.26	10	2
C	0.02	110	3
D	0.02	111	3

8.4.2. Cross-Entropy

Goal: Quantify distance between distributions p and q.

Cross-entropy: For two discrete distributions p and q over the same outcomes, the cross-entropy is

$$H(p,q) = -\sum_i p_i \log(q_i) \tag{8.9}$$

Cross-entropy satisfies $H(p,q) \geq H(p)$, with equality if and only if $p = q$.

Coding interpretation: With base-2 logarithm, $H(p,q)$ is the average number of bits needed to encode outcomes drawn from p using a code optimized for q.

Example 8.5: Let $q_i = \left\{\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\right\}$ (optimal code: two bits per letter) and $p_i = \left\{\frac{1}{2}, \frac{1}{2}, 0, 0\right\}$. Then $H(p) = 1$ bit, but $H(p,q) = 2$ bits: using a code designed for q wastes one bit per symbol on average when the true distribution is p.

The convention $\lim_{x\to 0} x \log(x) = 0$ is used so that zero-probability events contribute nothing. For loss functions, the natural logarithm (ln) is used. Minimizing cross-entropy with respect to model parameters $\mathbf{w}$ is equivalent to maximum likelihood estimation (MLE).

8.4.3. Binary Cross-Entropy (BCE) Loss

Goal: Convex loss for binary classification with probabilistic interpretation.

For a binary outcome, the cross-entropy reduces to

$$H(p,q) = -p_0 \log(q_0) - p_1 \log(q_1) \tag{8.10}$$

which is visualized in Fig. 8.5. When $y = 1$ (i.e., $p_0 = 0, p_1 = 1$), the expression reduces to $H(p,q) = -\log(q_1)$, which penalizes small predicted probabilities q_1.

For a single sample with true label $y \in \{0,1\}$ and predicted probability $\hat{y} = f_{\boldsymbol{\theta}}(\mathbf{x}) \in (0,1)$, the target and predicted distributions are

$$\begin{aligned} p_0 &= 1 - y, \quad & p_1 &= y \\ q_0 &= 1 - f_{\boldsymbol{\theta}}(\mathbf{x}), \quad & q_1 &= f_{\boldsymbol{\theta}}(\mathbf{x}) \end{aligned}$$

Substituting into Eq. (8.10):

$$H(p,q) = -(1-y)\log\big(1 - f_{\boldsymbol{\theta}}(\mathbf{x})\big) - y \log\big(f_{\boldsymbol{\theta}}(\mathbf{x})\big)$$

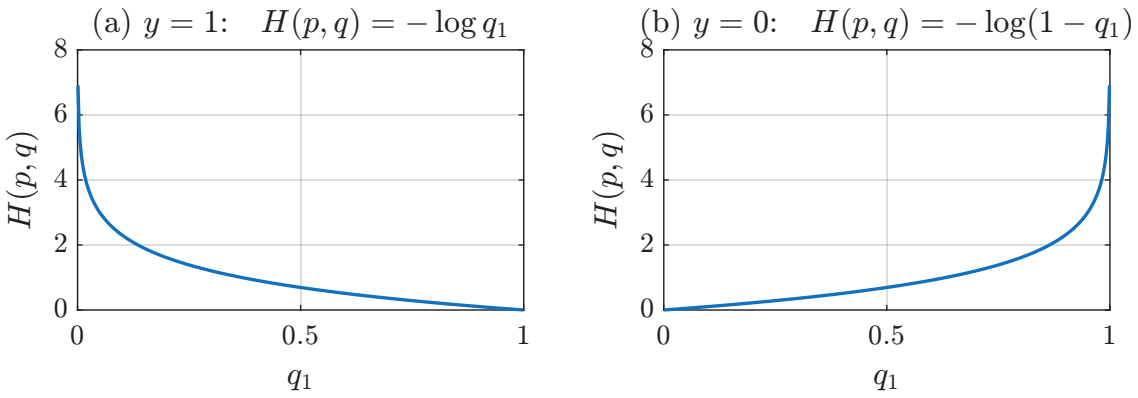


Figure 8.5.: Binary cross-entropy for the two cases: $y = 1$ (left) and $y = 0$ (right) (Eq. (8.10)).

BCE loss: Binary cross-entropy (BCE) loss for a single sample:

$$\mathcal{L}(y, \hat{y}) = -(1-y)\log(1-\hat{y}) - y\log(\hat{y}) \tag{8.11}$$

For M samples, the loss is averaged over all elements:

$$\begin{aligned} \mathcal{L} &= -\frac{1}{M}\sum_{j=1}^{M}\Big[(1-y_j)\log\big(1-\hat{y}_j\big) + y_j\log\big(\hat{y}_j\big)\Big] \\ &= -\frac{1}{M}\Big[(1-\mathbf{y})^T\log(1-\hat{\mathbf{y}}) + \mathbf{y}^T\log(\hat{\mathbf{y}})\Big] \end{aligned} \tag{8.12}$$

Properties: The BCE loss:

- Is continuous, differentiable, and convex—suitable for gradient-based optimization.
- Has a unique global minimum.
- Provides probabilistic predictions:

$$\begin{aligned} \Pr\big(y=1|\mathbf{x},\mathbf{w}\big) &= \sigma(\mathbf{w}^T\mathbf{x}) \\ \Pr\big(y=0|\mathbf{x},\mathbf{w}\big) &= 1-\sigma(\mathbf{w}^T\mathbf{x}) \end{aligned} \tag{8.13}$$

- Yields a classification decision via thresholding, e.g. $\hat{y} \lessgtr \frac{1}{2}$.

8.5. BCE Loss for Logistic Regression

Probabilistic interpretation

The predicted probability $\hat{y} = \sigma(\mathbf{w}^T\mathbf{x})$ partitions the feature space into regions of varying confidence. Fig. 8.6 illustrates four regions: high-confidence positive ($\hat{y} > 0.999$), moderate positive ($0.8 \geq \hat{y} \geq 0.5$), moderate negative ($0.5 > \hat{y} \geq 0.2$), and high-confidence negative ($0.2 > \hat{y}$). Points near the decision boundary have predictions closer to 0.5, reflecting greater classification uncertainty.

The probabilistic output provides confidence levels but does not eliminate classification errors.

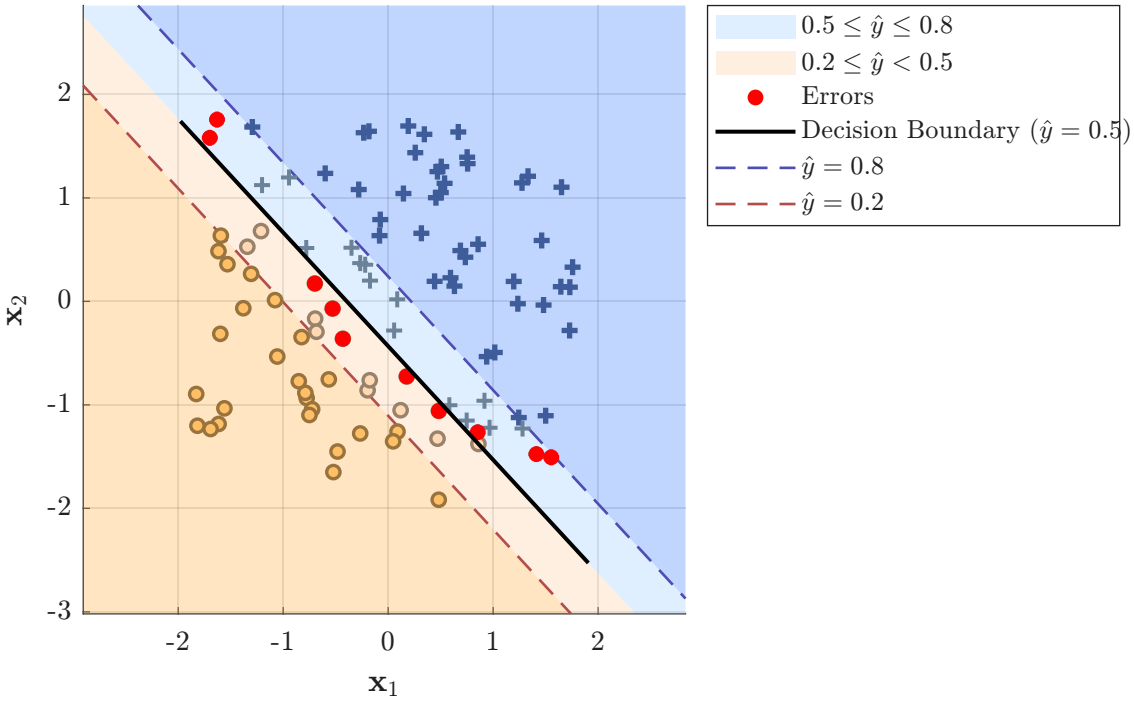


Figure 8.6.: Probabilistic interpretation: regions of predicted probability $\hat{y} = \sigma(\mathbf{w}^T\mathbf{x})$ with boundaries at 0.2 and 0.8. Misclassified points are marked in red (see also Fig. 8.1).

Decision boundary (*)

From Eq. (8.6), the classification rule is

$$\hat{\mathbf{y}} = \begin{cases} 1 & \mathbf{w}^T\mathbf{x} \geq 0 \\ 0 & \mathbf{w}^T\mathbf{x} < 0 \end{cases} \tag{8.14}$$

The decision boundary is the set $\{\mathbf{x} : \mathbf{w}^T\mathbf{x} = 0\}$, i.e., $w_0 + w_1x_1 + w_2x_2 + \cdots = 0$. Geometrically, $\mathbf{w}^T\mathbf{x} = \|\mathbf{x}\|\|\mathbf{w}\|\cos(\theta)$, so the boundary consists of all points $\mathbf{x}$ perpendicular to $\mathbf{w}$ ($\theta = 90°$), forming a hyperplane with normal vector $\mathbf{w}$.

Loss minimization

Substituting $\hat{\mathbf{y}} = \sigma(\mathbf{X}\mathbf{w})$ into the BCE loss gives the logistic regression objective in vector form:

$$\mathcal{L} = \frac{1}{M}\left[-\mathbf{y}^T \log\big(\sigma(\mathbf{X}\mathbf{w})\big) - (1-\mathbf{y})^T \log\big(1-\sigma(\mathbf{X}\mathbf{w})\big)\right] \tag{8.15}$$

Setting $\nabla_{\mathbf{w}}\mathcal{L} = \mathbf{0}$ has no closed-form solution for $\mathbf{w}$. However, the gradient has a compact form:

$$\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}) = \frac{1}{M}\mathbf{X}^T\left(\sigma\left(\mathbf{X}\mathbf{w}\right) - \mathbf{y}\right) \tag{8.16}$$

which enables GD using mostly matrix–vector operations:

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \alpha\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}) \tag{8.17}$$

Regularization and feature mapping

- **Regularization.** L_2 regularization can be applied, similar to Eq. (4.23)

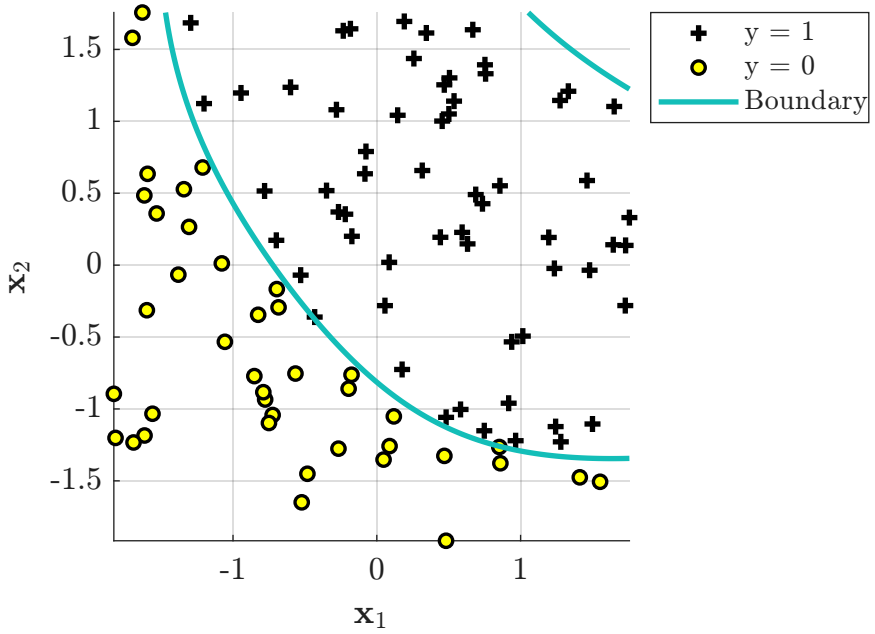
$$\mathcal{L}_{\text{reg}} = \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) + \frac{\lambda}{2M} \sum_{i=1}^{N} w_i^2 \tag{8.18}$$

- **Feature mapping.** Mapping functions or kernels extend the model to non-linear decision boundaries.

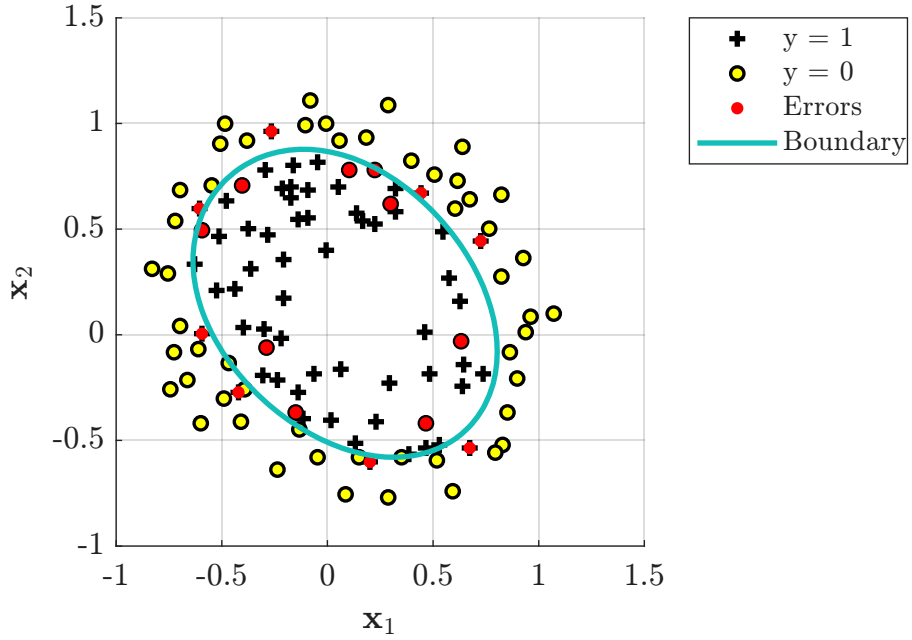
Example 8.6: For example, a polynomial mapping

$$\varphi(x_1, x_2) = \langle 1, x_1, x_1^2, x_2, x_2^2, x_1 x_2, x_1^2 x_2, x_1 x_2^2, x_1^2 x_2^2 \rangle$$

replaces $\mathbf{x}$ with $\varphi(\mathbf{x})$, yielding a non-linear boundary in the original input space while remaining linear in the feature space (Fig. 8.7).



(a) Same dataset as Figs. 8.1 and 8.6: the quadratic boundary separates both classes without errors. Note the boundary in the upper-left corner.



(b) A different dataset with a non-convex class structure; the elliptical boundary captures the inner class, though some points (red) are misclassified.

Figure 8.7.: Degree-2 polynomial feature mapping $\varphi(x_1, x_2)$ produces non-linear decision boundaries in the original input space.

8.5.1. Maximum Likelihood Estimation (MLE) (*)

The use of the BCE loss is not arbitrary; it is the direct result of applying the maximum likelihood principle.

Given a dataset of M samples, and assuming the samples are independent and identically distributed (i.i.d.), the likelihood of the parameters $\mathbf{w}$ is the joint probability of observing the targets $\mathbf{y}$ given the inputs $\mathbf{X}$:

$$L(\mathbf{w}) = \prod_{i=1}^{M} \Pr\big(y_i \mid \mathbf{x}_i, \mathbf{w}\big) \tag{8.19}$$

Since each y_i follows a Bernoulli distribution with parameter $p_i = \sigma(\mathbf{w}^T \mathbf{x}_i)$, we write:

$$\Pr\big(y_i \mid \mathbf{x}_i, \mathbf{w}\big) = \hat{y}_i^{\,y_i} (1 - \hat{y}_i)^{1-y_i} \tag{8.20}$$

To simplify optimization, we maximize the log-likelihood:

$$\ell(\mathbf{w}) = \log L(\mathbf{w}) = \sum_{i=1}^{M} \big[y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i) \big] \tag{8.21}$$

Maximizing the log-likelihood is mathematically equivalent to minimizing the negative log-likelihood. Dividing by M to get the average, we arrive exactly at the BCE loss formula defined in Sec. 8.4:

$$\min_{\mathbf{w}} \mathcal{L} = \min_{\mathbf{w}} \left\{ -\frac{1}{M} \ell(\mathbf{w}) \right\} \tag{8.22}$$

8.6. Odd and Logit

If $\Pr(\hat{y} = 1) = p$, than the *odds* (probability of event divided by probability of no event) is defined by ratio

$$\text{odds} = \frac{\Pr(\hat{y} = 1)}{\Pr(\hat{y} = 0)} = \frac{\Pr(\hat{y} = 1)}{1 - \Pr(\hat{y} = 1)} = \frac{p}{1 - p} \tag{8.23}$$

The odds range from $[0, \infty)$.

Taking the natural logarithm gives the *logit* function (or log-odds):

$$\text{logit} = \ln \left(\frac{p}{1 - p} \right) \tag{8.24}$$

The logit maps probabilities from the range $(0, 1)$ to the entire real line $(-\infty, \infty)$.

Logistic Regression As already mentioned in (8.13),

$$\Pr(y = 1) = \sigma(\mathbf{w}^T \mathbf{x}) = \frac{1}{1 + \exp\big(-\mathbf{w}^T \mathbf{x}\big)} = p \tag{8.25}$$

By reformulation,

$$\frac{p}{1 - p} = \frac{\Pr(\hat{y} = 1)}{\Pr(\hat{y} = 0)} = \exp\Big(\mathbf{w}^T \mathbf{x}\Big) \tag{8.26}$$

$$\ln \left(\frac{p}{1 - p} \right) = \mathbf{w}^T \mathbf{x} \tag{8.27}$$

To illustrate the influence of $\tilde{x}_i = x_i + \Delta x$,

$$\begin{aligned}\frac{\text{odds}_{\Delta x}}{\text{odds}} &= \frac{\exp\big(w_0 + w_1 x_1 + \cdots + w_i(x_i + \Delta x) + \cdots + w_N x_n\big)}{\exp(w_0 + w_1 x_1 + \cdots + w_i x_i + \cdots + w_N x_n)} \\ &= \exp\big(w_i(x_i + \Delta x) - w_i x_i\big) = \exp(w_i \Delta x)\end{aligned} \tag{8.28}$$

Example 8.7: A logistic regression model for exam pass/fail with a single feature (hours studied, x_1) yields weights $w_0 = -4,\ w_1 = 0.5$. For a student who studied $x_1 = 10$ hours:

$$\mathbf{w}^T\mathbf{x} = w_0 + w_1 x_1 = -4 + 0.5 \cdot 10 = 1$$
$$p = \sigma(1) = \frac{1}{1 + e^{-1}} \approx 0.731$$

The odds and logit are:

$$\text{odds} = \frac{p}{1-p} = \frac{0.731}{0.269} \approx 2.72 = e^1$$
$$\text{logit} = \ln(2.72) = 1 = \mathbf{w}^T\mathbf{x}$$

An odds of 2.72 means the model considers passing roughly 2.7 times more likely than failing.

If the student studies $\Delta x = 2$ additional hours, the odds ratio is

$$\frac{\text{odds}_{\Delta x}}{\text{odds}} = \exp(w_1 \Delta x) = \exp(0.5 \cdot 2) = e \approx 2.72$$

Each extra two hours of study multiplies the odds of passing by ≈ 2.72, regardless of the starting point.

8.7. Summary

- Logistic regression applies the sigmoid function to a linear model, producing bounded output with probabilistic interpretation.
- BCE loss is convex, differentiable, and equivalent to maximum likelihood estimation.
- The decision boundary is a hyperplane in input space; non-linear boundaries require feature mapping.
- Optimization is performed via gradient descent (no closed-form solution for $\mathbf{w}$).
- Regularization and polynomial/kernel mappings extend naturally from linear regression.
- Complete separation failure: If there is a feature that would perfectly separate the two classes, the weight for that feature would not converge, because the optimal weight would be infinite.

9. Feature Engineering and Selection

Goal: Working with feautures:

- Feature engineering (FE) transforms raw features into representations that improve model performance. It extends existing features, $M \times N \to M \times N'$, such that $N < N'$ or $N \ll N'$.
- Feature selection (FS) selects a subset of relevant features, $M \times N \to M \times N'$, such that $N > N'$ or $N \gg N'$.

Contents

9.1. Categorical Encoding . . . 110
9.1.1. Label Encoding . . . 111
9.1.2. Ordinal Encoding . . . 111
9.1.3. One-Hot Encoding . . . 111
9.1.4. When to Use Which . . . 111
9.2. Feature Transformations . . . 112
9.3. Date and Time Features . . . 113
9.4. Feature Selection Concept . . . 114
9.5. Statistical-Based . . . 115
9.5.1. Variance Threshold . . . 115
9.5.2. ANOVA . . . 116
9.6. Classifier-Based . . . 118
9.7. Adjusted Coefficient of Determination . . . 118

Feature engineering transforms raw variables into representations that are easier for a model to use. Instead of only removing columns, it creates, reformulates, or encodes features so that useful patterns become more explicit. Common goals are to expose non-linear relationships and better match the limitations of a learning algorithm.

Feature engineering and feature selection serve different purposes. Feature engineering creates new or improved representations, while feature selection removes irrelevant or redundant ones. In a typical pipeline, feature engineering is applied first, followed by feature selection.

9.1. Categorical Encoding

Goal: Convert categorical (non-numeric) features into numeric representations suitable for mathematical models.

Cardinality: The cardinality C of a categorical feature is the number of its unique values (categories). A feature with many unique values is referred to as having *high cardinality*.

9.1.1. Label Encoding

Label encoding[1] assigns each category an integer in arbitrary (typically alphabetical) order. For a feature with categories $\{A, B, C\}$, label encoding maps $A \mapsto 0$, $B \mapsto 1$, $C \mapsto 2$. The assigned integers carry no semantic meaning, so label encoding is primarily intended for target labels in classification rather than for input features.

9.1.2. Ordinal Encoding

Ordinal encoding[2] also maps each category to an integer, but using a user-specified order so that the encoded values preserve the natural rank of the categories. For example, $\{\text{low}, \text{medium}, \text{high}\} \mapsto \{0, 1, 2\}$. Unlike label encoding, ordinal encoding is intended for input features whose categories have a meaningful order.

9.1.3. One-Hot Encoding

One-hot encoding[3] creates a binary indicator column for each category. A feature with C categories is replaced by C binary features.

Example 9.1 (One-Hot Encoding): Consider a feature "Color" with three categories: $\{\text{Red}, \text{Green}, \text{Blue}\}$ and $M = 4$ samples:

Color	Red	Green	Blue
Red	1	0	0
Blue	0	0	1
Green	0	1	0
Red	1	0	0

The single categorical feature is replaced by three binary features.

9.1.4. When to Use Which

- *Label encoding* is used for *target labels* in classification, where the integer assignment is just an identifier and its order does not matter.
- *Ordinal encoding* is suitable for *ordinal* input features (where the order is meaningful) and for tree-based models (e.g., decision trees, random forests), which split on thresholds and are therefore unaffected by the particular integer values, as long as the relative order is preserved.

[1] In Python: `sklearn.preprocessing.LabelEncoder`
[2] In Python: `sklearn.preprocessing.OrdinalEncoder`
[3] In Python: `sklearn.preprocessing.OneHotEncoder`

- *One-hot encoding* is preferred for *nominal* features (no natural order) and for linear/logistic models, which interpret numeric values as quantities and would otherwise assume a meaningful ordering among categories.

Label or ordinal encoding applied to *nominal* features can mislead *distance-based* models such as k-NN or SVM with an RBF kernel. For example, encoding $\{\text{Red} \mapsto 0,\ \text{Green} \mapsto 1,\ \text{Blue} \mapsto 2\}$ implies that the distance between Red and Blue is twice the distance between Red and Green, a relationship that is meaningless for nominal features. One-hot encoding avoids this problem: in the one-hot representation every pair of distinct categories is equidistant (Euclidean distance $\sqrt{2}$).

9.2. Feature Transformations

Goal: Transform numerical features into representations that better match the structure of the problem and the assumptions of the model.

Many models are easiest to train when the relationship between the features and the target is relatively simple. Feature transformations apply a mapping $x \mapsto \phi(x)$ that can linearize trends, expose interactions, or reduce the influence of extreme values.

Polynomial features

Polynomial features[4] generate interaction and higher-order terms from the original features. For N features and degree d, the transformation produces all monomials up to degree d. This allows a linear model to represent non-linear relationships in the original input space. It is closely related to the polynomial kernel approach (see Sec. 4.1 and Kernels in chapter 5).

Example 9.2 (Polynomial Features): For two features x_1, x_2 with degree $d = 2$, the expanded feature set is:

$$[x_1, x_2] \to [1,\ x_1,\ x_2,\ x_1^2,\ x_1x_2,\ x_2^2]. \tag{9.1}$$

The dimensionality increases from $N = 2$ to $N' = 6$ (including the bias term).

The main advantage of polynomial features is that they let a linear model fit non-linear patterns and feature interactions without changing the learner itself. Their main drawback is dimensionality growth: the number of terms scales as $\binom{N+d}{d}$, so both training cost and the risk of overfitting grow rapidly with N or d.

Log transform applies $x' = \log(x)$ (for $x > 0$) to reduce right-skewness and compress the range of features spanning several orders of magnitude. It is useful for variables such as income, population, or concentration, where multiplicative changes are often more meaningful than additive ones. The transform converts multiplicative relationships into additive ones. Its main strength is that it stabilizes variance and tames heavy-tailed distributions, which typically helps linear models. Its main limitation is that it is defined only for strictly positive inputs, so a shift such as $\log(1 + x)$ is required when zeros are present.

[4] In Python: `sklearn.preprocessing.PolynomialFeatures`

Binning (discretization) converts continuous features into categorical ones by partitioning the value range into intervals. For example, an age feature can be binned into groups: $[0, 18)$, $[18, 65)$, $[65, \infty)$. This can reduce the effect of outliers and capture non-linear step-like relationships when only coarse ranges are important. On the positive side, binning is robust to outliers and can expose coarse threshold-like effects that a linear model would otherwise miss. On the negative side, all information within a bin is discarded, and the bin edges introduce somewhat arbitrary discontinuities.

9.3. Date and Time Features

Goal: Extract informative features from date and time variables to capture temporal patterns.

A raw datetime variable (e.g., a timestamp) usually carries little direct meaning for a learning algorithm. If it is used as a single number, the model may treat time as only increasing, while ignoring weekly, monthly, or daily patterns. Extracting its components creates features that expose cyclic and calendar-based structure:

- *Day of week* (1–7): captures weekly seasonality (e.g., weekday vs. weekend behavior).
- *Day of month* (1–31): captures monthly patterns (e.g., salary-related spending).
- *Month* (1–12): captures annual seasonality (e.g., weather, holidays).
- *Hour of day* (0–23): captures intra-day patterns (e.g., traffic peaks).
- *Quarter* (1–4): captures business-cycle effects.
- *Is weekend* (binary): a derived indicator for weekend days.

Because many of these features are cyclic (e.g., hour 23 is close to hour 0), encoding them as a single integer can mislead distance-based models. A common solution is *cyclical encoding* using sine and cosine:

$$x_{\sin} = \sin\left(\frac{2\pi\, x}{P}\right), \quad x_{\cos} = \cos\left(\frac{2\pi\, x}{P}\right), \tag{9.2}$$

where P is the period (e.g., $P = 7$ for day of week, $P = 24$ for hour of day). This maps each cyclic value onto the unit circle, so that the first and last values in the cycle are adjacent.

Example 9.3 (Date Feature Extraction): Given a timestamp `2024-12-20 14:30`, the extracted features are:

Day of week	5 (Friday)
Day of month	20
Month	12
Hour	14
Is weekend	0

Component extraction exposes weekly, monthly, and daily patterns that are invisible in a raw timestamp, and cyclical sine/cosine encoding preserves the natural proximity between the first and last values of a cycle. The cost is higher dimensionality, since cyclical encoding produces two features per component, and the choice of which components to extract depends on domain knowledge about the problem at hand.

9.4. Feature Selection Concept

Goal: Select a subset of relevant features to reduce dimensionality and improve model performance.

Feature selection chooses a subset of the original features. Given a dataset

$$X \in \mathbb{R}^{M \times N}, \tag{9.3}$$

it constructs a reduced representation

$$X' \in \mathbb{R}^{M \times N'}, \quad N' < N, \tag{9.4}$$

by keeping only the most informative columns. The objective is not only to reduce dimensionality, but to preserve or improve predictive performance on unseen data.

Why perform feature selection?

- Some features are *irrelevant* or noisy and do not contribute useful predictive information.
- Some features are *redundant*, for example when two features are highly correlated and carry nearly the same information.
- In high-dimensional problems, unnecessary features can increase overfitting and reduce generalization.
- Each feature may have a cost: measurement, storage, preprocessing, and model-training complexity.
- A smaller feature set often improves interpretability by making the model easier to analyze.

Feature selection also introduces a tradeoff. Removing useful features may degrade performance, so the goal is not to find the smallest possible subset, but rather the smallest *informative* subset according to a chosen criterion.

Main families of methods

Feature selection methods are commonly divided into three groups:

- *Filter methods* score features using statistical properties of the data, independently of the final predictor. In this chapter, variance threshold and ANOVA are examples.
- *Wrapper methods* evaluate subsets by training and testing a predictor. The classifier-based greedy search methods in the next section belong to this family.
- *Embedded methods* perform feature selection as part of model training itself. Tree-based feature importance is an example of this idea.

In addition, some model-comparison criteria explicitly penalize the number of features or parameters. An example of a feature-number-aware comparison metric is adjusted R^2 in Sec. 9.7.

Example 9.4 (Why Feature Selection Can Help): Suppose two sensors measure nearly the same physical quantity, producing features x_1 and x_2, while a third feature x_3 is mostly random noise. A useful feature-selection method may keep only x_1, discard x_2 as redundant, and discard x_3 as irrelevant. The dimensionality decreases from $N = 3$ to $N' = 1$ with little or no loss in predictive information.

Search complexity Exhaustive search over all non-empty subsets of N features requires

$$\sum_{k=1}^{N} \binom{N}{k} = 2^N - 1 \tag{9.5}$$

candidate subsets. This becomes computationally prohibitive even for moderate N. For example, for $N = 15$ features there are

$$2^{15} - 1 = 32767 \tag{9.6}$$

possible subsets to examine.

Fixed subset size. When the target size N' is fixed in advance, the search restricted to subsets of exactly that size, costing $\binom{N}{N'}$ model fits. For $N = 15$ and $N' = 3$ this gives $\binom{15}{3} = 455$ candidate subsets, compared with $2^{15} - 1 = 32767$ for the full exhaustive search. This approach is guaranteed to find the best subset of size N', but its cost grows combinatorially once N' is no longer very small relative to N.

Possible data leakage

Feature selection must be learned using training data only. If features are selected before the train/test split, or before each cross-validation fold is formed, information from the validation or test set leaks into the model-design process. This leads to overly optimistic performance estimates.

9.5. Statistical-Based

There are also statistical FS methods that are not related to a particular classifier.

9.5.1. Variance Threshold

Goal: Eliminate features with little or no variation across samples.

The variance threshold[5] method removes features whose variance falls below a specified threshold, on the premise that a feature which barely changes across samples cannot help discriminate between them. The typical use case is dropping features with zero (or near-zero) variance.

Variance threshold: For a given feature, let M be the number of samples, x_i the i-th sample, and $s_x^2$is the variance. Features with $s_x^2 < \tau$ are removed, where τ is the threshold parameter. Setting $\tau = 0$ drops only exactly constant columns, but floating-point arithmetic during data generation or preprocessing may leave a nominally constant feature with a tiny nonzero variance (on the order of 10^{-15}). A small positive τ (e.g., 10^{-8}) is therefore used in practice to also remove such numerically constant columns.

Variance threshold is *unsupervised*: it does not consult the target variable.

[5] In Python: `sklearn.feature_selection.VarianceThreshold`

9.5.2. ANOVA

Goal: Rank features by how well they separate the classes, using only per-class statistics.

Intuition

Analysis of Variance (ANOVA)[6] asks a single question for each feature: do the classes differ from each other by more than they differ internally?

A feature is informative when the class means are far apart (large *between-class* variation) relative to the spread of samples inside each class (small *within-class* variation). If the two are comparable, the apparent gap between class means could be explained by noise alone, and the feature carries little discriminative signal.

This comparison is summarized by a single number, the *F-statistic*: larger F means the feature separates the classes more cleanly than random fluctuation would suggest. Features are then ranked by F and the top-scoring ones are kept (Fig. 9.1).

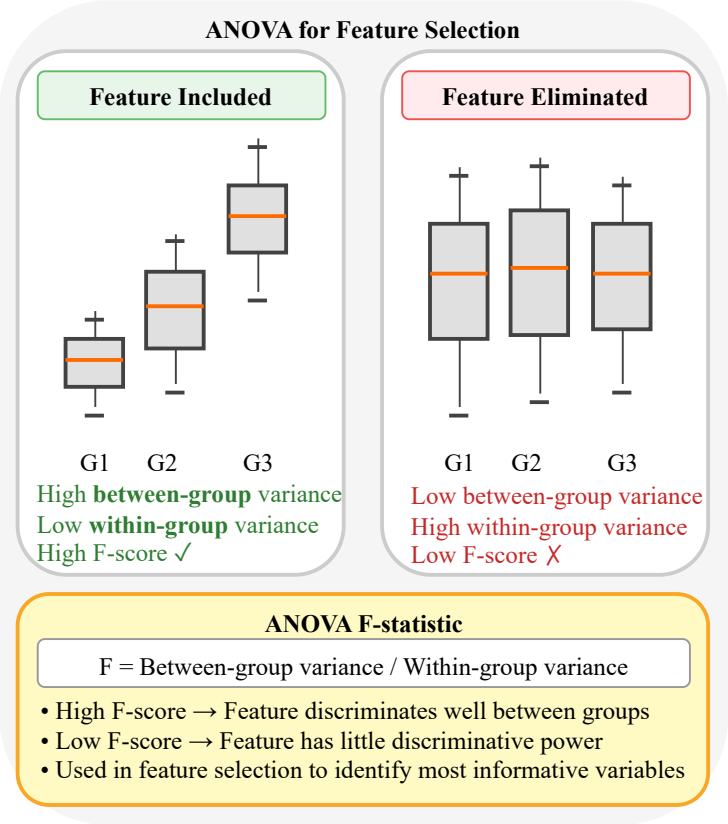


Figure 9.1.: Illustration of ANOVA FS.

Calculation

For a given feature, let:

[6] In Python: `sklearn.feature_selection.f_classif`

- K be the number of classes,
- M_k the number of samples in class k,
- $x_{k,i}$ the i-th sample in class k,
- $\bar{x}_k$ the mean of class k, and
- $\bar{x}$ the global mean.

The between-class sum of squares measures how much class means deviate from the global mean:

$$SS_B = \sum_{k=1}^{K} M_k(\bar{x}_k - \bar{x})^2. \tag{9.7}$$

The within-class sum of squares measures variability within each class:

$$SS_W = \sum_{k=1}^{K} \sum_{i=1}^{M_k} (x_{k,i} - \bar{x}_k)^2. \tag{9.8}$$

The F-statistic is the ratio of mean squares:

$$F = \frac{SS_B/(K-1)}{SS_W/(M-K)} = \frac{MS_B}{MS_W}, \tag{9.9}$$

where $M = \sum_{k=1}^{K} M_k$ is the total number of samples, and $K-1$ and $M-K$ are the corresponding degrees of freedom that normalize each sum of squares. ANOVA assumes features are independent and normally distributed within each class.

Example 9.5 (ANOVA Feature Selection): Consider $K = 2$ classes with $M_1 = M_2 = 3$ samples each, and two candidate features:

Feature 1 (good separation): Class 1: $\{2, 4, 6\}$, Class 2: $\{8, 10, 12\}$.

- Class means: $\bar{x}_1 = 4$, $\bar{x}_2 = 10$, global mean $\bar{x} = 7$.
- $SS_B = 3(4-7)^2 + 3(10-7)^2 = 27 + 27 = 54$.
- $SS_W = (2-4)^2 + (4-4)^2 + (6-4)^2 + (8-10)^2 + (10-10)^2 + (12-10)^2 = 16$.
- $F = \frac{54/1}{16/4} = \frac{54}{4} = 13.5$.

Feature 2 (poor separation): Class 1: $\{5, 6, 7\}$, Class 2: $\{4, 6, 8\}$.

- Class means: $\bar{x}_1 = 6$, $\bar{x}_2 = 6$, global mean $\bar{x} = 6$.
- $SS_B = 3(6-6)^2 + 3(6-6)^2 = 0$.
- $SS_W = 1 + 0 + 1 + 4 + 0 + 4 = 10$.
- $F = 0$.

Feature 1 ($F = 13.5$) is selected over Feature 2 ($F = 0$) since it better separates the classes.

Pros:

- Fast computation: closed-form solution with no model training required.
- Classifier-independent: can be used as a preprocessing step for any model.

Cons:

- Assumes linear separability: only measures differences in means, not complex relationships.
- Evaluates features independently: does not capture feature interactions.
- Sensitive to assumptions: assumes normally distributed data within classes.

9.6. Classifier-Based

Goal: Evaluate each feature by its contribution to a trained classifier's performance.

Greedy search[7] builds a feature subset incrementally, making the locally optimal choice at each step. It does not guarantee a globally optimal subset, but it reduces the search cost from exponential to polynomial in N.

Forward feature selection starts from an empty set and, at each step, adds the feature whose inclusion most improves model performance, measured by cross-validated accuracy or another evaluation metric. The process stops when no remaining feature improves performance or when a predetermined number N' of features is reached.

Backward feature selection starts from the full feature set and, at each step, removes the feature whose removal causes the smallest drop (or the largest gain) in performance. The process stops on the same kinds of criteria.

Forward selection requires about $N \cdot N'$ model fits, and backward selection about $N \cdot (N - N')$, both far fewer than the $2^N - 1$ fits of exhaustive search. Forward selection is therefore cheaper when the target subset is small ($N' \ll N$), while backward selection is preferred when only a few features need to be removed.

Pros:

- Classifier-tailored: feature relevance is evaluated in the context of the actual model.
- Captures feature interactions that affect model performance.

Cons:

- May converge to a local optimum due to its greedy nature.
- Computationally expensive: requires retraining the classifier for each candidate subset.
- Hyperparameter tuning may be needed at every subset size.

9.7. Adjusted Coefficient of Determination

Goal: Provide a comparison metric for multivariate linear models that accounts for different numbers or subsets of features.

While Section 2.3.1 discusses the standard R^2 metric, the regular coefficient of determination has a critical flaw in multivariate regression: it mathematically cannot decrease when new features are added to the model, even if those features are entirely uninformative.

To address this, the *adjusted* R^2 ($\bar{R}^2$) is introduced. It uses a penalty term based on the number of features N relative to the number of samples M. It is given by:

$$\bar{R}^2 = 1 - \left(1 - R^2\right) \frac{M - 1}{M - N - 1} \tag{9.10}$$

[7] In Python: `sklearn.feature_selection.SequentialFeatureSelector`

The primary advantage of $\bar{R}^2$ is that it penalizes the model for adding irrelevant variables. It will only increase if a new feature improves the model's predictive power more than would be expected by chance.

Further Reading

- Sequential Backward and Forward Selection, Andrii Chornyi, 2025 (accessed 16-Apr-26)

10. k-Nearest Neighbors (k-NN)

Goal: Non-parametric classifier based on local similarity.

Unlike logistic regression which learns a parametric decision boundary, k-NN classifies a new point by majority vote among its k nearest neighbors in the training set.

Contents

10.1. Algorithm **120**
10.2. Distance Metrics **121**
10.2.1. Properties 121
10.2.2. Euclidean (L_2) 122
10.2.3. Manhattan (L_1, city block) 122
10.2.4. Minkowski (generalized, hyperparameter p) 122
10.2.5. Canberra 122
10.2.6. Cosine similarity 123
10.3. Bias-Variance Trade-Off **123**
10.4. Curse of Dimensionality **125**
10.4.1. Volume concentrates at the corners 125
10.4.2. Volume concentrates near the surface 126
10.4.3. Samples live in the tails of the density 127
10.4.4. Consequences for Classification 129
10.5. Summary **130**

10.1. Algorithm

For a new sample $\mathbf{x}_0$ and the dataset $\{x_k, y_k\}_{k=1}^{M}$

1. Compute distances between $\mathbf{x}_0$ to each of the dataset samples, $d(\mathbf{x}_0, \mathbf{x}_i)$ (M distances).
2. Select the k samples with smallest distances.
3. Assign the class by majority vote among the k neighbors.

Example 10.1: Fig. 10.1 shows k-NN classification for a query point (star) with $k = 1$, $k = 5$, and $k = 15$. The decision boundary (dashed) becomes smoother as k increases: $k = 1$ produces a complex, noisy boundary, while $k = 15$ yields a simpler one at the cost of reduced local sensitivity.

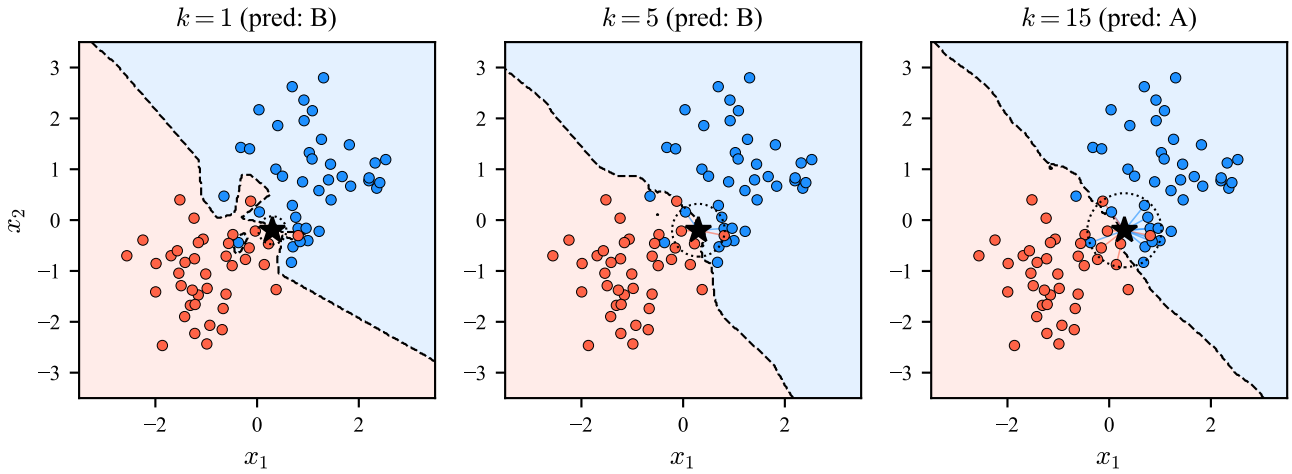


Figure 10.1.: k-NN decision boundaries for different values of k. The query point (star) and its k nearest neighbors (connected by lines) are shown. Larger k smooths the boundary.

Tie-Breaking

When $k > 1$ neighbors sometimes yield equal votes. The ways to handle this situation are:

- Random selection among tied classes.
- Select class of the nearest neighbor among tied samples, i.e. use $k = 1$.

10.2. Distance Metrics

The choice of distance defines what "near" means for k-NN and is therefore as important as the choice of k itself. Different distance metrics emphasize different aspects of the data. The metric is a hyperparameter and is typically selected by cross-validation (Sec. 4.5). Most metrics are not scale-invariant, so features should be standardized before distances are computed (Sec. 4.7).

10.2.1. Properties

A function $d : \mathbb{R}^N \times \mathbb{R}^N \to \mathbb{R}$ is a *distance* (or *metric*) if for all $\mathbf{a}, \mathbf{b}, \mathbf{c}$:

- *Non-negativity*: $d(\mathbf{a}, \mathbf{b}) \geq 0$.
- *Identity of indiscernibles*: $d(\mathbf{a}, \mathbf{b}) = 0 \iff \mathbf{a} = \mathbf{b}$.
- *Symmetry*: $d(\mathbf{a}, \mathbf{b}) = d(\mathbf{b}, \mathbf{a})$.
- *Triangle inequality*: $d(\mathbf{a}, \mathbf{c}) \leq d(\mathbf{a}, \mathbf{b}) + d(\mathbf{b}, \mathbf{c})$.

A function satisfying only a subset of these is a *semi-metric* or *pseudo-metric*; cosine distance is a common example that fails the triangle inequality (see below).

The most popular distance metrics are summarized below.

10.2.2. Euclidean (L_2)

The straight-line distance in $\mathbb{R}^N$,

$$d(\mathbf{a}, \mathbf{b}) = \|\mathbf{a} - \mathbf{b}\| = \sqrt{\sum_{j=1}^{N} (a_j - b_j)^2} \tag{10.1}$$

The default choice for continuous, real-valued features. Because the differences are squared, features with large numerical ranges dominate the sum, so standardization (zero mean, unit variance) is essentially mandatory (Sec. 4.7). Euclidean distance is invariant to rotations and translations of the coordinate system.

10.2.3. Manhattan (L_1, city block)

The sum of absolute coordinate differences,

$$d(\mathbf{a}, \mathbf{b}) = \sum_{j=1}^{N} \left| a_j - b_j \right| \tag{10.2}$$

The distance one would travel on a grid of city blocks, hence the name. Because deviations are not squared, L_1 is less sensitive to outliers and large per-feature gaps than L_2.

10.2.4. Minkowski (generalized, hyperparameter p)

The Minkowski distance is a family that interpolates between Manhattan, Euclidean, and Chebyshev as the exponent p varies; p itself can be treated as a hyperparameter and tuned by cross-validation. For $p \geq 1$,

$$d(\mathbf{a}, \mathbf{b}) = \left(\sum_{j=1}^{N} \left| a_j - b_j \right|^p \right)^{1/p} \tag{10.3}$$

Special cases: $p = 1$ gives Manhattan, $p = 2$ gives Euclidean. As $p \to \infty$ the sum is dominated by its largest term, yielding the Chebyshev (L_∞) distance

$$d_\infty(\mathbf{a}, \mathbf{b}) = \lim_{p \to \infty} \left(\sum_{j=1}^{N} \left| a_j - b_j \right|^p \right)^{1/p} = \max_j \left| a_j - b_j \right|. \tag{10.4}$$

For $p < 1$, $d(\mathbf{a}, \mathbf{b}) = \left(\sum_{j=1}^{N} \left| a_j - b_j \right|^p \right)$ should be used.

10.2.5. Canberra

A weighted L_1 distance that normalizes each coordinate gap by the sum of the magnitudes:

$$d(\mathbf{a}, \mathbf{b}) = \sum_{j=1}^{N} \frac{\left| a_j - b_j \right|}{\left| a_j \right| + \left| b_j \right|}, \tag{10.5}$$

with the convention that the j-th term is 0 when $a_j = b_j = 0$. Each term is bounded in $[0, 1]$, so every coordinate contributes on the same scale regardless of its magnitude, giving the metric an implicit per-feature normalization and a total bounded by N.

Canberra is most useful for non-negative data such as counts, intensities, or ranked features, where relative differences matter more than absolute ones. Compared to Manhattan, a small absolute gap near the origin counts as much as a large gap far from it, so the metric is particularly sensitive to changes near zero. Conversely, when coordinates can be negative or pass through zero, the denominator becomes unstable and the metric loses its appeal.

10.2.6. Cosine similarity

Measures the angle between two non-zero vectors rather than their absolute separation. It is the natural choice for high-dimensional sparse data where vector magnitude is uninformative and only direction carries meaning:

$$\cos(\mathbf{a}, \mathbf{b}) \;=\; \frac{\mathbf{a}^\top \mathbf{b}}{\|\mathbf{a}\|\,\|\mathbf{b}\|} \;=\; \frac{\sum_{j=1}^{N} a_j b_j}{\sqrt{\sum_{j=1}^{N} a_j^2}\,\sqrt{\sum_{j=1}^{N} b_j^2}} \;\in\; [-1, 1]. \tag{10.6}$$

A value of 1 means the vectors point in the same direction, 0 means they are orthogonal, and -1 means opposite directions. The corresponding *cosine distance*

$$d_{\cos}(\mathbf{a}, \mathbf{b}) \;=\; 1 - \cos(\mathbf{a}, \mathbf{b}) \tag{10.7}$$

ranges in $[0, 2]$. Unlike L_p distances, $d_{\cos}$ is scale-invariant: $d_{\cos}(\mathbf{a}, \mathbf{b}) = d_{\cos}(\alpha\mathbf{a}, \beta\mathbf{b})$ for any $\alpha, \beta > 0$, so only the *direction* of the vectors matters.

Strictly speaking, $d_{\cos}$ is not a metric in the mathematical sense (the triangle inequality fails), but on the unit sphere it is monotonically related to Euclidean distance: $\|\hat{\mathbf{a}} - \hat{\mathbf{b}}\|_2^2 = 2\, d_{\cos}(\mathbf{a}, \mathbf{b})$ for unit-normalized $\hat{\mathbf{a}} = \mathbf{a}/\|\mathbf{a}\|$ and $\hat{\mathbf{b}} = \mathbf{b}/\|\mathbf{b}\|$.

Example 10.2: 2-D illustration (See Fig. 10.2) For $\mathbf{a} = (1, 1)$ and $\mathbf{b} = (4, 5)$ the coordinate differences are $|a_1 - b_1| = 3$ and $|a_2 - b_2| = 4$.

- Manhattan (L_1): $d_1 = 3 + 4 = 7$ (sum along axes).
- Euclidean (L_2): $d_2 = \sqrt{3^2 + 4^2} = 5$, the hypotenuse of the right triangle formed by the two legs (Pythagorean theorem).
- Chebyshev (L_∞): $d_\infty = \max(3, 4) = 4$, the larger coordinate gap.
- Canberra: $d = 3/5 + 4/6 \approx 1.267$, each coordinate contributing a value in $[0, 1]$ regardless of its magnitude (no visualization).
- Cosine: $\cos\theta = 9/\sqrt{82} \approx 0.994$, $d_{\cos} \approx 0.006$. Seen as vectors from the origin, $\mathbf{a}$ and $\mathbf{b}$ are nearly parallel, so cosine treats them as very similar despite the non-trivial L_2 gap.

Many additional, domain-specific distances exist beyond the presented ones. The right choice depends on the data type and the notion of similarity the task requires.

10.3. Bias-Variance Trade-Off

The hyperparameter k directly controls the bias-variance trade-off (Sec. 4.4) of the classifier.

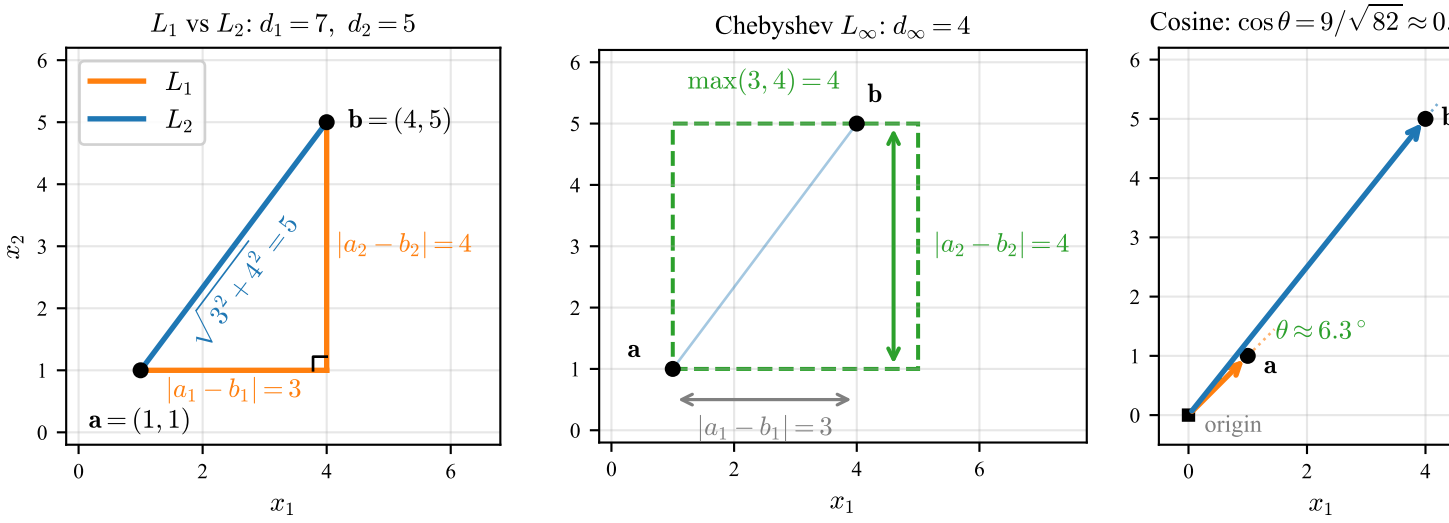


Figure 10.2.: Distance metrics in 2-D for $\mathbf{a} = (1, 1)$, $\mathbf{b} = (4, 5)$. Left: Manhattan path along the legs ($3+4 = 7$) vs Euclidean hypotenuse ($\sqrt{3^2 + 4^2} = 5$). Middle: Chebyshev distance equals the larger coordinate gap, $\max(3, 4) = 4$. Right: cosine similarity is the cosine of the angle between $\mathbf{a}$ and $\mathbf{b}$ seen from the origin; $\cos\theta \approx 0.994$, so the vectors are nearly parallel.

Small k (high variance, low bias)

At $k = 1$, the prediction at every point equals the label of its nearest training sample. The decision boundary follows the data closely and can represent very fine local structure (low bias), but it is also extremely sensitive to noise and to the particular training set drawn: a single mislabeled point or a small shift in the sample changes the boundary in its neighborhood. Different training sets produce visibly different classifiers, which is the sign of high variance.

Large k (low variance, high bias)

As k grows, each prediction averages over more neighbors and the boundary becomes smoother. Random label noise in any individual sample is washed out by the vote, so the classifier is stable across training sets (low variance). The cost is that fine local structure also gets smoothed away: a small minority region surrounded by the opposite class can be outvoted and absorbed (high bias). In the limit $k = M$, every query receives the global majority label.

Fig. 10.1 visualizes this progression: $k = 1$ produces a jagged, noise-following boundary, while $k = 15$ yields a much smoother one that may miss local detail.

Choosing k

k is typically selected by cross-validation (Sec. 4.5) over a grid of candidate values. A common rule of thumb is $k \approx \sqrt{M}$, but cross-validation is preferred whenever the cost permits. Odd k avoids ties in binary problems.

Asymptotic behavior (*)

In the limit of infinite data, k-NN is Bayes-consistent: its error converges to the Bayes-optimal error R^* (Sec. 4.3), provided

- $M \to \infty$: the training set grows without bound;
- $k \to \infty$: the vote averages more samples, so by the law of large numbers its variance vanishes and it becomes a reliable estimate of $P(y \mid \mathbf{x})$;
- $k/M \to 0$ (e.g. $k = \sqrt{M}$): k grows slower than M, so the neighborhood still shrinks around the query point and the bias vanishes.

With all three conditions met, both bias and variance go to zero, so the error reaches the irreducible floor R^*.

The 1-NN classifier, in contrast, fixes $k = 1$ and therefore does no averaging at all. Cover & Hart (1967) showed that even so its asymptotic error satisfies

$$R^* \;\le\; R_{\text{1-NN}} \;\le\; 2R^*(1 - R^*) \;\le\; 2R^*, \tag{10.8}$$

so 1-NN never closes the gap to optimality but stays within a factor of two. This is the price of pure variance: minimum bias, no averaging, a permanent gap above R^*.

10.4. Curse of Dimensionality

Goal: Show why distance-based reasoning, and k-NN in particular, breaks down as the feature dimension grows, and what this implies for classifier design.

10.4.1. Volume concentrates at the corners

A ball centered at the origin of a hypercube occupies a vanishing fraction of the cube's volume as d grows; almost all the mass is in the corners.

The volume of a d-dimensional ball of radius r is

$$V_d(r) = \frac{\pi^{d/2}}{\Gamma(d/2 + 1)}\, r^d, \tag{10.9}$$

where $\Gamma(\cdot)$ is the gamma function, the continuous extension of the factorial ($\Gamma(n) = (n-1)!$ for positive integers n). The familiar cases are $V_1(r) = 2r$, $V_2(r) = \pi r^2$, and $V_3(r) = \frac{4}{3}\pi r^3$.

Consider the hypercube $[-a, a]^d$, of volume $(2a)^d$, and an inscribed hypersphere of radius $r = a$ (Fig. 10.3, left). The fraction of the cube's volume contained in the sphere is

$$f_d = \frac{V_d(a)}{(2a)^d} = \frac{\pi^{d/2}}{2^d\,\Gamma(d/2 + 1)} = \frac{\pi^{d/2}}{d\,2^{d-1}\,\Gamma(d/2)}, \qquad \lim_{d\to\infty} f_d = 0,$$

using $\Gamma(d/2 + 1) = (d/2)\,\Gamma(d/2)$. The cube's volume therefore concentrates at its corners rather than at its center.

Example 10.3: In 2-D the sphere fills $\pi/4 \approx 78.5\%$ of the square (Fig. 10.3). By $d = 5$ the fraction is already below 20%, by $d = 10$ it is about $2.5 \cdot 10^{-3}$, and by $d = 15$ it is on the order of 10^{-5}. A Monte-Carlo check (Sec. 1.6), drawing N uniform points in $[-a, a]^d$ and counting the fraction with $\|\mathbf{x}\| \le a$, reproduces f_d to within $1/\sqrt{N}$.

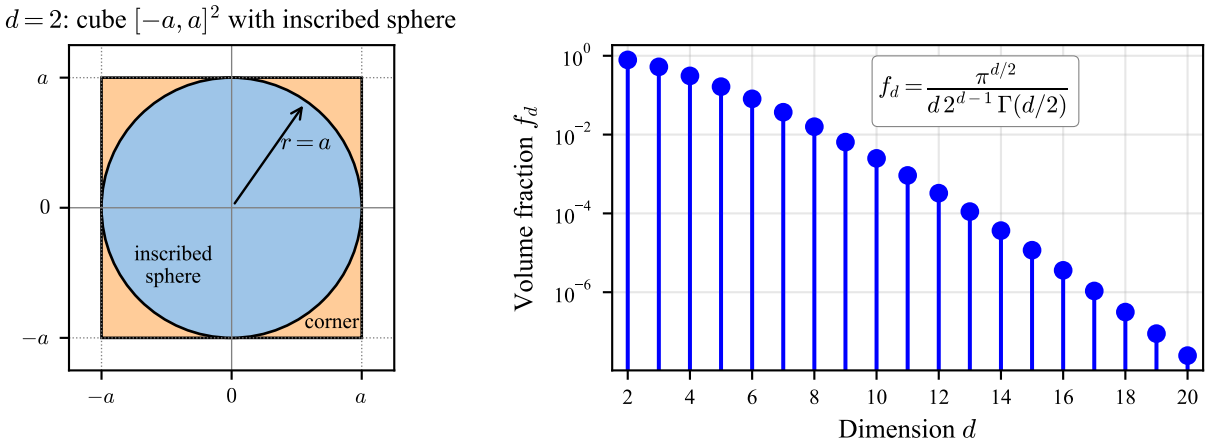


Figure 10.3.: Left: in 2-D the inscribed sphere (blue) fills $\pi/4 \approx 78.5\%$ of the square $[-a, a]^2$; the corner regions (orange) take the rest. Right (log scale): the fraction f_d of hypercube volume contained in the inscribed hypersphere drops by several orders of magnitude as d grows.

k-NN consequence The "neighborhood" is no longer local: the k chosen points span most of the feature space, so the vote reflects the overall class proportions rather than the class composition near the query. In bias-variance terms (Sec. 10.3), this drives *bias* up even at small k, since the de-facto neighborhood is almost global; tuning k downward no longer recovers the low-bias regime that small k provides in low dimensions.

10.4.2. Volume concentrates near the surface

For a hypersphere, almost all volume sits in a thin shell at the boundary; the interior is essentially empty.

Consider two concentric hyperspheres of radii r and $r - \epsilon$ (Fig. 10.4, left). The fraction of the larger sphere's volume (10.9) in the thin shell between them is

$$f_s = \frac{V_d(r) - V_d(r-\epsilon)}{V_d(r)} = 1 - \left(1 - \frac{\epsilon}{r}\right)^d, \qquad \lim_{d\to\infty} f_s = 1.$$

Equivalently, the interior fraction $1 - f_s = (1 - \epsilon/r)^d$ decays geometrically in d.

Example 10.4: **A thin 10% shell takes the volume.** For $\epsilon/r = 0.1$ (a shell occupying only the outer tenth of the radius), the interior fraction is 0.9^d: about 73% at $d = 3$, 35% at $d = 10$, and 12% at $d = 20$ (Fig. 10.4). Data distributed uniformly over a hypersphere therefore clusters near the boundary, leaving interior regions empty.

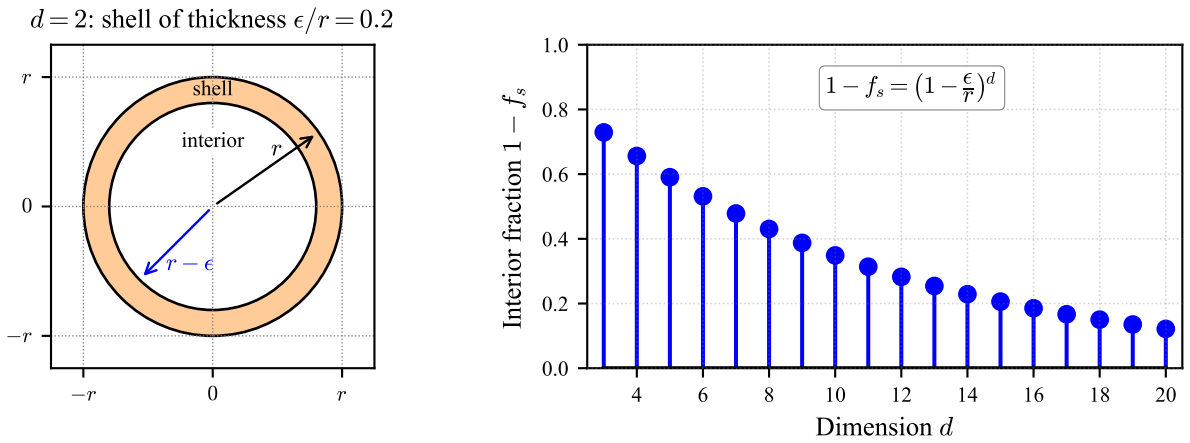


Figure 10.4.: Left: 2-D annular shell of thickness ϵ between the outer disk of radius r and the inner disk of radius $r-\epsilon$. Right: the interior fraction $1-f_s = (1-\epsilon/r)^d$ decays geometrically with dimension, so virtually all volume migrates into the shell.

k-NN consequence. Pairwise distances concentrate: the ratio of farthest to nearest neighbor distance approaches 1, so the ranking of training points by distance becomes nearly arbitrary; the same k points might as well have been chosen at random.

In bias-variance terms (Sec. 10.3), this is a *variance* explosion at every k; different draws of the training set yield different neighbor sets and different predictions at the same query.

10.4.3. Samples live in the tails of the density

For the standard multivariate normal $\mathbf{x} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_d)$ (zero mean, unit variance per coordinate, uncorrelated coordinates), samples cluster far from the mean[1] as d grows; in high dimensions essentially every sample is a tail outlier (Sec. 1.3).

This sounds paradoxical, since the bell curve peaks at the origin. The catch: the chance of landing in a region A equals density times volume,

$$P(\mathbf{x} \in A) = \int_A f(\mathbf{x})\, \mathrm{d}\mathbf{x},$$

and there is far more volume far from the origin than near it. In d dimensions, the volume of a thin shell at radius ρ grows like ρ^{d-1} (the surface area of a d-sphere), while the density drops as $e^{-\rho^2/2}$. So the radial density (the chance of landing at distance ρ, summed over all directions) is

$$g(\rho) \;\propto\; \rho^{d-1} e^{-\rho^2/2},$$

which is largest at $\rho^* = \sqrt{d-1}$. Samples therefore concentrate on a thin shell at distance roughly $\sqrt{d}$ from the origin, where the density itself is already tiny. The origin is the single most likely point; the bulk of the probability sits on the shell.

[1] Because the standard normal is symmetric and unimodal, the mean, the median, and the mode, all coincide at the origin $\mathbf{0}$, so the three terms are interchangeable here.

How concentrated this shell is depends on d. The squared distance $\|\mathbf{x}\|^2$ follows a $\chi^2(d)$ distribution, so for any radius r the probability of landing outside the sphere of radius r is

$$\Pr\big(\|\mathbf{x}\| > r\big) = \Pr\Big(\chi^2(d) > r^2\Big), \qquad \lim_{d\to\infty} \Pr\Big(\chi^2(d) > r^2\Big) = 1,$$

i.e. for any fixed radius, the sample eventually escapes it almost surely as d grows.

Example 10.5: *From bulk to tails as d grows.* Take as reference the density level set $\{\mathbf{x} : f(\mathbf{x}) = 0.01\, f(\mathbf{0})\}$, i.e. the contour where the density equals 1% of its peak value at the origin (Fig. 10.5, left). Since $\ln\big(f(\mathbf{x})/f(\mathbf{0})\big) = -\|\mathbf{x}\|^2/2$, this contour is the sphere of radius $r_{1\%} = \sqrt{-2\ln 0.01} \approx 3.03$, the same numerical threshold in every dimension. Note that "1%" here is the *density value* on the contour, *not* the probability mass outside it. The actual tail probability $P_{\text{tail}}(d) = \Pr\big(\chi^2(d) > -2\ln 0.01\big) = \Pr\big(\chi^2(d) > 9.21\big)$ depends on d:

- $d = 1$: $P_{\text{tail}} = 2(1 - \Phi(3.03)) \approx 0.0024$, only about 1 sample in 400 lies outside, far below 1%. The threshold 3.03σ is much stricter than the actual 1% tail of a 1-D Gaussian (at $\approx 2.58\sigma$), and the 1-D "volume" at radius 3 is tiny, so very little mass reaches that far.
- $d = 2$: $P_{\text{tail}} = 0.01$ exactly. This is a coincidence of $\chi^2(2)$ being exponential: $\Pr\big(\chi^2(2) > c\big) = e^{-c/2}$, and with $c = -2\ln 0.01$ this evaluates to 0.01.
- Between $d = 5$ and $d = 10$ the mass migrates rapidly into the tails (Fig. 10.5, right): the radial volume factor ρ^{d-1} inflates fast enough to push the tail mass well past 1%.
- $d = 10$: more than half the samples lie outside the 1%-of-mode contour; equivalently $\Pr\big(\|\mathbf{x}\| \geq 1.6\big) = 0.99$.
- $d = 20$: $P_{\text{tail}} \approx 0.98$, so almost every sample is outside the contour.
- In very high dimensions, virtually every sample is a tail outlier.

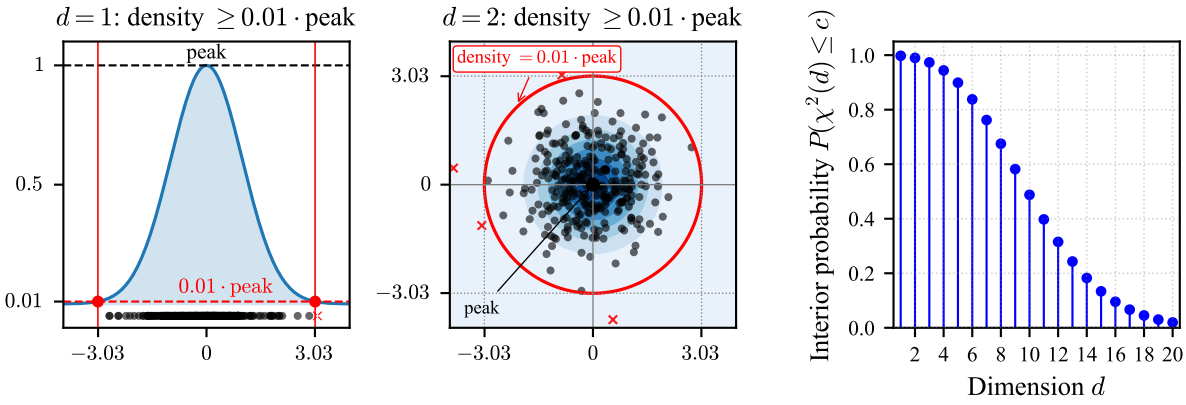


Figure 10.5.: Left: 1-D standard normal density with the 1%-of-mode interval marked in red; almost every sample (black dots on the rug) lies inside, only a handful (red crosses) fall outside. Center: same construction in 2-D, where the interior region is a disk and a few more samples land in the tails. Right: probability that a sample from a d-dimensional standard normal falls *inside* the contour; this interior probability collapses, so beyond $d \approx 10$ virtually every sample is in the tails.

k-NN consequence k-NN guesses the class of a query by looking at the labels of nearby training points. This only works only in the area of the actual training points. When samples concentrate in

the tails, every test query sits in a region where only a few training points landed, so the k nearest neighbors come from far away and do not really reflect the query's surroundings.

In bias-variance terms (Sec. 10.3), small k gives high *variance* (a handful of far-away neighbors are noisy), and large k gives high *bias* (the average drifts toward the global majority); no choice of k works well.

10.4.4. Consequences for Classification

The three phenomena above, corner concentration, shell concentration, and tail concentration, are not specific to k-NN. Every classifier relies on some kind of geometric contrast (distances, densities, or margins), and each kind weakens in its own way as d grows.

- *Distance-based methods* (e.g., k-NN, kernel SVM, RBF networks, kernel density classifiers). Hurt most directly: the contrast between near and far collapses, so neighborhood votes and kernel weights stop carrying information. A learned or supervised metric helps only if the data actually lives on a low-dimensional manifold.
- *Generative and density-based classifiers* (Naive Bayes with continuous features, QDA, GMMs, KDE). Needs sample sizes that grow exponentially in d; tail concentration means every test point sits in a region the training set never sampled densely, so the density is extrapolated rather than estimated.
- *Linear and parametric classifiers* (logistic regression, linear SVM). Geometrically less exposed, but vulnerable to a different face of the curse: when d approaches the sample size M, the data becomes almost surely linearly separable for any labeling, so the training loss can be driven to zero without learning anything. Strong regularization (L_2/ridge, L_1/lasso, early stopping; Sec. 4.9) becomes mandatory rather than optional.
- *Tree-based methods* (decision trees, random forests, gradient boosting). Splits use one coordinate at a time, so they ignore distance concentration entirely, but they still need enough samples per leaf; the chance of selecting a useful split shrinks as the number of noise features dilutes the informative ones.

Practical remedies all reduce the *effective* dimension well below the nominal d:

- *Feature selection* (Ch. 9): keep only the features that carry signal, discard the rest.
- *Dimensionality reduction* (Ch. 11): project the data onto a lower-dimensional subspace (e.g. PCA) that preserves the variance that matters.
- *Regularization* (Sec. 4.9): shrink the parameter space the model can explore, so even in nominally high d the fitted model has few effective degrees of freedom.
- *Learned representations* (deep networks): map raw features to a much lower intrinsic dimension that captures the task structure.
- *Domain-driven feature engineering*: encode prior knowledge directly, replacing many noisy raw features with a few informative ones.

Whichever route is chosen, more samples never hurt, but the required M to undo the curse grows exponentially in d, so doubling the data has far less effect than removing irrelevant features.

10.5. Summary

Hyperparameters:

- the distance metric (Sec. 10.2),
- for Minkowski distance, the exponent p (Sec. 10.2.4), and
- the number of neighbors k (Sec. 10.3).

All three are typically chosen by cross-validation (Sec. 4.5).

Pros.

- Instance-based and non-parametric: no training phase, no model to fit, few hyperparameters, and new data is absorbed by adding points to the index.
- Probabilistic reading: the fraction of class-c neighbors estimates the class empirically.
- Flexible decision boundary: small k tracks fine local structure; large k smooths automatically.
- Strong baseline for small, low-dimensional, well-normalized datasets.

Cons.

- No learned structure: k-NN memorizes, it does not summarize; it cannot share information across queries or identify which features matter.
- High inference cost: $\mathcal{O}(M\,N)$ per query (one N-dimensional distance to each of M training points). Approximate-nearest-neighbor indexes help up to moderate N.
- Memory cost: the entire training set must be kept at inference time.
- Sensitive to feature scale and to irrelevant features; standardization (Sec. 4.7) and feature selection (Ch. 9) are essentially mandatory.
- Sensitive to outliers and label noise, especially at small k.
- Performance degrades quickly in high dimensions (curse of dimensionality); shrink the effective dimension before tuning k.

Further Reading

- Cover and Hart, "Nearest neighbor pattern classification" (*IEEE Trans. Inf. Theory*, 1967): the $R^* \le R_{\text{1-NN}} \le 2R^*(1 - R^*)$ bound used in Sec. 10.3.
- Paradoxical behavior of multidimensional data: interactive MATLAB illustration of the three phenomena in Sec. sec-curse-of-dimensionality.

11. Dimensionality Reduction

Goal: Map N-dimensional data vectors $\mathbf{x}_1, \ldots, \mathbf{x}_M$ into a lower-dimensional representation $\mathbf{z}_1, \ldots, \mathbf{z}_M \in \mathbb{R}^K$ with $K \ll N$, while preserving as much of the structure of the data as possible.

Contents

11.1. Characterization **131**
11.1.1. Local vs. global structure 131
11.1.2. General Features 132
11.2. Principal Component Analysis (PCA) **133**
11.2.1. Mean as a zero-dimensional representation 133
11.2.2. Definition 133
11.2.3. Choosing the number of components 135
11.3. t-SNE **137**
11.4. UMAP **137**
11.5. PaCMAP **137**
11.6. Summary **138**

Common motivations: visualization in two or three dimensions, noise reduction, decorrelation of features, lossy compression, and mitigating the curse of dimensionality before downstream classification or regression.

11.1. Characterization

11.1.1. Local vs. global structure

The methods in this chapter are compared by how well they preserve the *local* and *global* structure of the data:

- *Local structure* is the geometry of small neighbourhoods: which points are close to which, the shape of a single cluster, fine-grained variation within a group. Preserving local structure means that points that were near each other in $\mathbb{R}^N$ remain near each other in $\mathbb{R}^K$.
- *Global structure* is the large-scale layout: relative positions of clusters, overall spread, and the distances between distant points. Preserving global structure means that clusters that were far apart in $\mathbb{R}^N$ remain far apart, and their relative arrangement is maintained.

Both aspects are illustrated on the same five-cluster ring dataset in Fig. 11.1. Panels (a) and (b) define the two notions on the original data; panels (c) and (d) show the two failure modes a low-dimensional embedding can fall into when the goals come into conflict. The trade-off is unavoidable: mapping $\mathbb{R}^N$ to $\mathbb{R}^K$ with $K \ll N$ must discard some pairwise-distance information, and *which* distances survive is dictated by the criterion the embedding optimises.

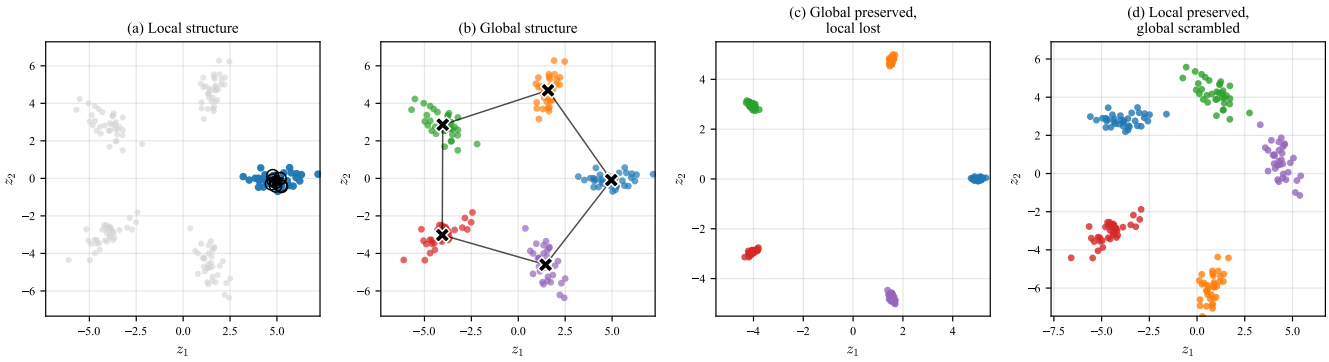


Figure 11.1.: Local vs. global structure on a single five-cluster ring dataset. (a) Local structure: a focal point ($\star$) and its nearest neighbours inside one cluster. (b) Global structure: the five cluster centroids ($\mathbf{X}$) and their ring arrangement. (c) Embedding that preserves the global ring arrangement but collapses each cluster (global preserved, local lost). (d) Embedding that keeps each cluster intact but scrambles their relative positions (local preserved, global distorted).

11.1.2. General Features

Deterministic vs. stochastic fits

A method is *deterministic* if it returns the same embedding every time it is run on a given dataset; it is *stochastic* if the result depends on a random initialisation or a randomised optimiser, so successive runs of the same code on the same data produce different layouts. Determinism matters when the embedding is used as a fixed preprocessing step in a pipeline (the same input must always give the same output) and for the reproducibility of published figures. Stochastic methods are usually made reproducible by fixing a random seed.

Embedding new points

A method *embeds new points* if, after fitting on a training set, it can place unseen samples in the same coordinate system without re-running the optimisation. This is essential when the low-dimensional representation is part of a deployed system (e.g., a learned 2D map onto which test or streaming data must be plotted). Methods that lack this property must be re-fit from scratch whenever the dataset changes, which makes them suitable mainly for one-off visualisation rather than for reusable preprocessing.

Supervised vs. unsupervised

A dimensionality-reduction method is *unsupervised* if it uses only the inputs $\mathbf{x}_1, \ldots, \mathbf{x}_M$ and *supervised* if it also uses the targets $\mathbf{y}_1, \ldots, \mathbf{y}_M$ to choose the embedding. Unsupervised methods optimise a purely geometric criterion (variance, neighbour structure), so they may discard directions that have low variance but are highly informative about $\mathbf{y}$.

11.2. Principal Component Analysis (PCA)

11.2.1. Mean as a zero-dimensional representation

The simplest dimensionality reduction maps every data point to a single point $\mathbf{c} \in \mathbb{R}^N$. The choice that minimizes the sum of squared distances,

$$\mathbf{c}^\star = \arg\min_{\mathbf{c}} \sum_{i=1}^{M} \|\mathbf{x}_i - \mathbf{c}\|^2, \tag{11.1}$$

is the sample mean

$$\boldsymbol{\mu} = \frac{1}{M} \sum_{i=1}^{M} \mathbf{x}_i. \tag{11.2}$$

We use K to denote the dimension of the approximation: a single point has dimension $K = 0$, a line $K = 1$, a plane $K = 2$, and so on. The sample mean is therefore the best $K = 0$ approximation of the dataset (Fig. 11.2).

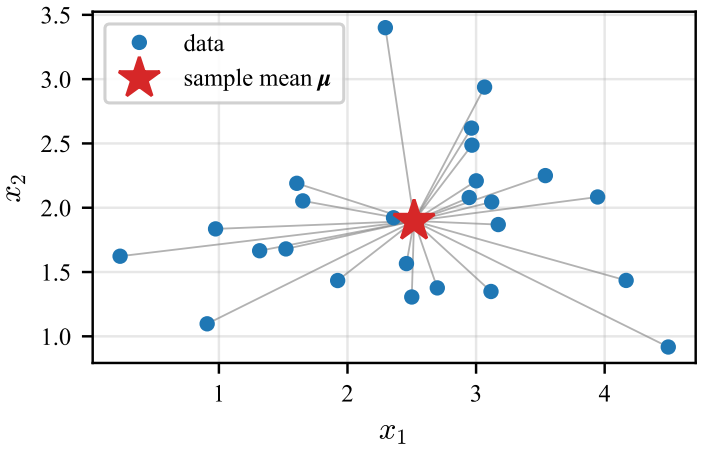


Figure 11.2.: The sample mean $\boldsymbol{\mu}$ minimizes the sum of squared distances to all data points; the best zero-dimensional approximation of the dataset.

11.2.2. Definition

Intuition For $K \geq 1$, PCA fits the data with a K-dimensional subspace through the sample mean (a line for $K = 1$, a plane for $K = 2$, and so on). This subspace is oriented along the directions

in which the data spreads the most. The first such direction is the line that best fits the centered points; the second is perpendicular to the first and captures the next-largest spread; and so on (Fig. 11.3).

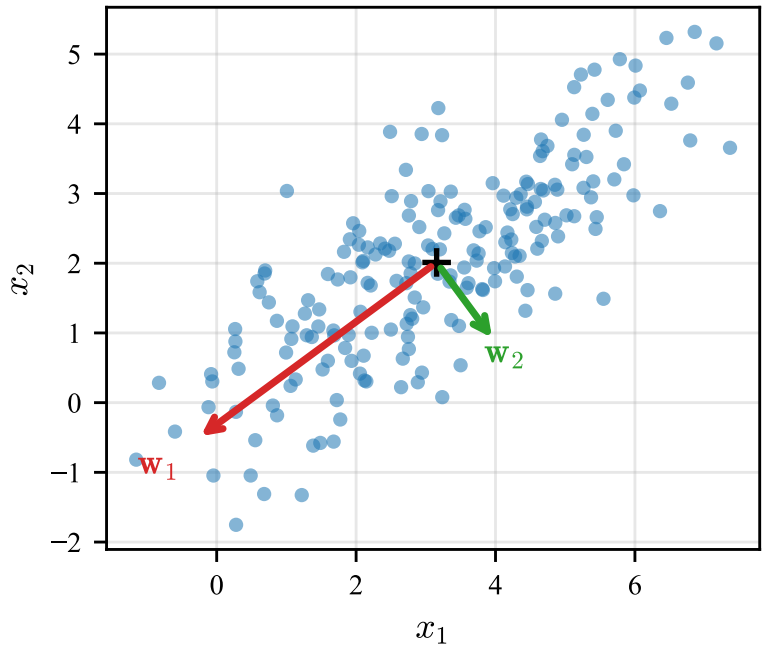


Figure 11.3.: First and second principal axes of a 2D point cloud, drawn at the centroid; arrow lengths are proportional to $\sqrt{\lambda_i}$ of the covariance matrix $\mathbf{S}$.

Objective To turn this picture into a computation, we need a formula for the reconstruction $\hat{\mathbf{x}}_i$ of each data point on the subspace. Write it as $\hat{\mathbf{x}}_i = \mathbf{c} + \mathbf{W}_K \mathbf{z}_i$, where $\mathbf{c} \in \mathbb{R}^N$ is a point the subspace passes through, the columns of $\mathbf{W}_K \in \mathbb{R}^{N \times K}$ are the K direction vectors of the subspace (unit-length and mutually perpendicular), and $\mathbf{z}_i \in \mathbb{R}^K$ holds the coordinates of point i within it. With this parametrization, the same squared-distance criterion as in the $K = 0$ case becomes

$$(\mathbf{c}^\star, \mathbf{W}_K^\star, \{\mathbf{z}_i^\star\}) = \underset{\substack{\mathbf{c},\, \mathbf{W}_K,\, \{\mathbf{z}_i\} \\ \mathbf{W}_K^T \mathbf{W}_K = \mathbf{I}_K}}{\arg\min} \sum_{i=1}^{M} \left\| \mathbf{x}_i - (\mathbf{c} + \mathbf{W}_K \mathbf{z}_i) \right\|^2. \tag{11.3}$$

Setting $K = 0$ removes the $\mathbf{W}_K \mathbf{z}_i$ term and recovers the mean problem.

Solution Solving (11.3) splits into two steps: locating the subspace (choosing $\mathbf{c}^\star$) and orienting it (choosing $\mathbf{W}_K^\star$). Both are read off the data through the sample mean and the sample covariance matrix. Let $\mathbf{X} \in \mathbb{R}^{M \times N}$ be the data matrix (rows are samples). Center it by subtracting the sample mean,

$$\tilde{\mathbf{X}} = \mathbf{X} - \mathbf{1}\boldsymbol{\mu}^T, \qquad \boldsymbol{\mu} = \frac{1}{M} \sum_{i=1}^{M} \mathbf{x}_i, \tag{11.4}$$

and form the sample covariance matrix

$$\mathbf{S} = \frac{1}{M-1} \tilde{\mathbf{X}}^T \tilde{\mathbf{X}}. \tag{11.5}$$

The best subspace passes through the sample mean, so $\mathbf{c}^\star = \boldsymbol{\mu}$. Its directions are the top K eigenvectors of $\mathbf{S}$: $\mathbf{W}_K^\star = [\mathbf{w}_1, \dots, \mathbf{w}_K]$, where the eigenvectors $\mathbf{w}_1, \dots, \mathbf{w}_N$ are ordered by decreasing eigenvalue $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_N \geq 0$. Each eigenvalue λ_i equals the variance of the data along $\mathbf{w}_i$, so picking the top K eigenvectors picks the K directions of largest spread, which matches the geometric picture from the intuition above. The eigendecomposition itself is reviewed in the appendix.

Encoding and decoding Once the directions $\mathbf{W}_K$ are fixed, each data point is encoded as

$$\mathbf{z} = \mathbf{W}_K^T(\mathbf{x} - \boldsymbol{\mu}) \in \mathbb{R}^K \tag{11.6}$$

and reconstructed as

$$\hat{\mathbf{x}} = \boldsymbol{\mu} + \mathbf{W}_K\mathbf{z}. \tag{11.7}$$

For $K = 0$ this collapses to $\hat{\mathbf{x}} = \boldsymbol{\mu}$, the zero-dimensional approximation. For $K \geq 1$, no other K-dimensional subspace through $\boldsymbol{\mu}$ gives a smaller reconstruction error in (11.3).

Computation. In practice PCA is computed from the singular value decomposition of $\tilde{\mathbf{X}}$ rather than by forming $\mathbf{S}$ explicitly. SVD is more numerically stable, especially when features are nearly collinear or N is large.

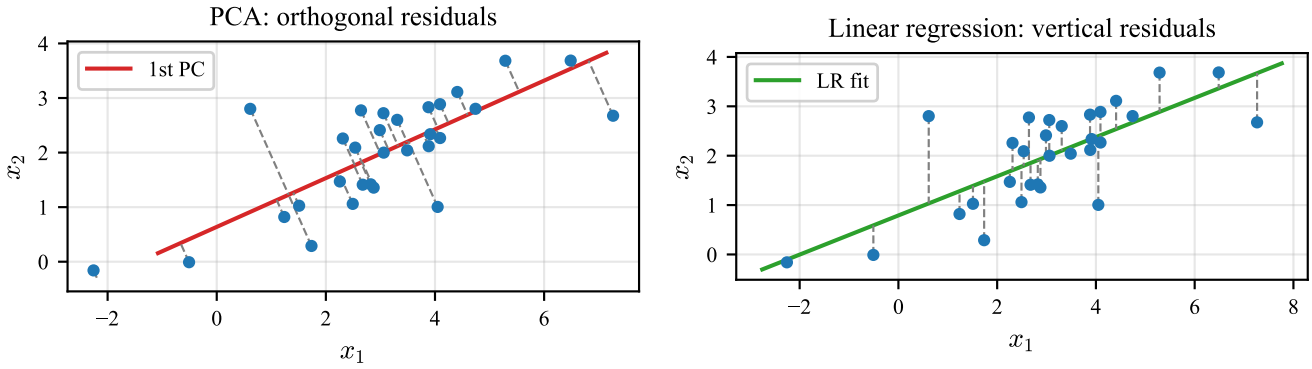


Figure 11.4.: PCA minimises orthogonal (perpendicular) reconstruction distance; linear regression minimises vertical residuals. The two fits coincide only in special cases.

The PCA fit is not the same as the linear regression fit. PCA minimises the perpendicular distance from each point to the fitted line (or subspace), treating all coordinates symmetrically. Linear regression minimises the vertical distance, treating x_2 as a function of x_1, and changes if the roles of the variables are swapped.

PCA can also be viewed as a linear autoencoder with a single hidden layer of width K and tied weights; this connection is taken up again in the chapter on neural networks.

11.2.3. Choosing the number of components

The fraction of variance explained by component i is $\lambda_i / \sum_j \lambda_j$. The cumulative explained variance by the top K components is

$$\mathrm{EV}(K) = \frac{\sum_{i=1}^{K} \lambda_i}{\sum_{j=1}^{N} \lambda_j} \times 100\%. \tag{11.8}$$

Two common heuristics for picking K are: (i) the smallest K such that $\mathrm{EV}(K)$ exceeds a target threshold (typically 90% or 95%), and (ii) the elbow of the scree plot, where the variance ratio drops sharply (Fig. 11.5).

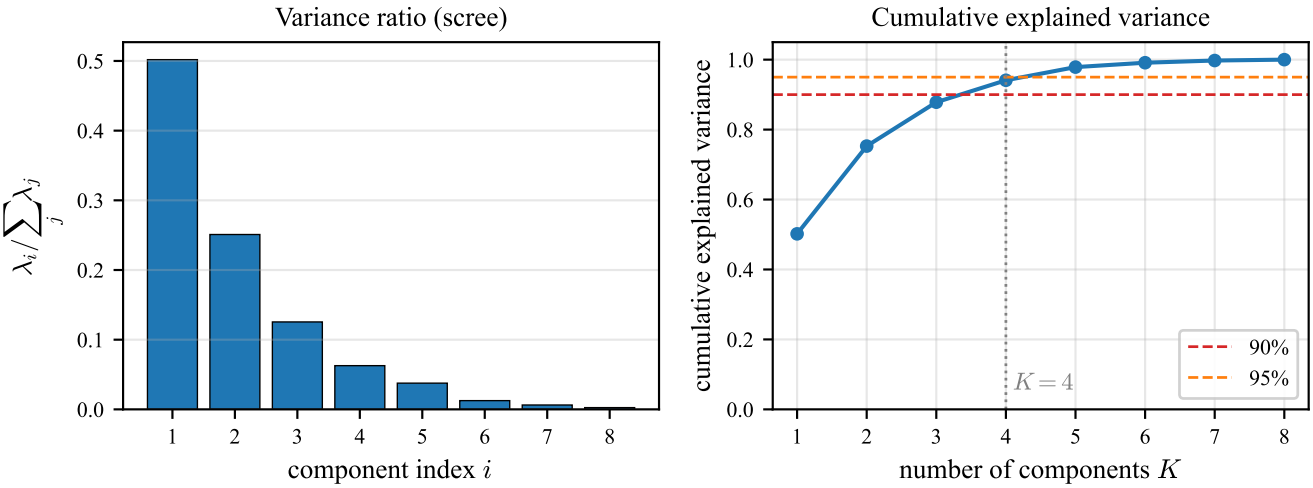


Figure 11.5.: Left: variance ratio per component (scree plot). Right: cumulative explained variance with 90% and 95% thresholds; here the 90% threshold is reached at $K = 4$.

Example 11.1 (Choosing K): For the eigenvalue spectrum $\lambda = (4, 2, 1, 0.5, 0.1)$ with total $\sum_j \lambda_j = 7.6$, the variance ratios are $(0.526, 0.263, 0.132, 0.066, 0.013)$ and the cumulative explained variance is

$$\mathrm{EV}(1) = 52.6\%,\ \mathrm{EV}(2) = 78.9\%,\ \mathrm{EV}(3) = 92.1\%,\ \mathrm{EV}(4) = 98.7\%.$$

The 90% threshold is met at $K = 3$.

PCA is deterministic, embeds new points trivially via $\mathbf{z}_{\text{new}} = \mathbf{W}_K^\top(\mathbf{x}_{\text{new}} - \boldsymbol{\mu})$, and is unsupervised: it does not use the target $\mathbf{y}$, so it may discard directions that are low-variance but discriminative.

> The number of components K is the only hyperparameter of PCA. It is typically chosen by an explained-variance threshold (e.g. 90% or 95%) or by the elbow of the scree plot, and validated with cross-validation when PCA feeds a downstream supervised model.

Practical notes

- *Scale sensitivity:* when input features are in different units, standardise them first (or apply PCA to the correlation matrix instead of the covariance matrix); otherwise the largest-variance feature dominates.
- *Use in supervised pipelines:* validate the chosen K with cross-validation rather than picking it from the scree plot alone.
- *Lossy compression and denoising:* dropping the small-eigenvalue components removes the directions of smallest variance, which often correspond to noise (eigenfaces are a classical example).
- *For visualization and clustering:* PCA tends to preserve global (inter-cluster) geometry but distorts local (intra-cluster) structure.

11.3. t-SNE

t-SNE (t-distributed stochastic neighbor embedding) is a popular tool for visualizing complex data in 2D or 3D. Instead of trying to keep the exact mathematical distances between points like PCA does, t-SNE simply tries to group similar data points together while pushing dissimilar points far apart.

t-SNE is stochastic and does not embed new points: each call re-runs the optimization from a random initialisation, and the resulting layout cannot be reused to place unseen samples without refitting on the combined data.

Hyper-parameter: Perplexity The main control knob is *perplexity*, typically set between 5 and 50. Think of this as the number of close neighbors each point cares about. If your plot looks like a cloud, adjusting the perplexity is the best way to uncover hidden structure.

Because t-SNE forces clusters to separate, it can be deceiving:

- *Don't trust the gaps:* t-SNE exaggerates the empty space between clusters. The physical distance between two different clusters on the plot is meaningless.
- *Cluster size doesn't matter:* A large, spread-out cluster on the plot isn't necessarily more diverse than a tight, dense one. t-SNE heavily distorts the density of the data.

11.4. UMAP

UMAP (uniform manifold approximation and projection) is a newer alternative to t-SNE. It targets the same goal (creating 2D or 3D plots of complex data) but uses a more efficient approach that gives it several major practical advantages.

Improvements over t-SNE:

- Faster
- Better preserves global structure.
- Can place unseen samples on a previously fitted map without refitting.

Hyper-parameters UMAP gives you two main ways to control the final plot:

- `n_neighbors`: Similar to t-SNE's perplexity. A low number focuses on grouping tiny local details, while a high number forces the algorithm to look at the broader, global structure.
- `min_dist`: Controls how tightly packed the points are allowed to be. Set it low (e.g., 0.1) if you want distinct, separated clusters; set it higher (e.g., 0.5) if you want to see the individual points without them overlapping too much.

11.5. PaCMAP

PaCMAP (pairwise controlled manifold approximation and projection) is a more recent visualisation method that explicitly aims to preserve both local clusters and global layout at the same time.

It is built on top of the same general idea as UMAP, but with a different recipe for choosing which pairs of points the optimisation should care about.

The distinguishing idea of PaCMAP is to balance three types of point pairs in the loss function:

- *Near pairs:* a few close neighbours of each point, used to keep local clusters tight.
- *Mid-near pairs:* pairs that are not the closest, but not far either, used to anchor the medium-scale layout.
- *Further pairs:* random distant pairs, used to push unrelated points apart.

By tuning the relative weight of these three types over the course of optimisation, PaCMAP recovers fine cluster structure without losing the big picture.

Hyper-parameter The main control is `n_neighbors` (size of the local neighbourhood, similar in spirit to UMAP's `n_neighbors` or t-SNE's perplexity). Defaults work well in most cases; the implementation also exposes ratios that control how many mid-near and further pairs are used per near pair, which usually do not need to be changed.

PaCMAP is stochastic, unsupervised, and embeds new points into an existing layout . As with the other nonlinear methods, absolute distances in a PaCMAP plot should not be over-interpreted, but the relative arrangement of clusters tends to be more trustworthy than in t-SNE.

11.6. Summary

How each method handles the local/global trade-off. The local vs. global distinction introduced at the start of the chapter (Fig. 11.1) is dictated by the criterion each method optimises. A loss built on overall variance or all pairwise distances (as in PCA) is dominated by the largest distances, so global layout is retained but within-cluster geometry is spent first when a direction is dropped. A loss built on neighbour relations (as in t-SNE) penalises only short-range errors, so within-cluster structure is preserved, but the optimiser is free to rearrange clusters because faraway distances do not enter the loss. Hybrid methods such as UMAP and PaCMAP mix near-neighbour and farther-pair terms in the loss, which is why their plots tend to keep clusters tight *and* their relative arrangement readable.

PCA is a linear projection, so it preserves global structure (large pairwise distances and overall spread) but compresses everything along the dropped directions, which often destroys fine local detail. t-SNE optimises a local objective, so it reveals clusters and within-cluster structure clearly but distorts global geometry: cluster sizes and inter-cluster distances in a t-SNE plot are not meaningful. UMAP sits between the two, recovering local structure similarly to t-SNE while retaining more of the global layout.

Figure 11.6 illustrates this on the handwritten-digits dataset (10 classes, 64 features): PCA shows heavy class overlap, while t-SNE and UMAP both separate the digit classes into compact clusters. Table 11.1 summarises how the four methods compare on the properties introduced earlier in Sec. 11.1.

Choosing a method

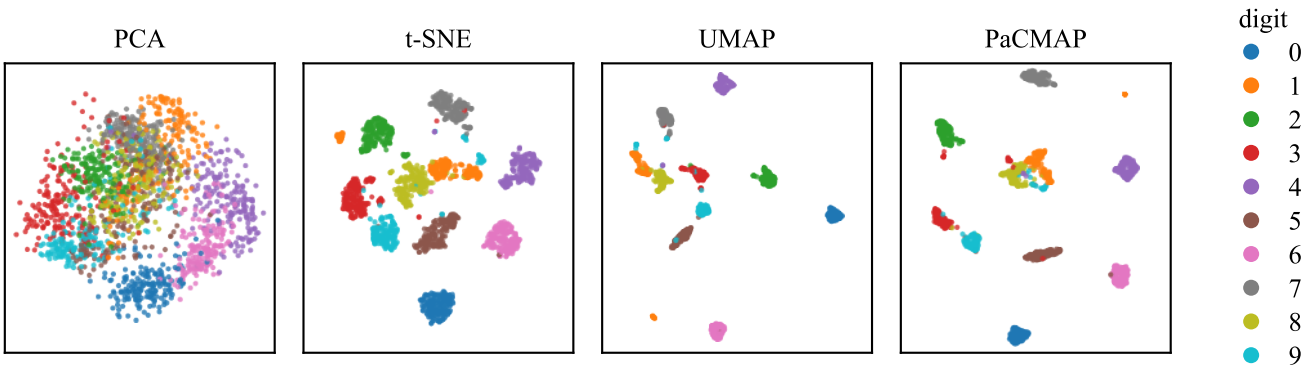


Figure 11.6.: The same handwritten-digits dataset (10 classes, 64 features) embedded into 2D by PCA, t-SNE, and UMAP. Colours encode the digit label.

Table 11.1.: Comparison of PCA, t-SNE, UMAP, and PaCMAP.

Property	PCA	t-SNE	UMAP	PaCMAP
Mapping	linear	nonlinear	nonlinear	nonlinear
Deterministic	yes	no	no	no
Supervised	no	no	no/yes (2)	no
Preserves global	yes	no	partially	yes
Preserves local	no	yes	yes	yes
Embeds new points	yes (trivial)	no	yes	yes
Complexity (1)	$\mathcal{O}(\min(MN^2, M^2N))$	$\mathcal{O}(M^2)$ / $\mathcal{O}(M \log M)$	$\sim \mathcal{O}(M^{1.14})$	$\mathcal{O}(M)$

(1) Complexity with M samples and N input features.
(2) UMAP is unsupervised by default; a supervised variant exists that uses labels $\mathbf{y}$ to guide the embedding.

- Use PCA as a default first step for small datasets. Its fit is a single SVD (seconds on $M, N \sim 10^4$, scaling as $\mathcal{O}(\min(MN^2, M^2N))$), it has no random seed, and the same fitted projection $\mathbf{W}_K$ can be applied to streaming or test data. Use it as a preprocessor that decorrelates features and trims dimension before regression, classification, or clustering, and as a quick sanity-check plot. Its blind spots are that it is linear (it cannot unfold curved manifolds) and unsupervised (it may drop low-variance directions that are discriminative for $\mathbf{y}$).
- Use t-SNE, UMAP, or PaCMAP when the goal is a 2D or 3D plot that reveals cluster structure that PCA hides. Prefer *t-SNE* only for small datasets (M up to a few thousand) where you mostly care about whether clusters exist. Prefer *UMAP* as the modern default for visualisation: it scales to millions of points, retains more of the global layout than t-SNE, and can embed new samples without refitting. Prefer *PaCMAP* when the relative arrangement of clusters carries meaning (hierarchies, smooth gradients between groups). In all three cases, treat absolute distances, cluster sizes, and inter-cluster gaps as not meaningful, fix the random seed for reproducibility, and verify that the layout is stable across a few seeds and hyper-parameter values before drawing conclusions.

Further Reading

- Matlab Uniform Manifold Approximation and Projection (UMAP) for dimension reduction

12. Classification Performance Metrics

Goal: Quantify the performance of a binary classifier on a test dataset.

Definitions:

- $\mathbf{y}$ - target values vector of the test database, $\mathbf{y} \in \mathbb{R}^M$
- $\hat{\mathbf{y}}$ - predicted value, $\hat{\mathbf{y}} \in \mathbb{R}^M$, output of some classifier $\hat{\mathbf{y}} = f_{\mathbf{w}}(\mathbf{X})$.

Typically, in binary classification, $y_i \in \{0, 1\}$.

Contents

12.1. Definitions . . . **142**
12.2. Confusion matrix . . . **142**
12.3. Performance Metrics . . . **143**
12.3.1. Accuracy . . . 143
12.3.2. Precision . . . 143
12.3.3. Recall (sensitivity) . . . 144
12.3.4. Specificity . . . 145
12.3.5. F_1-score . . . 145
12.3.6. Balanced accuracy . . . 145
12.3.7. Summary . . . 146
12.4. Imbalanced Dataset . . . **146**
12.4.1. Definition . . . 146
12.4.2. Majority classifier . . . 148
12.4.3. Small or imbalanced datasets . . . 148
12.5. Metrics for Probabilistic Classifiers (ROC & AUC) . . . **149**
12.5.1. Receiver Operating Characteristics (RoC) . . . 149
12.5.2. Area Under Curve (AUC) . . . 150
12.6. Multi-class performance . . . **150**
12.6.1. Per-class performance . . . 152
12.6.2. Macro-averaging . . . 153
12.6.3. Micro-averaging . . . 154
Further reading . . . **156**

12.1. Definitions

Goal: Classification between two groups (only).

Basic terminology:

- '1' – positive group or result
- '0' – negative group or result
- Y – actual class
- $\hat{Y}$ – predicted class
- 'True' – prediction matches the actual class, $\hat{Y} = Y$ (correct classification)
- 'False' – prediction does not match the actual class, $\hat{Y} \neq Y$ (incorrect classification)

Positive/negative terminology is rather arbitrary. Typically, the result of interest is termed positive.

12.2. Confusion matrix

Goal: Summarize classification results of a test set of a particular database.

The summarization is in the for m of a 2D non-normalized histogram of $(Y, \hat{Y})$.

		Predicted values	
		Positive, $\hat{Y} = 1$	**Negative**, $\hat{Y} = 0$
Actual values	**Positive**, $Y = 1$	**TP** True Positive $Y = 1, \hat{Y} = 1$	**FN** False Negative $Y = 1, \hat{Y} = 0$
	Negative, $Y = 0$	**FP** False Positive $Y = 0, \hat{Y} = 1$	**TN** True Negative $Y = 0, \hat{Y} = 0$

Figure 12.1.: Confusion matrix. Note, sometimes, transposed representation is used.

The (test) database has M values, among them:

- TP + FN positive values
- FP + TN negative values

This is the most common way to summarize the performance of a particular classifier on a particular dataset. It can be easily extended for multi-class classifiers.

12.3. Performance Metrics

Goal: Characterization is useful to compare classifiers and/or performance on different datasets.

12.3.1. Accuracy

Goal: The most intuitive metric, fraction of $Y = \hat{Y}$, among all the classification results, $\Pr\left(Y = \hat{Y}\right)$.

$$\begin{aligned} \text{Accuracy} &= \frac{\text{correct predictions}}{\text{total predictions}} \\ &= \frac{TP + TN}{TP + TN + FP + FN} \end{aligned} \tag{12.1}$$

Example 12.1: Covid antibody (fast non-PCR) test performance. The example includes test statistics of 239 participants [7], as presented below.

		Predicted	
		Yes	No
Actual	Yes	141	67
	No	0	31

The resulting accuracy is

$$\text{Accuracy} = \frac{141 + 31}{239} = 0.7196652 \approx 72.0\% \tag{12.2}$$

In the example, FN=67 is a bad performance, and FP=0 is probably something good. However, accuracy does not reflect the discrepancy between these two. Additional metrics are used to quantify these aspects.

12.3.2. Precision

Goal: Proportion of positive classification that is actually correct, $\Pr\left(Y = 1 | \hat{Y} = 1\right)$.

From the probability theory,

$$\Pr\left(Y = 1 | \hat{Y} = 1\right) = \frac{\Pr\left(Y = 1, \hat{Y} = 1\right)}{\Pr\left(\hat{Y} = 1\right)} \tag{12.3}$$

$$\Pr\left(\hat{Y} = 1\right) = \Pr\left(\hat{Y} = 1, Y = 0\right) + \Pr\left(\hat{Y} = 1, Y = 1\right) \tag{12.4}$$

$$\text{Precision} = \frac{TP}{FP + TP} = \frac{\text{Correctly predicted 1's}}{\text{All predicted 1's}} \tag{12.5}$$

TP = Correctly predicted 1's
TP + FP = All predicted 1's

Term	Radar Interpretation
Accuracy	Percentage of all correctly identified as planes or not planes
Precision	Among all classified as planes, the portion that is correctly classified as planes
Recall (sensitivity)	Among all existing planes, portion of correctly classified as planes
Specificity	Among all actual non-planes, portion of correctly classified as non-planes

Table 12.1.: Radar interpretation of the classification metrics.

Example 12.1: Back to the previous example,

$$\text{Precision} = \frac{141}{0+141} = 1 = 100\% \tag{12.6}$$

The high value of the precision is due to the low FP. From the medical point of view, all positive results are actually positive. Whoever was identified by this test as Covid-positive is really positive.

12.3.3. Recall (sensitivity)

Goal: Proportion of positives identified correctly, $\Pr\left(\hat{Y}=1|Y=1\right)$.

From the probability theory,

$$\Pr\left(\hat{Y}=1|Y=1\right) = \frac{\Pr\left(Y=1,\hat{Y}=1\right)}{\Pr(Y=1)} \tag{12.7}$$

$$\Pr(Y=1) = \Pr\left(Y=1,\hat{Y}=0\right) + \Pr\left(Y=1,\hat{Y}=1\right) \tag{12.8}$$

$$\text{Recall} = \frac{TP}{TP+FN} = \frac{\text{Correctly predicted 1's}}{\text{Actual 1's}} \tag{12.9}$$

Medical meaning: portion of correctly classified ill among all the ill.

Example 12.1: Back to the previous example,

$$\text{Recall} = \frac{141}{141+67} = 0.678 = 67.8\% \tag{12.10}$$

The low value of the recall is due to the high FN. From the medical point of view, among all actually positive (ill) individuals, only 67.8% were correctly identified.

12.3.4. Specificity

Goal: Proportion of negatives identified correctly, $\Pr\left(\hat{Y} = 0 | Y = 0\right)$.

$$\text{Specificity} = \frac{TN}{FP + TN} = \frac{\text{Correctly predicted 0's}}{\text{Actual 0's}} \tag{12.11}$$

Medical meaning: portion of classified healthy among all the healthy.

Example 12.1: Back to the previous example,

$$\text{Specificity} = \frac{31}{0 + 31} = 1 = 100\% \tag{12.12}$$

From the medical point of view, all negative results are really negative.

12.3.5. F_1-score

Goal: Combination of precision and recall.

The harmonic mean between precision and recall,

$$F_1 = \frac{2}{\frac{1}{recall} + \frac{1}{precision}} = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \tag{12.13}$$

Example 12.1: Back to the previous example,

$$F_1 = \frac{141}{141 + \frac{1}{2}(0 + 67)} = 0.808 = 80.8\% \tag{12.14}$$

12.3.6. Balanced accuracy

Goal: An accuracy variant that gives each class equal weight, making it robust to class imbalance.

The arithmetic mean of recall and specificity, i.e. the mean of the per-class recall of the positive and negative classes,

$$\text{BACC} = \frac{1}{2}(\text{Recall} + \text{Specificity}) = \frac{1}{2}\left(\frac{TP}{TP + FN} + \frac{TN}{TN + FP}\right). \tag{12.15}$$

When the two classes are balanced and equally well classified, BACC coincides with ordinary accuracy. When one class dominates, BACC no longer rewards a classifier that simply predicts the majority class: such a classifier scores about 0.5 regardless of the imbalance.

Example 12.1: Back to the previous example,

$$\text{BACC} = \frac{1}{2}(0.678 + 1.0) = 0.839 = 83.9\% \tag{12.16}$$

The value sits between the poor recall (67.8%) and the perfect specificity (100%), giving a more balanced picture than the 72.0% accuracy.

		Predicted	
		Yes	No
Actual	Yes	55	5
	No	5	35

Example 12.2: The logistic regression classifier from Figs. 8.1 and 8.6 has the following confusion matrix:

The resulting metrics are

$$\begin{aligned}
\text{Accuracy} &= \frac{55+35}{100} = 0.9 = 90\% \\
\text{Precision} &= \frac{55}{5+55} = \frac{55}{60} \approx 91.7\% \\
\text{Recall} &= \frac{55}{55+5} = \frac{55}{60} \approx 91.7\% \\
\text{Specificity} &= \frac{35}{5+35} = \frac{35}{40} = 87.5\% \\
F_1 &= \frac{55}{55+\frac{1}{2}(5+5)} = \frac{55}{60} \approx 91.7\% \\
\text{BACC} &= \frac{1}{2}(0.917+0.875) = 0.896 = 89.6\%
\end{aligned} \tag{12.17}$$

Note, since $FP = FN$, precision, recall, and F_1 are all equal.

12.3.7. Summary

12.4. Imbalanced Dataset

12.4.1. Definition

Imbalanced dataset: Dataset with significant differences between the numbers of labels of each class. The following examples present a few problems related to imbalanced datasets.

Example 12.3: Let's take a dataset with 1000 samples:

- 990 samples labeled '0'
- 10 samples labeled '1'

What are the performance metrics of the classifier that always predicts $\hat{Y} = 0$?

Solution: The resulting confusion matrix is

Metric	Formula	Question answered
Accuracy	$\frac{TP+TN}{TP+TN+FP+FN}$	Of all predictions, what fraction is correct?
Precision	$\frac{TP}{TP+FP}$	Of all samples predicted positive, what fraction is actually positive?
Recall (sensitivity)	$\frac{TP}{TP+FN}$	Of all actual positives, what fraction is correctly identified?
Specificity	$\frac{TN}{TN+FP}$	Of all actual negatives, what fraction is correctly identified?
F_1	$\frac{\cdot TP}{\cdot TP+\frac{1}{2}(FP+FN)}$	Harmonic mean of precision and recall.
Balanced accuracy	$\frac{1}{2}(\text{Recall}+\text{Specificity})$	Mean per-class recall; accuracy that is robust to class imbalance.

Table 12.2.: Summary of binary classification performance metrics.

		Predicted	
		Yes	No
Actual	Yes	0	10
	No	0	990

and the resulting quantities are

$$\begin{aligned}
\text{Accuracy} &= \frac{990}{1000} = 0.99 = 99\% \\
\text{Precision} &= \frac{TP}{FP+TP} = \frac{0}{0+0} = \text{Undefined} \\
\text{Recall} &= \frac{TP}{TP+FN} = \frac{0}{0+10} = 0 \\
\text{Specificity} &= \frac{TN}{FP+TN} = \frac{990}{1000} = 0.99 = 99\% \\
F_1 &= \frac{TP}{TP+\frac{1}{2}(FP+FN)} = \frac{0}{\cdots} = 0 \\
\text{BACC} &= \frac{1}{2}(\text{Recall}+\text{Specificity}) = \frac{1}{2}(0+0.99) = 0.495 \approx 49.5\%
\end{aligned} \tag{12.18}$$

The balanced accuracy of 49.5% exposes the failure that the 99% accuracy hides: the classifier is no better than a coin toss once both classes are weighted equally.

- Note, accuracy is insufficient metrics!

- Note, while the convention is to label '1' for the most important class outcome, sometimes it is interchangeable.

12.4.2. Majority classifier

Majority classifier is a classifier that always predicts the most frequent class in the dataset, as in Example 12.3.

Despite achieving high accuracy on imbalanced data, a majority classifier has zero recall and zero F_1 for the minority class, and its balanced accuracy is about 0.5. It is commonly used as a baseline: any useful classifier should outperform it on metrics beyond accuracy.

The accuracy of a majority classifier equals the *majority-class rate*,

$$J_{\text{majority-class}} = \frac{\max(M_0, M_1)}{M} \tag{12.19}$$

where M_0 and M_1 are the class counts and M is the total number of samples.

12.4.3. Small or imbalanced datasets

The same issue arises in two related situations: the test set itself is small, or the test set is large but one class is heavily under-represented so that the per-class subset is small. In either case, a per-class metric is estimated from very few samples and is therefore subject to high sampling variability. The observed value can deviate substantially from the true classifier performance purely by chance.

The following example quantifies this gap for a class with only ten samples.

Example 12.4: Consider the dataset from Example 12.3, in which the minority class contains only $n = 10$ positive samples. Assume a classifier whose *true* per-sample accuracy on class '1' is $p = 0.8$. What is the probability that it correctly classifies 6 or fewer of these 10 positive samples (i.e. that the observed recall is at most 60%)?

Solution: Each classification is an independent Bernoulli trial with success probability $p = 0.8$, so the number of correct classifications follows a binomial distribution, $X \sim \text{Bin}(n = 10, p = 0.8)$. The cumulative probability is

$$\Pr(X \leq 6) = \sum_{k=0}^{6} \binom{10}{k} p^k (1-p)^{10-k} \approx 12.09\%.$$

Thus there is a ≈12% chance of observing a recall of 60% or less, even though the true accuracy is 80%. Conversely, $\Pr(X = 10) \approx 10.74\%$: a perfect observed recall is plausible by chance, even for an imperfect classifier.

This is fundamentally a problem of *confidence* in performance evaluation: a single point estimate computed from a small (sub-)set says little about the underlying classifier. A formal treatment via confidence intervals is out of scope for this chapter; the practical takeaway is that such estimates must be interpreted with caution, and small differences between classifiers on small or imbalanced datasets should not be over-interpreted.

Anomaly detection Anomaly detection is a sub-field of classification where the class of interest (the anomaly) is extremely rare compared to the normal class. Examples include fraud detection, network intrusion detection, and equipment failure prediction. The extreme class imbalance (e.g., 1:1 000) makes accuracy meaningless and amplifies the small dataset problem for the minority class. Precision and recall are the primary evaluation metrics in this setting.

12.5. Metrics for Probabilistic Classifiers (ROC & AUC)

Goal: Quantify performance of classifiers with probabilistic loss (probabilistic predictions). This methods extent the methods in Sec. 14.1.

With a probabilistic classifier, the output is the estimated probability of the positive class,

$$\Pr\big(\hat{y} = 1 \mid \mathbf{x}\big) = f_{\mathbf{w}}(\mathbf{x}) \tag{12.20}$$

12.5.1. Receiver Operating Characteristics (RoC)

Goal: Quantify the trade-off between confusion matrix elements as a function of the decision threshold and visualize classifier performance across all thresholds.

The binary decision is obtained by comparing $f_{\mathbf{w}}(\mathbf{x})$ with a threshold thr:

$$\hat{y} = \begin{cases} 1 & f_{\mathbf{w}}(\mathbf{x}) \geq \mathsf{thr} \\ 0 & f_{\mathbf{w}}(\mathbf{x}) < \mathsf{thr} \end{cases} \tag{12.21}$$

The default threshold is $\mathsf{thr} = 0.5$. Changing thr shifts the trade-off between TP, FP, FN, and TN, and therefore changes all derived metrics (precision, recall, specificity, F_1).

The RoC curve plots the following two quantities as thr varies from 0 to 1:

- **True Positive Rate (TPR)**: synonym for recall,

$$TPR = \frac{TP}{TP + FN} = \text{recall} \tag{12.22}$$

- **False Positive Rate (FPR)**: complement of specificity,

$$FPR = \frac{FP}{FP + TN} = 1 - \text{specificity} \tag{12.23}$$

As $\mathsf{thr} \to 0$, everything is classified as positive ($TPR \to 1$, $FPR \to 1$). As $\mathsf{thr} \to 1$, everything is classified as negative ($TPR \to 0$, $FPR \to 0$).

RoC is a legacy term from the field of detector theory and communication system theory.

12.5.2. Area Under Curve (AUC)

Goal: Quantify threshold-independent performance with a single scalar.

AUC: AUC is the area under the RoC curve.

Range: A random (coin-toss) classifier has $\mathsf{AUC} = 0.5$ and an ideal classifier has $\mathsf{AUC} = 1$. All practical classifiers fall in the range $0.5 \leq \mathsf{AUC} \leq 1$.

The relationship between classifier quality and the RoC curve is illustrated in Fig. 12.2a. When the two class distributions (positive and negative) are well separated, the classifier achieves high TPR at low FPR, producing a RoC curve that hugs the top-left corner ($\mathsf{AUC} \to 1$). As the distributions overlap, the FN and FP regions grow, and the RoC curve shifts toward the diagonal ($\mathsf{AUC} \to 0.5$).

The choice of threshold thr controls which part of the overlap region is assigned to each class. Lowering thr classifies more samples as positive, increasing TP but also FP. Raising thr increases TN but also FN. Fig. 12.2b illustrates this trade-off: moving the threshold left or right redistributes errors between FN and FP while tracing the RoC curve.

Advantages:

- **Scale-invariant**: AUC measures how well predictions are ranked, rather than their absolute values.
- **Threshold-invariant**: AUC summarizes performance across all thresholds, without requiring a specific threshold choice.

Limitations:

- **Scale invariance** may be undesirable when well-calibrated probabilities are needed (e.g., for risk assessment), since AUC is insensitive to the predicted probability values.
- **Threshold invariance** may be undesirable when the application requires a specific trade-off between false negatives and false positives. For example, in spam detection, minimizing false positives (legitimate email marked as spam) is more important than minimizing false negatives. AUC does not capture such asymmetric cost preferences.

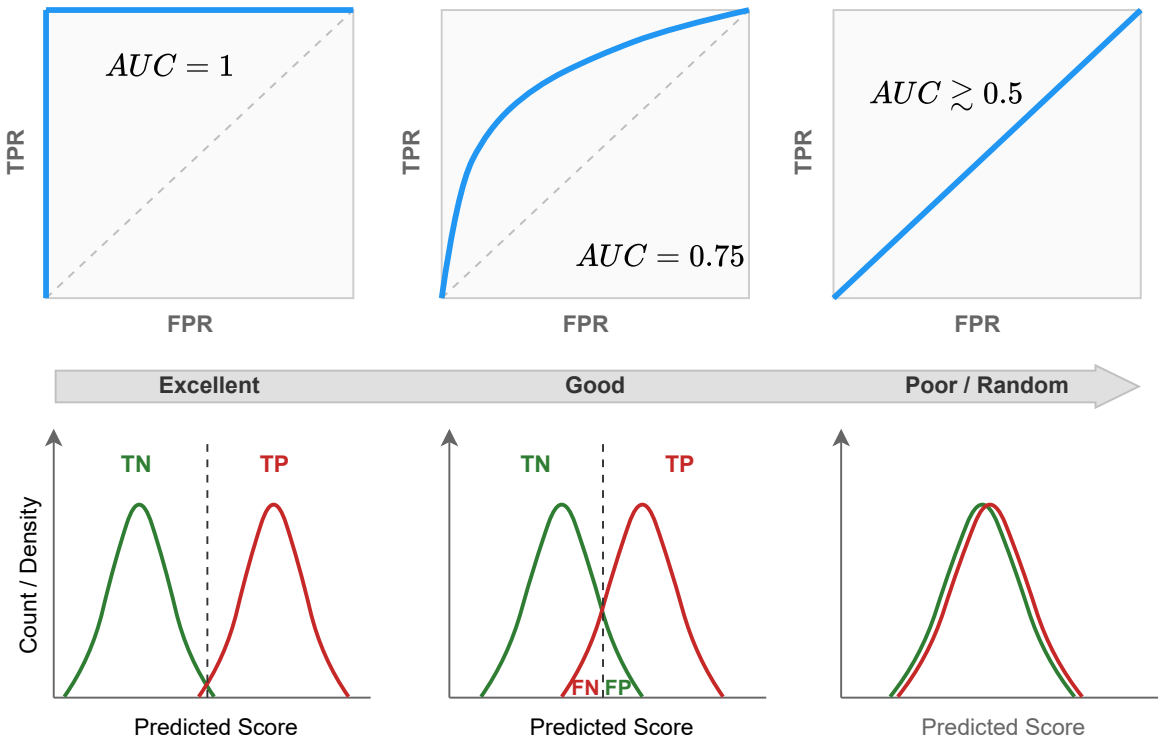
12.6. Multi-class performance

Goal: Extend the binary metrics to the multi-class case, where each sample belongs to one of $C \geq 2$ classes.

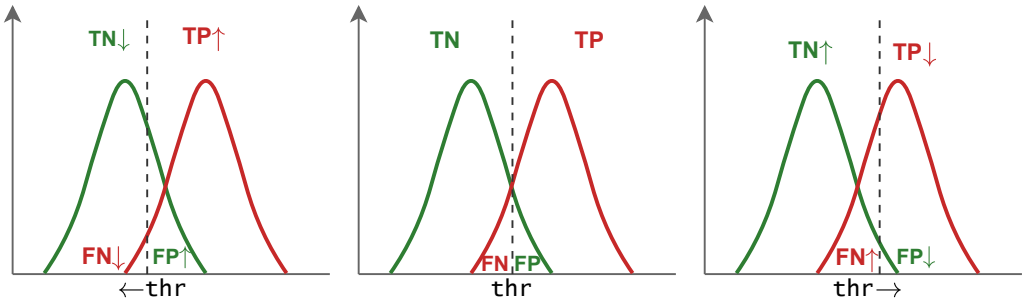
Single-label multi-class classification: Each sample carries exactly one true class label out of $C \geq 2$ classes, and the classifier outputs exactly one predicted class. Both target and prediction are elements of $\{1, \ldots, C\}$.

Multi-label multi-class classification: Each sample may carry any subset of the C class labels (possibly empty), and the classifier outputs an independent binary decision per class. Both target and prediction are subsets of $\{1, \ldots, C\}$, equivalently binary vectors in $\{0, 1\} \in \mathbb{R}^C$.

The remainder of this section assumes single-label classification; the multi-label setting is contrasted briefly in Sec. 12.6.3.



(a) RoC curves (top) and corresponding class distributions (bottom) for three classifiers. Well-separated distributions yield AUC ≈ 1; overlapping distributions produce FN/FP errors and AUC $\to 0.5$.



(b) Effect of threshold on classification: lowering thr (left) reduces FN but increases FP; raising thr (right) reduces FP but increases FN. The arrows indicate the direction of change relative to the default threshold (center).

Figure 12.2.: RoC curves, class distributions, and the effect of the decision threshold.

For single-label multi-class classification, the confusion matrix becomes a $C \times C$ table whose entries

$$N_{cc'} = \sum_{i=1}^{M} \mathbb{1}[Y_i = c]\ \mathbb{1}\left[\hat{Y}_i = c'\right] \tag{12.24}$$

count the samples with true class c predicted as c'. Accuracy generalizes naturally as the diagonal sum normalized by the total number of samples,

$$\text{Accuracy} = \frac{\sum_c N_{cc}}{\sum_c N_{cc} + \sum_c \sum_{c' \neq c} N_{cc'}} = \frac{1}{M} \sum_c N_{cc}. \tag{12.25}$$

Precision, recall, and F_1, however, are defined with respect to a single positive class, so they must first be computed per class and then aggregated.

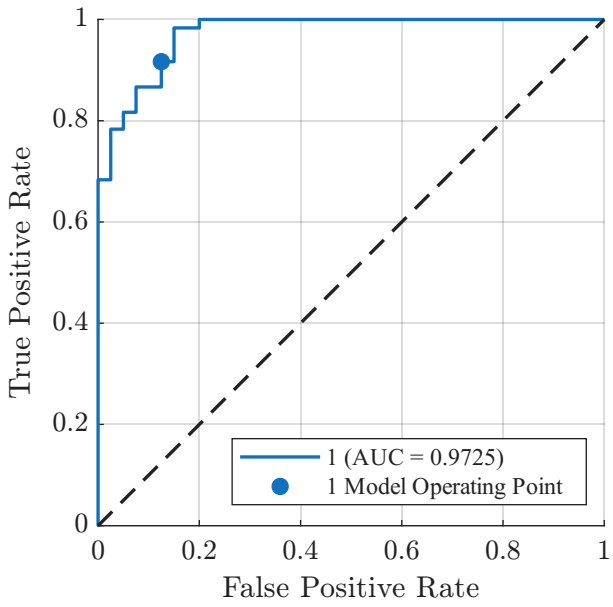


Figure 12.3.: RoC of logistic regression example in Fig. 8.1. The model operation point is $\mathsf{thr} = 0.5$.

Example 12.5: Consider a single-label 3-class problem with $M = 100$ samples and the following confusion matrix (rows = actual class, columns = predicted class):

		Predicted			
		A	B	C	Total
Actual	A	72	6	2	80
	B	8	6	1	15
	C	2	1	2	5
	Total	82	13	5	100

The classes are heavily imbalanced (A: 80, B: 15, C: 5). Taking class A as the positive class, the diagonal cell (green) is TP_A, the off-diagonal cells in column A (red) are FP_A, and the off-diagonal cells in row A (yellow) are FN_A. Reading the matrix row-by-row and column-by-column gives the per-class counts

$TP_A = 72$	$FP_A = 10$	$FN_A = 8$
$TP_B = 6$	$FP_B = 7$	$FN_B = 9$
$TP_C = 2$	$FP_C = 3$	$FN_C = 3$

This running example is used throughout the rest of the section.

12.6.1. Per-class performance

The metrics in Sec. 12 are defined with respect to the positive class ($Y = 1$). To assess all classes, the metrics can be computed separately for each class by treating it, in turn, as the positive class.

For class c, define:

- TP_c – samples of class c correctly predicted as class c
- FP_c – samples of other classes incorrectly predicted as class c
- FN_c – samples of class c incorrectly predicted as another class

Equivalently, in terms of the confusion-matrix entries $N_{cc'}$ of Eq. 12.24,

$$TP_c = N_{cc}, \qquad FP_c = \sum_{c' \neq c} N_{c'c}, \qquad FN_c = \sum_{c' \neq c} N_{cc'}. \tag{12.26}$$

The per-class metrics are then

$$\text{Precision}_c = \frac{TP_c}{TP_c + FP_c}, \tag{12.27}$$

$$\text{Recall}_c = \frac{TP_c}{TP_c + FN_c}, \tag{12.28}$$

$$F_{1,c} = \frac{2 \cdot \text{Precision}_c \cdot \text{Recall}_c}{\text{Precision}_c + \text{Recall}_c} \tag{12.29}$$

Note the symmetry: for binary classification, $FP_0 = FN_1$ and $FP_1 = FN_0$, so precision for one class equals recall for the other only when $FP_0 = FP_1$.

Example 12.5: Back to Example 12.5. Plugging the per-class counts into the per-class formulas:

Metric	Class A (80)	Class B (15)	Class C (5)
Precision	$72/82 \approx 87.8\%$	$6/13 \approx 46.2\%$	$2/5 = 40.0\%$
Recall	$72/80 = 90.0\%$	$6/15 = 40.0\%$	$2/5 = 40.0\%$
F_1	$\approx$88.9%	$\approx$42.9%	40.0%

Table 12.3.: Per-class metrics for the running 3-class example.

Class A (the majority) is classified well, while the rare classes B and C are classified poorly on every per-class metric.

Example 12.1: Back to the Covid antibody test example (Example 12.1, with $TP = 141$, $FP = 0$, $FN = 67$, $TN = 31$).

The asymmetry is significant: $FP = 0$ yields perfect precision for class 1 but also perfect recall for class 0, while the high $FN = 67$ degrades recall for class 1 and precision for class 0. This illustrates the symmetry $FP_1 = FN_0$ noted above.

To collapse the C per-class metrics into a single scalar, two averaging strategies are commonly used.

12.6.2. Macro-averaging

The arithmetic mean of the per-class metric across all C classes:

$$\text{Precision}_{\text{macro}} = \frac{1}{C} \sum_{c=1}^{C} \text{Precision}_c, \tag{12.30}$$

Metric	Class 1 (positive)	Class 0 (negative)
Precision	$\frac{141}{141+0} = 100\%$	$\frac{31}{31+67} \approx 31.6\%$
Recall	$\frac{141}{141+67} \approx 67.8\%$	$\frac{31}{31+0} = 100\%$
F_1	$\frac{2 \cdot 1.0 \cdot 0.678}{1.0+0.678} \approx 80.8\%$	$\frac{2 \cdot 0.316 \cdot 1.0}{0.316+1.0} \approx 48.1\%$

Table 12.4.: Per-class metrics for the Covid antibody test.

$$\text{Recall}_{\text{macro}} = \frac{1}{C}\sum_{c=1}^{C} \text{Recall}_c, \tag{12.31}$$

$$F_{1,\text{macro}} = \frac{1}{C}\sum_{c=1}^{C} F_{1,c}. \tag{12.32}$$

Each class contributes equally, regardless of its support (the number of samples in the class). A rare class with poor performance pulls the macro score down as much as a frequent class. Macro-averaging is therefore the natural choice when minority-class performance matters, for example in imbalanced datasets or when fairness across categories is required.

The balanced accuracy of Sec. 12.3.6 generalizes to the multi-class case as exactly the macro-averaged recall, $\text{Recall}_{\text{macro}}$; for the running 3-class example this is 56.7% (computed below).

Example 12.5: Back to Example 12.5. Averaging the per-class metrics:

$$\begin{aligned}
\text{Precision}_{\text{macro}} &= \tfrac{1}{3}(0.878 + 0.462 + 0.400) \approx 58.0\%, \\
\text{Recall}_{\text{macro}} &= \tfrac{1}{3}(0.900 + 0.400 + 0.400) \approx 56.7\%, \\
F_{1,\text{macro}} &= \tfrac{1}{3}(0.889 + 0.429 + 0.400) \approx 57.3\%.
\end{aligned}$$

The macro F_1 (57.3%) is far below the overall accuracy of 80%, because the rare classes B and C, on which the classifier performs poorly, contribute equally to the macro mean despite holding only 20 of the 100 samples.

12.6.3. Micro-averaging

Pool the confusion-matrix counts across all classes first, then compute the metric once on the pooled counts:

$$\text{Precision}_{\text{micro}} = \frac{\sum_c TP_c}{\sum_c (TP_c + FP_c)}, \tag{12.33}$$

$$\text{Recall}_{\text{micro}} = \frac{\sum_c TP_c}{\sum_c (TP_c + FN_c)}. \tag{12.34}$$

Each instance contributes equally, regardless of the class label, so frequent classes dominate the score.

Example 12.5: Back to Example 12.5. Pooling the counts across the three classes:

$$\text{Precision}_{\text{micro}} = \frac{TP_A + TP_B + TP_C}{(TP_A + FP_A) + (TP_B + FP_B) + (TP_C + FP_C)} = \frac{72 + 6 + 2}{82 + 13 + 5} = \frac{80}{100} = 80.0\%.$$

The same value coincides with the overall accuracy of the classifier, $(72 + 6 + 2)/100 = 80\%$. The next subsubsection shows that this is not a coincidence in the single-label setting.

Equivalence to accuracy in single-label classification

In the single-label multi-class setting (every sample has exactly one true class, and the model predicts exactly one class), micro-averaged precision, recall, and F_1 all collapse to accuracy:

$$\text{Precision}_{\text{micro}} = \text{Recall}_{\text{micro}} = F_{1,\text{micro}} = \text{Accuracy}. \tag{12.35}$$

The argument has three steps.

1. *Misclassifications are double-counted across classes*: Take any wrong prediction: a sample whose true class is a but predicted as b ($a \neq b$). From class b's perspective it is a false positive (predicted b but not actually b), contributing $+1$ to FP_b. From class a's perspective it is a false negative (actually a but missed), contributing $+1$ to FN_a. The same error therefore raises exactly one FP_c and exactly one FN_c. Summing over all classes:

$$\sum_c FP_c = \sum_c FN_c = (\text{total number of misclassified samples}). \tag{12.36}$$

2. $\sum_c TP_c$ *is the total number of correct predictions*: Each correct prediction adds 1 to TP_c for its single true class, so

$$\sum_c TP_c = M \cdot \text{Accuracy}, \tag{12.37}$$

   where M is the dataset size.

3. *Plug into the micro formulas*:

$$\text{Precision}_{\text{micro}} = \frac{\sum_c TP_c}{\sum_c TP_c + \sum_c FP_c} = \frac{\text{correct}}{\text{correct} + \text{wrong}} = \frac{\text{correct}}{M} = \text{Accuracy}. \tag{12.38}$$

   The denominator for recall is identical because $\sum_c FN_c = \sum_c FP_c$, so $\text{Recall}_{\text{micro}} = \text{Accuracy}$. Since precision equals recall, their harmonic mean $F_{1,\text{micro}}$ takes the same value.

> The identity $\sum_c FP_c = \sum_c FN_c$ relies on each error contributing exactly one FP and one FN. In *multi-label* classification (a sample can belong to several classes simultaneously), a single sample can be wrong on multiple labels independently, so the bookkeeping no longer pairs up and micro-precision, micro-recall, and accuracy become genuinely different quantities.

Practical takeaway. In a single-label multi-class setting, reporting micro-F_1 alongside accuracy is redundant: they are the same number. To expose minority-class behavior, use macro-averaging instead.

With balanced classes, macro and micro averages converge. With imbalance, they can differ substantially: micro tracks accuracy, while macro exposes poor minority-class performance.

Example 12.5: Back to Example 12.5 once more, the proof can be checked numerically. The total number of misclassified samples is $M - (72 + 6 + 2) = 20$, and summing the per-class counts:

$$\sum_c FP_c = 10 + 7 + 3 = 20, \qquad \sum_c FN_c = 8 + 9 + 3 = 20,$$

matching the count of misclassifications, as predicted by step 1 of the argument. Therefore $\text{Precision}_{\text{micro}} = \text{Recall}_{\text{micro}} = F_{1,\text{micro}} = 80\% = \text{Accuracy}$, while the macro F_1 of 57.3% exposes the poor performance on the rare classes B and C.

Further reading

Multi-class classification performance

- Accuracy, precision, and recall in multi-class classification (accessed 27-Apr-2026).

13. Sanity Checks for Cross-Validation of Classifiers

Goal: Verify that cross-validation results reflect genuine predictive signal rather than artifacts of the evaluation procedure.

Contents

13.1. Cross-Validation Strategy Pitfalls **158**
13.1.1. Grouped, Stratified, and Grouped Stratified CV 158
13.1.2. Group Leakage Diagnostic 159
13.1.3. Leave-One-Out Cross-Validation (LOOCV) 160
13.2. Data Integrity **161**
13.2.1. Data Leakage 161
13.2.2. Domain Consistency 162
13.3. Statistical Validation **163**
13.3.1. Permutation Test 163
13.3.2. Omnibus Permutation Test 164
13.3.3. Per-Class Permutation Test 165
13.3.4. Random Feature Baseline 166
13.3.5. Cross-class sample permutation 166

Cross-validation (Sec. 4.5) provides an estimate of model performance on unseen data. However, this estimate can be misleading—especially with small datasets ($M \lesssim 10^2$), high-dimensional feature spaces ($N \gg M$), or data collected from heterogeneous sources. A classifier may report high accuracy not because it has learned the true relationship between $\mathbf{X}$ and $\mathbf{y}$, but because the evaluation procedure itself is flawed.

Common sources of misleadingly optimistic CV results include:

- An inappropriate CV splitting strategy that leaks information between folds.
- Data-dependent pre-processing steps (e.g., feature selection, normalization) fitted on the entire dataset before splitting.
- High-dimensional feature spaces where spurious correlations arise by chance.
- Domain-specific artifacts (e.g., scanner calibration, patient baselines) that the model memorizes instead of learning generalizable patterns.

This chapter presents a series of sanity checks designed to detect these problems. The checks are organized from general methodology (CV strategy and data integrity) to specific diagnostic tests (statistical validation and image-specific techniques). Each check is independent and can be applied selectively based on the problem domain.

13.1. Cross-Validation Strategy Pitfalls

Goal: Ensure that the CV splitting strategy does not artificially inflate performance estimates.

The choice of how to partition data into folds is itself a modeling decision that can introduce bias. This section reviews two common pitfalls: ignoring group structure in the data, and using leave-one-out CV in high-dimensional settings.

13.1.1. Grouped, Stratified, and Grouped Stratified CV

Standard random cross-validation implicitly assumes that samples are independent and identically distributed (i.i.d.), and that the classes are relatively balanced. When these assumptions fail, specialized splitting strategies are required:

- *Grouped CV* (introduced in Sec. 4.5) ensures that all samples from a given group remain in the same fold, preventing leakage when multiple samples are drawn from the same underlying entity (e.g., a patient or sensor).
- *Stratified CV* preserves the original class proportions in every fold, addressing artifacts caused by class imbalance (Sec. 12.4).
- *Grouped stratified CV* attempts to combine both constraints simultaneously: no group spans multiple folds, and the class distribution is maintained as closely as possible across folds.

Stratification for imbalanced data When a dataset is highly imbalanced (Sec. 12.4), standard random splitting can result test folds where the minority class is underrepresented or entirely absent. Stratified CV enforces that the target class distribution is maintained in each fold, ensuring that performance metrics such as recall and F_1-score are evaluated on a representative sample. This is particularly crucial for small datasets where random variation in fold composition can heavily change the overall performance estimate.

Example 13.1 (Stratified CV for Rare Classes): Consider a dataset with $M = 100$ samples, containing 90 majority and 10 minority class samples, split into 5 folds of 20 samples each:

	Standard 5-fold		Stratified 5-fold	
Fold	Majority	Minority	Majority	Minority
1	17	3	18	2
2	20	0	18	2
3	19	1	18	2
4	16	4	18	2
5	18	2	18	2

With standard splitting, fold 2 contains zero minority samples, making it impossible to compute precision or recall on that test fold. Stratified CV prevents this by strictly preserving the 9:1 class ratio in every fold, providing stable performance estimates.

Combining grouping and stratification When data is both grouped and imbalanced (e.g., medical data with multiple scans per patient, where some patients have a rare condition), grouped stratified CV is necessary. However, unlike pure stratification, combining both grouping and stratification is sometimes an inherently challenging optimization problem. Because groups can vary significantly in size and contain mixed class distributions, achieving perfect stratification is generally mathematically impossible without violating the grouping constraint.

Grouped stratified CV algorithms therefore seek a compromise: they strictly enforce the grouping constraint (ensuring no data leakage) while heuristically assigning groups to folds to approximate the global class distribution as closely as possible.

Example 13.2 (The Challenge of Grouped Stratified CV): A medical dataset contains scans from three patients. The total is 25 normal and 15 abnormal scans (ratio 5:3).

	Normal	Abnormal	Total
Patient A	10	10	20
Patient B	15	0	15
Patient C	0	5	5
Dataset	25	15	40

The grouped stratified algorithm assigns whole patients to folds, approximating the global class ratio as closely as possible:

	Patients	Normal	Abnormal	Ratio
Ideal fold	—	12.5	7.5	5 : 3
Fold 1	A	10	10	1 : 1
Fold 2	B, C	15	5	3 : 1

Both folds deviate from the ideal 5:3 ratio, but this is the best achievable split that strictly prevents patient leakage between folds.

13.1.2. Group Leakage Diagnostic

Goal: Detect whether a model exploits domain-specific signatures rather than generalizable patterns.

When data is collected from distinct logical groups, such as multiple patients, different sensors, or separate geographic locations, standard random cross-validation can inadvertently place samples from the same group into both the training and test folds. Consequently, the model may learn to recognize group-specific signatures (e.g., a specific patient's baseline signal amplitude or a particular sensor's calibration offset) instead of uncovering the true, generalizable underlying phenomena.

A simple and effective diagnostic for this issue is to evaluate the same ML pipeline using two different splitting strategies:

1. *Standard (ungrouped) random CV.*

2. *Grouped CV* (Sec. 4.5), which strictly ensures all samples from a given group are isolated within a single fold.

Confounder: A variable that correlates with both the input features and the target label but is not part of the true underlying relationship. A model that relies on confounders achieves high apparent accuracy without learning the genuine pattern, and fails when the confounding correlation no longer holds (e.g., when deployed on data from a new domain).

Example 13.3 (Confounder in Medical Imaging): A classifier is trained to detect pneumonia from chest X-rays collected at two hospitals. Hospital A uses portable X-ray machines (mostly for bedridden, sicker patients) and Hospital B uses fixed machines (mostly for ambulatory patients). The classifier learns to distinguish portable from fixed X-ray artifacts (a confounder that correlates with disease severity) rather than the actual lung pathology. It achieves high accuracy on data from both hospitals but fails on X-rays from a third hospital with a different equipment mix.

If the ungrouped cross-validation yields substantially higher accuracy than the grouped cross-validation, it is a strong indication that the model is exploiting group-specific confounders. In such scenarios, relying solely on grouped CV for evaluation may be insufficient if the ultimate goal is to build a highly accurate, generalizable model. The underlying data representation itself must be improved to prevent the model from learning the confounders in the first place. Common mitigation strategies include:

- **Subject-specific normalization:** Removing baseline offsets by standardizing features on a per-group basis (e.g., subtracting a patient's resting baseline from their active measurements) rather than computing statistics globally across the entire dataset.
- **Feature engineering:** Designing or extracting features that are fundamentally invariant to the known group-specific confounders (e.g., focusing on relative frequency-domain changes instead of absolute raw time-domain amplitudes).
- **Domain adaptation techniques:** Employing models, architectures, or loss functions (such as adversarial training) specifically designed to learn invariant representations across different domains or groups.

13.1.3. Leave-One-Out Cross-Validation (LOOCV)

Goal: The choice of the cross-validation strategy itself can drastically inflate performance estimates if misapplied to the wrong dataset regime. A prime example is using LOOCV on high-dimensional data.

LOOCV uses k-fold cross-validation with $k = M$, leaving exactly one sample out per fold. While sometimes necessary for extremely small datasets (Sec. 4.5), LOOCV suffers from several drawbacks that make it a potential source of misleadingly optimistic results:

1. *High computational cost.* LOOCV requires training the model M times. For large datasets this is impractical; it is only feasible when M is small.

2. *High variance in performance estimates.* Because only one sample is left out, the M training sets are nearly identical to each other and have *low bias*. However, the resulting models are highly correlated, and the average of correlated estimates can have *higher variance* than 5-fold or 10-fold cross-validation.

3. *Sensitivity to outliers.* Each data point serves as the sole validation sample in exactly one fold. A single outlier or noisy point can strongly influence that fold's error, producing unreliable overall estimates compared to k-fold where outliers are diluted within larger folds.

4. *Potential for overfitting.* Because each training set contains $M-1$ points, the resulting models are very similar to the full-data model. In high-dimensional or noisy settings ($N \gg M$), LOOCV can favor overly complex models that fit the training data too closely.

Example 13.4 (LOOCV Overfitting on Random Noise): LOOCV with $N \gg M$ can yield near-perfect accuracy even on pure noise.

Consider $M = 80$ samples with $N = 2048$ features drawn entirely from i.i.d. $\mathcal{N}(0, 1)$ (pure noise) and binary labels with class sizes 51 vs. 29 (majority baseline ≈ 0.64). A pipeline of standardization $\rightarrow$ PCA(5) $\rightarrow$ RBF-SVM ($C = 0.05$) with LOOCV achieves 100% accuracy across 10 random seeds—despite the features containing no real signal.

This illustrates how LOOCV combined with high dimensionality ($N \gg M$) can produce misleadingly optimistic performance estimates. The diagnostic methods described in the remainder of this chapter—particularly the random feature baseline (Sec. 13.3.4) and the permutation test (Sec. 13.3.1)—can detect such conditions.

13.2. Data Integrity

Goal: Identify situations where the data itself—rather than the model—is responsible for inflated performance.

Even with a correctly configured CV strategy, problems in the data can produce misleading results. This section covers two common issues: data leakage (information from outside the training set influencing model training) and domain inconsistency (distribution shifts between training and deployment).

13.2.1. Data Leakage

Data leakage: (also termed *train-test contamination*) Information from outside the training set (for example, test labels, future observations, or global statistics) influences model training, producing overly optimistic performance estimates.

Common sources of leakage:

- *Pre-processing leakage*: fitting data-dependent transformations (e.g., standardization, PCA, feature selection) on the entire dataset *before* splitting into CV folds. The test fold then contains information that was used during training.
- *Temporal leakage*: in time-series problems, using future observations to predict the past. Standard random splits violate the causal ordering; temporal (forward-chaining) splits should be used instead.
- *Target leakage*: including features that are derived from or strongly correlated with the target variable (e.g., a feature computed from the label, or a proxy that is unavailable at prediction time).

Pre-processing leakage example. Consider a pipeline that applies standardization (Sec. 4.7) before classification with k-fold CV (Sec. 4.5). If the mean $\bar{\mathbf{x}}$ and standard deviation $s_{\mathbf{x}}$ are computed on the *entire* dataset (all M samples) and then the standardized features are split into CV folds, the test fold's features encode information about the global data distribution, including the test samples themselves. The correct approach is to compute $\bar{\mathbf{x}}$ and $s_{\mathbf{x}}$ *inside* each CV fold on the training partition only, and then transform the test partition using the training-derived statistics. The same principle applies to any other data-dependent transformation.

Prevention & Best Practices

- All data-dependent pre-processing steps (e.g., supervised feature selection, PCA, normalization as in Sec. 4.7) must be fitted **strictly inside** each CV fold on the training partition only.
- Audit feature provenance to ensure no target-derived or future-derived information enters the feature matrix.
- Ensure an appropriate number of folds: too few folds, especially when combined with class imbalance, can produce biased estimates; too many folds (approaching LOOCV) can produce high-variance estimates and mask overfitting in high-dimensional settings (Sec. 13.1.3).

13.2.2. Domain Consistency

Goal: Test whether two datasets share the same underlying distribution before combining them for training or expecting cross-domain generalization.

Even when grouped CV (Sec. 13.1.2) correctly prevents within-group leakage, the model may still fail if the deployment domain differs systematically from the training domains.

Domain Inconsistency (Dataset Shift): A scenario where the distribution of the data (features or labels) in the deployment environment differs from the distribution in the training data. Common sources include differences in acquisition hardware (e.g., scanner manufacturer), environmental conditions, population demographics, or labeling conventions.

Given two datasets $\mathbf{X}_A \in \mathbb{R}^{M_A \times N}$ and $\mathbf{X}_B \in \mathbb{R}^{M_B \times N}$, a natural question is: can a classifier tell them apart? If yes, the domains are inconsistent.

Classifier two-sample test

The idea is to recast the distributional question as a binary classification problem:

1. Assign artificial labels $y = 0$ to all samples in $\mathbf{X}_A$ and $y = 1$ to all samples in $\mathbf{X}_B$.
2. Concatenate into $\mathbf{X} = [\mathbf{X}_A; \mathbf{X}_B] \in \mathbb{R}^{M \times N}$, $M = M_A + M_B$, with label vector $\mathbf{y}$.
3. Train a binary classifier on $(\mathbf{X}, \mathbf{y})$ using stratified k-fold CV and record the accuracy J_{real}.

The chance-level accuracy is the majority-class rate $J_{\text{chance}} = \max(M_A, M_B)/M$ (Eq. 13.2), and if $J_{\text{real}} \approx J_{\text{chance}}$ than $\mathbf{X}_A$ and $\mathbf{X}_B$ are drawn from the same distribution, Otherwise, $J_{\text{real}} \gg J_{\text{chance}}$ and the distributions differ.

Statistical significance via permutation test To determine whether J_{real} is significantly above J_{chance}, apply the permutation test (Sec. 13.3.1) with the following application-specific interpretation:

- *Labels shuffled*: the domain labels $\mathbf{y}$ (the artificial 0/1 assignments), breaking any association between features and domain membership.
- *Null distribution*: the classifier accuracy expected when samples from $\mathbf{X}_A$ and $\mathbf{X}_B$ are interchangeable, i.e., when no domain difference exists.
- *Decision*: if $p < \alpha$, reject H_0 — the two datasets are statistically distinguishable and domain inconsistency is present.

Example 13.5 (Classifier Two-Sample Test): Dataset A contains $M_A = 60$ EEG recordings from Hospital 1 and dataset B contains $M_B = 40$ recordings from Hospital 2, both with $N = 128$ spectral features:

	Hospital 1 ($y = 0$)	Hospital 2 ($y = 1$)
Samples	60	40
Chance accuracy J_{chance}	$60/100 = 0.60$	
Domain classifier accuracy J_{real}	$0.91 \gg J_{\text{chance}}$	
Permutation test ($\alpha = 0.05$)	$p < 0.001 \Rightarrow$ reject H_0	

The classifier easily separates the two hospitals ($J_{\text{real}} = 0.91 \gg 0.60$, $p < \alpha$), indicating that the feature distributions differ substantially—likely due to hardware or acquisition protocol differences. A task classifier trained on Hospital 1 data alone should not be expected to generalize to Hospital 2 without domain adaptation.

Rejecting H_0 does not necessarily mean the task classifier will fail—only that the feature distributions differ. However, high domain separability is a strong warning that cross-domain generalization may be poor.

13.3. Statistical Validation

Goal: Apply formal diagnostic tests to determine whether the observed CV performance reflects genuine signal or statistical artifacts.

Both tests in this section share a common logic: destroy the signal that a valid pipeline should rely on, and verify that performance drops to chance. The permutation test destroys the labels; the random feature baseline destroys the features.

13.3.1. Permutation Test

Goal: Test whether the model has learned a genuine relationship or the observed accuracy is indistinguishable from chance.

Permutation test: A statistical test that evaluates whether the observed CV performance is significantly better than what would be obtained by chance:

1. Train and evaluate the model using CV, obtaining the real score J_{real}.
2. Randomly shuffle the labels $\mathbf{y}$, breaking any true relationship between $\mathbf{X}$ and $\mathbf{y}$.
3. Re-run the same CV on the shuffled labels, obtaining a permuted score $J_{\text{perm}}^{(i)}$.
4. Repeat steps 2–3 for T times (typically $T = 1000$) to build a *null distribution*—the distribution of scores when there is no true relationship between $\mathbf{X}$ and $\mathbf{y}$.

The test is framed as a hypothesis test:

- H_0: The features $\mathbf{X}$ and labels $\mathbf{y}$ are independent — the model has no predictive power.
- H_1: $\mathbf{X}$ carries genuine information about $\mathbf{y}$.

Under H_0, shuffling the labels does not change the joint distribution, so the permuted scores $J_{\text{perm}}^{(i)}$ represent the distribution of performance expected by chance.

The p-value is the fraction of permuted scores that are at least as good as the real score,

$$p = \frac{1}{T+1}\left(\sum_{i=1}^{T} \mathbf{1}\left[J_{\text{perm}}^{(i)} \geq J_{\text{real}}\right] + 1\right) \tag{13.1}$$

where $\mathbf{1}[\cdot]$ is the indicator function. The $+1$ in numerator and denominator accounts for the real score itself.

Decision rule The decision is made by comparing the p-value to a predetermined *significance level* α, which is the maximum acceptable probability of incorrectly rejecting H_0. Common choices are $\alpha = 0.05$ (5%) and $\alpha = 0.01$ (1%).

- If $p < \alpha$, reject H_0: the model has learned a genuine relationship between $\mathbf{X}$ and $\mathbf{y}$.
- If $p \geq \alpha$, fail to reject H_0: the observed accuracy is not statistically distinguishable from chance.

Choosing T The number of permutations T determines the resolution of the p-value: the smallest achievable p-value is $\frac{1}{T+1}$. For standard significance testing at $\alpha = 0.05$, $T = 100$ suffices. For more precise p-values (e.g., $p < 0.001$), use $T \geq 1000$. Each permutation requires a full CV run, so the total cost is $(T + 1)$ times a single CV evaluation.

13.3.2. Omnibus Permutation Test

Goal: Obtain a single significance verdict on whether a multi-class classifier carries *any* genuine predictive signal, without the multiple-comparison inflation of running one test per class.

With K classes, a tempting approach is to run the permutation test (Sec. 13.3.1) separately for each class in a one-vs-rest fashion. This gives K chances to reject H_0 by accident: even when no class is predictable, the probability that *at least one* of the K tests is spuriously significant (the *family-wise error rate*) far exceeds the nominal α.

The omnibus permutation test avoids this by reducing the entire multi-class problem to a *single* test. The procedure is identical to Sec. 13.3.1, but each CV run is scored with one global statistic that

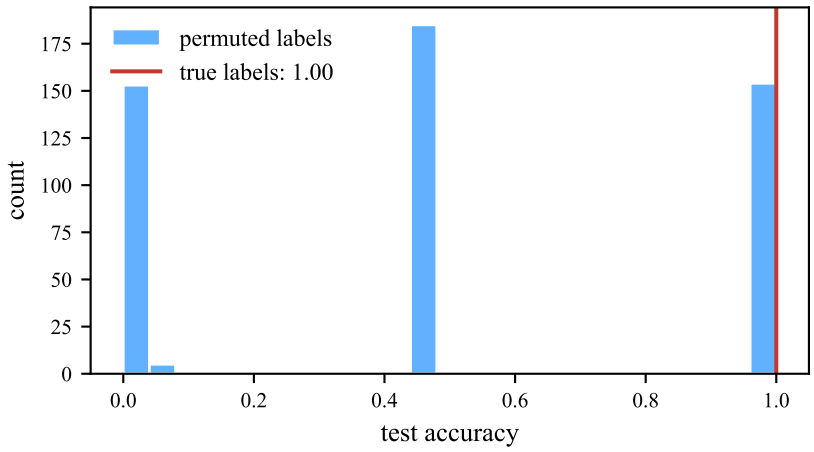


Figure 13.1.: Distribution of test accuracy under T label permutations. The vertical line marks the true-label accuracy; the area to the right is the permutation p-value.

aggregates all classes, such as overall accuracy, balanced (macro) accuracy, or macro-F_1 (preferred under class imbalance, Sec. 12.4). The full label vector $\mathbf{y}$ is shuffled jointly, producing one null distribution and one p-value via the same $\frac{1}{T+1}$ estimator. The hypotheses become:

- H_0: $\mathbf{X}$ and $\mathbf{y}$ are independent — no class is predictable.
- H_1: $\mathbf{X}$ carries information about at least one class.

Example 13.6 (Omnibus Test on a 4-Class Problem): A pipeline is evaluated on $K = 4$ balanced classes (chance accuracy ≈ 0.25). The observed macro-F_1 is 0.41, and across $T = 1000$ label permutations no permuted run reaches this value, giving $p < 0.001$. The omnibus test rejects H_0: the classifier carries genuine multi-class signal.

Rejecting H_0 establishes that the pipeline learned *something*, but not *which* classes. The per-class follow-up of Sec. 13.3.3 localizes the signal.

The omnibus test inherits every assumption of the base permutation test: the same CV splitting strategy and all pre-processing fitted strictly inside each fold (Sec. 13.2.1). A leaky pipeline produces a misleadingly significant omnibus p-value just as readily as a per-class one.

13.3.3. Per-Class Permutation Test

Goal: After the omnibus test rejects, identify *which* classes carry signal while controlling the error rate across the K simultaneous tests.

For each class k, collapse the labels to one-vs-rest ($y = 1$ for class k, $y = 0$ otherwise) and run the permutation test of Sec. 13.3.1 with a per-class statistic (e.g. that class's F_1 or one-vs-rest AUC), yielding raw p-values $p_1, \ldots, p_K$. Because K hypotheses are tested at once, these raw p-values must be adjusted:

- *Bonferroni*: reject class k if $p_k < \alpha/K$. This controls the family-wise error rate (the probability of *any* false positive) but is conservative.
- *Benjamini-Hochberg (FDR)*: sort the p-values $p_{(1)} \leq \cdots \leq p_{(K)}$, find the largest i with $p_{(i)} \leq \frac{i}{K}\alpha$, and reject it together with all smaller ranks. This controls the expected *fraction* of false

positives among the rejected classes, and is more powerful when several classes are genuinely predictable.

Standard practice is to gate the per-class scan on a significant omnibus result (Sec. 13.3.2): the omnibus answers "is there any signal?" and the per-class scan answers "which classes?" only once the global test has rejected.

Example 13.7 (Localizing the Signal): Continuing the $K = 4$ example (Sec. 13.3.2), the raw per-class p-values are (0.002, 0.02, 0.31, 0.6). Under Bonferroni, $\alpha/K = 0.05/4 = 0.0125$, so only class 1 survives ($0.02 > 0.0125$). Benjamini-Hochberg additionally retains class 2, since $p_{(2)} = 0.02 \leq \frac{2}{4} \cdot 0.05 = 0.025$, illustrating how the choice of correction trades power against strictness.

Never report the single most significant class without a multiple-comparison correction. Selecting the best of K raw p-values is precisely the inflation the omnibus test (Sec. 13.3.2) was designed to guard against.

13.3.4. Random Feature Baseline

Goal: Verify that the pipeline does not achieve high accuracy on meaningless input.

While the permutation test (Sec. 13.3.1) keeps the real features and destroys the labels, the random feature baseline takes the complementary approach: keep the real labels but destroy the features.

Replace the real feature matrix $\mathbf{X}$ with random Gaussian noise $\mathbf{X}_{\text{rand}} \sim \mathcal{N}(0, 1)$ of the same dimensions, and run the identical pipeline (same pre-processing, same model, same CV). The expected accuracy should be close to the majority-class rate,

$$J_{\text{chance}} = \frac{\max(M_0, M_1)}{M} \tag{13.2}$$

where M_0 and M_1 are the class counts.

If the pipeline achieves accuracy substantially above J_{chance} on random features, the evaluation procedure itself is flawed (e.g., data leakage as in Sec. 13.2.1, or LOOCV in high dimensions as in Example 13.4).

On imbalanced datasets with stratified CV, the random-feature baseline may slightly exceed J_{chance} due to the interaction between stratification and the classifier's bias toward the majority class. A deviation of a few percentage points above J_{chance} is not necessarily alarming; a substantial deviation (e.g., $> 10\%$ above chance) indicates a pipeline problem.

13.3.5. Cross-class sample permutation

Goal: Detect per-sample leakage cues (filename, timestamp, recording-session id, residual artifacts) that survive a swap of samples between classes.

Where the permutation test (Sec. 13.3.1) destroys labels globally and the random-feature baseline (Sec. 13.3.4) destroys features, the cross-class swap destroys the class-conditional signal incrementally. Swap a controlled fraction $\alpha \in [0, 0.5]$ of samples between class buckets and refit the full pipeline. Accuracy should degrade approximately monotonically with α and approach the label-shuffle null at $\alpha = 0.5$.

- *Monotone degradation*: the model is learning a class-conditional signal that is being progressively corrupted.
- *Flat curve*: the model is not exploiting the class-conditional signal at all; it relies on a per-sample cue that is preserved under the swap, e.g. filename, timestamp, recording-session id, duration, or any other metadata that survived into the feature vector (Sec. 13.2.2).
- *Non-monotone curve*: typically a sign of two competing cues, one of which is leakage that the swap inadvertently reinforces.

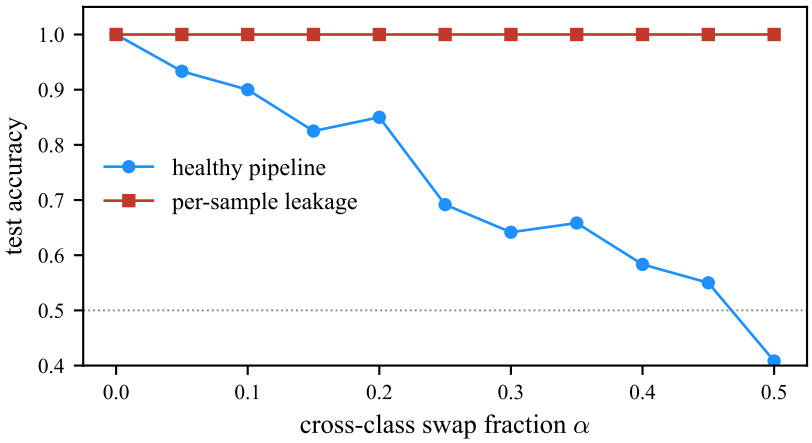


Figure 13.2.: Accuracy vs. cross-class swap fraction α. A monotone curve (blue) is healthy; a flat curve (red) indicates per-sample leakage.

Flat cross-class curve = silent leakage

A flat accuracy curve under cross-class swap is the strongest leakage signature among the permutation diagnostics. The model has identified each sample individually (often via metadata that survived into the feature vector) and is reporting test accuracy by recognition rather than by classification. Trace back the per-sample cues introduced by the acquisition or storage pipeline.

14. Classifier Comparison

Goal: Compare two classifiers using both hard decisions and predicted probabilities.

Contents

14.1. Comparison Between Two Non-Probabilistic Classifiers . . . 169
14.1.1. Summary . . . 169
14.1.2. Contingency Table . . . 169
14.1.3. Disagreement Measure . . . 170
14.1.4. McNemar's Test . . . 171
14.1.5. Cohen's Kappa Coefficient . . . 171
14.1.6. Q-Statistic (Yule's Q) . . . 172
14.2. Comparison Between Two Probabilistic Classifiers . . . 176
14.2.1. Summary . . . 176
14.2.2. Brier Score . . . 177
14.2.3. Calibration Curve . . . 179
14.2.4. Paired Scoring Rule Comparison . . . 181
14.2.5. Paired t-test on Score Differences . . . 182
14.2.6. Wilcoxon Signed-Rank Test on Score Differences . . . 183
14.2.7. DeLong's Test for AUC Comparison . . . 184
14.2.8. Error Correlation . . . 184
14.2.9. Spearman Rank Correlation of Scores . . . 185
14.3. Comparison Between Two Multi-Class Classifiers . . . 188
14.3.1. Summary . . . 188
14.3.2. Agreement Matrix . . . 190
14.3.3. Disagreement Measure (Multi-Class) . . . 191
14.3.4. Bowker's and Stuart-Maxwell Tests (*) . . . 191
14.3.5. Omnibus Permutation Test (*) . . . 193
14.3.6. Cohen's Kappa (Multi-Class) . . . 194
14.3.7. Multi-Class Brier Score . . . 195
14.3.8. Multi-Class Calibration: Top-Label and Classwise ECE . . . 195
14.3.9. Multi-Class AUC and DeLong . . . 197
14.3.10. Per-Sample-Score Methods (Carry Over Unchanged) . . . 198
14.3.11. Ensemble Recommendation . . . 200

14.1. Comparison Between Two Non-Probabilistic Classifiers

Goal: Metric for dedicated comparison between *binary* classifiers' predictions.

General methods concentrate on a performance of the particular classifier. In the following, dedicated inter-classifier methods are provided.

14.1.1. Summary

The following methods compare two classifiers by analyzing their hard label predictions on a shared test set. All are built on the four counts of the contingency table (Sec. 14.1.2).

Method	Question answered
Contingency table (Sec. 14.1.2)	Where do the classifiers disagree?
Disagreement (Sec. 14.1.3)	What fraction of samples do they disagree on?
McNemar's test (Sec. 14.1.4)	Is the difference statistically significant?
Cohen's κ (Sec. 14.1.5)	How much do they agree beyond chance?
Yule's Q (Sec. 14.1.6)	How correlated are their errors?

Table 14.1.: Summary of non-probabilistic classifier comparison methods.

14.1.2. Contingency Table

Goal: Evaluate how often each classifier uniquely gets examples right that the other misses. Analog of a confusion matrix is used.

Denote the four cells entries (Fig. 14.1) as:

- n_{11} – both classifiers correct
- n_{10} – C_1 correct, C_2 incorrect
- n_{01} – C_1 incorrect, C_2 correct
- n_{00} – both classifiers incorrect

with $n_{11} + n_{10} + n_{01} + n_{00} = M$.

The accuracy of each classifier in terms of the contingency table entries is

$$\text{Accuracy}_1 = \frac{n_{11} + n_{10}}{M}, \qquad \text{Accuracy}_2 = \frac{n_{11} + n_{01}}{M} \tag{14.1}$$

The diagonal cells (n_{11}, n_{00}) represent samples on which both classifiers agree. The off-diagonal cells (n_{10}, n_{01}) are the informative ones: they count samples where exactly one classifier succeeds

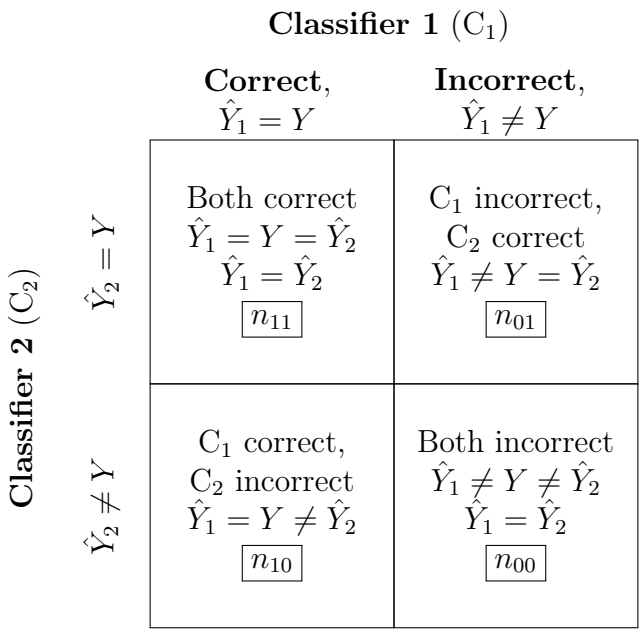


Figure 14.1.: Contingency table between two classifiers.

and the other fails. If $n_{10} \gg n_{01}$, classifier 1 is superior; if $n_{01} \gg n_{10}$, classifier 2 is superior; if $n_{10} \approx n_{01}$, they perform similarly despite possibly making errors on different samples.

Example 14.1: Two classifiers (Table 14.2) are evaluated on $M = 200$ test samples. Classifier 1 accuracy: $(150 + 25)/200 = 87.5\%$. Classifier 2 accuracy: $(150 + 15)/200 = 82.5\%$.

		Classifier 1	
		Correct	Incorrect
Classifier 2	Correct	150	15
	Incorrect	25	10

Table 14.2.: Contingency table for two classifiers on $M = 200$ samples.

The accuracies are relatively close (5% difference). However, the comparison matrix reveals a clearer picture: classifier 1 uniquely gets $n_{10} = 25$ samples right that classifier 2 misses, while classifier 2 uniquely gets only $n_{01} = 15$. Classifier 1 is correct on $25 - 15 = 10$ more "disputed" samples.

14.1.3. Disagreement Measure

Goal: Quantify the fraction of instances on which two classifiers disagree.

The disagreement measure is the ratio of instances on which the two classifiers produce different outcomes to the total number of instances:

$$\text{Dis} = \frac{n_{10} + n_{01}}{M} \tag{14.2}$$

Range: $\text{Dis} \in [0, 1]$. $\text{Dis} = 0$ indicates that the classifiers always make the same predictions (zero diversity). As $\text{Dis} \to 1$, the disagreement increases.

Example 14.1: Back to the previous example (Table 14.2), $n_{10} = 25$, $n_{01} = 15$, $M = 200$.

$$\text{Dis} = \frac{25 + 15}{200} = \frac{40}{200} = 0.2 \tag{14.3}$$

The classifiers disagree on 20% of the samples.

14.1.4. McNemar's Test

Goal: Determine whether the difference in performance between two classifiers is statistically significant.

The comparison matrix shows *how* two classifiers differ, but does not indicate whether the difference is due to chance. McNemar's test addresses this by testing the null hypothesis:

$$\begin{aligned} H_0 &: n_{10} = n_{01} \\ H_1 &: n_{10} \neq n_{01} \end{aligned} \tag{14.4}$$

i.e., the two classifiers have the same error rate, and any observed difference is due to sampling variability.

> McNemar's test only uses the off-diagonal cells (n_{10}, n_{01}). Samples where both classifiers agree provide no information about which is better.

Parametric and non-parametric forms McNemar's is the *parametric* (asymptotic) form of the test: under H_0 it is based on probabilistic assumptions. Its *non-parametric* (distribution-free) counterpart is a permutation test (Sec. 13.3.1).

p-value interpretation The p-value is the probability of observing a test statistic, assuming H_0 is true [1]. The decision is made by comparing the p-value to a predetermined *significance level* α, which is the maximum probability of incorrectly rejecting H_0 (i.e., concluding the classifiers differ when they actually do not). Common choices are $\alpha = 0.05$ (5%) and $\alpha = 0.01$ (1%).

- If $p < \alpha$, we reject H_0: the observed difference is statistically significant at level α.
- If $p \geq \alpha$, we fail to reject H_0: we cannot conclude that one classifier is better.

14.1.5. Cohen's Kappa Coefficient

Goal: Quantify the level of agreement between two classifiers, accounting for agreement expected by chance.

McNemar's test determines whether two classifiers differ significantly, but does not measure how much they agree. Cohen's kappa coefficient κ addresses this by comparing observed agreement with the agreement expected under independent (random) predictions.

[1] The calculation of the particular p-value for this test is out of the scope of this chapter. It can be found numerically by the appropriate code implementation.

Using the contingency table notation (Fig. 14.1), define the *observed* agreement as the fraction of samples on which both classifiers give the same result (both correct or both incorrect):

$$p_o = \frac{n_{11} + n_{00}}{M} \tag{14.5}$$

Recall that the accuracy of each classifier is

$$\text{Accuracy}_1 = \frac{n_{11} + n_{10}}{M}, \qquad \text{Accuracy}_2 = \frac{n_{11} + n_{01}}{M} \tag{14.6}$$

If the two classifiers were independent, the probability that both are correct on the same sample is $\text{Accuracy}_1 \cdot \text{Accuracy}_2$, and the probability that both are incorrect is $(1-\text{Accuracy}_1)(1-\text{Accuracy}_2)$. The *expected* agreement by chance is the sum of these two cases:

$$p_e = \underbrace{\text{Accuracy}_1 \cdot \text{Accuracy}_2}_{\text{both correct}} + \underbrace{(1 - \text{Accuracy}_1)(1 - \text{Accuracy}_2)}_{\text{both incorrect}} \tag{14.7}$$

Cohen's kappa is then

$$\kappa = \frac{p_o - p_e}{1 - p_e} \tag{14.8}$$

Range: $\kappa = 1$ indicates perfect agreement (the classifiers always agree). $\kappa = 0$ indicates agreement no better than chance. $\kappa < 0$ indicates agreement worse than chance (systematic disagreement).

Example 14.1: Back to the previous example (Table 14.2), $n_{11} = 150$, $n_{10} = 25$, $n_{01} = 15$, $n_{00} = 10$, $M = 200$.

$$\begin{aligned} p_o &= \frac{150 + 10}{200} = 0.80 \\ \text{Accuracy}_1 &= \frac{150 + 25}{200} = 0.875, \quad \text{Accuracy}_2 = \frac{150 + 15}{200} = 0.825 \\ p_e &= 0.875 \times 0.825 + 0.125 \times 0.175 = 0.7219 + 0.0219 = 0.7438 \\ \kappa &= \frac{0.80 - 0.7438}{1 - 0.7438} = \frac{0.0562}{0.2562} \approx 0.219 \end{aligned} \tag{14.9}$$

The low $\kappa \approx 0.22$ indicates only slight agreement beyond chance, consistent with the McNemar's test result that the classifiers do not differ significantly yet agree only modestly.

> Kappa measures agreement, not individual accuracy. Two equally poor classifiers that make the same errors on the same samples will have high κ despite low accuracy.

14.1.6. Q-Statistic (Yule's Q)

Goal: Measure the association between two classifiers' correctness using the contingency table.

While Cohen's κ measures the absolute level of agreement, the Q-statistic (Yule's Q) measures the *direction and strength of association* between the classifiers' outcomes. Using the contingency table notation (Fig. 14.1):

$$Q = \frac{n_{11}\, n_{00} - n_{10}\, n_{01}}{n_{11}\, n_{00} + n_{10}\, n_{01}} \tag{14.10}$$

Range: $Q \in [-1, 1]$.

- $Q = 1$: the classifiers are perfectly positively associated (whenever one is correct, so is the other).
- $0 < Q < 1$: the classifiers are positively correlated, meaning they tend to recognize the same instances correctly and incorrectly.
- $Q = 0$: the classifiers are independent.
- $-1 < Q < 0$: the classifiers are negatively correlated, meaning they tend to make errors on different instances.
- $Q = -1$: perfect negative association (one is correct if and only if the other is incorrect).

Example 14.1: Back to the previous example (Table 14.2), $n_{11} = 150$, $n_{10} = 25$, $n_{01} = 15$, $n_{00} = 10$.

$$Q = \frac{150 \times 10 - 25 \times 15}{150 \times 10 + 25 \times 15} = \frac{1500 - 375}{1500 + 375} = \frac{1125}{1875} = 0.6 \tag{14.11}$$

The moderate positive $Q = 0.6$ indicates that the classifiers' errors are positively correlated: they tend to succeed and fail on the same samples. Compare with $\kappa \approx 0.22$, which reflects only slight agreement beyond chance. This illustrates the distinction: Q captures the correlation direction, while κ measures absolute agreement level.

Summary of Non-Probabilistic Comparisons

For quick reference, Table 14.3 summarizes how to interpret each subrange of the four metrics.

Metric	Range	Subrange	Interpretation
Disagreement d (Sec. 14.1.3)	$[0, 1]$	$d \approx 0$	Classifiers behave nearly identically on the test set
		$d \to 1$	Classifiers make different predictions on many samples
McNemar's p-value (Sec. 14.1.4)	$[0, 1]$	$p < 0.05$	Accuracy difference is statistically significant
		$p \geq 0.05$	No evidence that the classifiers differ in accuracy
Cohen's κ (Sec. 14.1.5)	$[-1, 1]$	$\kappa < 0$	Worse than chance (systematic disagreement)
		$0 \leq \kappa < 0.20$	Slight agreement beyond chance
		$0.20 \leq \kappa < 0.40$	Fair agreement
		$0.40 \leq \kappa < 0.60$	Moderate agreement
		$0.60 \leq \kappa < 0.80$	Substantial agreement
		$0.80 \leq \kappa \leq 1$	Almost perfect agreement
Yule's Q (Sec. 14.1.6)	$[-1, 1]$	$Q = -1$	Perfect negative association: one is correct iff the other is wrong
		$-1 < Q < 0$	Errors anti-correlated, a desirable property for ensembling
		$Q \approx 0$	Classifier outcomes independent
		$0 < Q < 1$	Errors positively correlated
		$Q = 1$	Perfect positive association: both correct (or both wrong) together

Table 14.3.: Range and subrange interpretation for the non-probabilistic comparison metrics. The κ bands follow the Landis-Koch convention.

The four methods above answer related but distinct questions. The contrasts between them clarify when each one is needed:

- *Disagreement (Sec. 14.1.3) vs. McNemar's test (Sec. 14.1.4).* Disagreement reports the raw fraction of samples on which the classifiers differ. McNemar adds a significance check: it asks whether the *asymmetry* between n_{10} and n_{01} is real or could plausibly arise by chance.
  Example 14.2: Two classifier pairs ($M = 100$) share the same disagreement rate $d = 0.30$ but differ sharply on McNemar.

		C_1	
		Cor	Inc
C_2	Cor	60	15
	Inc	15	10

		C_1	
		Cor	Inc
C_2	Cor	60	3
	Inc	27	10

Symmetric (left): $d = 30/100 = 0.30$, $\chi^2 = (15-15)^2/(15+15) = 0$, $p = 1$. Asymmetric (right): $d = 0.30$, $\chi^2 = (27-3)^2/(27+3) = 19.2$, $p \approx 10^{-5}$. The same raw disagreement leads to opposite conclusions about whether the accuracy gap is real.

- *McNemar's test (Sec. 14.1.4) vs. Cohen's κ (Sec. 14.1.5).* McNemar tests whether the two classifiers have significantly different accuracy. Cohen's κ measures how much they agree beyond chance. Two classifiers can pass McNemar (no significant accuracy gap) and still have low κ if their joint agreement is no better than independent guessing.
  Example 14.3: A table with all four cells equal, $M = 100$.

		C_1	
		Cor	Inc
C_2	Cor	25	25
	Inc	25	25

  Both accuracies equal 0.50 and the off-diagonals match, so McNemar gives $\chi^2 = 0$, $p = 1$. But $p_o = (25+25)/100 = 0.50$ and $p_e = 0.5^2 + 0.5^2 = 0.50$, hence $\kappa = 0$. Passing McNemar is compatible with chance-level agreement.
- *Cohen's κ (Sec. 14.1.5) vs. Yule's Q (Sec. 14.1.6).* High Q with low κ indicates strongly correlated errors at different accuracy levels. $Q \approx 0$ with moderate κ indicates classifiers that agree often but whose errors are uncorrelated, a desirable property for ensemble methods.
  Example 14.4: Two highly accurate classifiers, $M = 100$.

		C_1	
		Cor	Inc
C_2	Cor	95	2
	Inc	2	1

  $Q = (95 \cdot 1 - 2 \cdot 2)/(95 \cdot 1 + 2 \cdot 2) = 91/99 \approx 0.92$, a strong positive association. Yet $\text{Acc}_1 = \text{Acc}_2 = 0.97$ gives $p_e = 0.97^2 + 0.03^2 = 0.9418$, and with $p_o = 0.96$ we obtain $\kappa \approx 0.31$, only fair agreement beyond chance. When accuracy is very high the chance baseline p_e is already close to 1, so κ shrinks even when Q reports strong association.
- *Disagreement (Sec. 14.1.3) vs. Yule's Q (Sec. 14.1.6).* Disagreement counts how many predictions differ. Q asks whether the *errors* line up on the same samples. Two classifiers can share an identical disagreement rate yet have very different Q, depending on how their mistakes overlap.
  Example 14.5: Two classifier pairs ($M = 100$) with identical $d = 0.20$ but opposite Q.

		C_1	
		Cor	Inc
C_2	Cor	80	10
	Inc	10	0

		C_1	
		Cor	Inc
C_2	Cor	40	10
	Inc	10	40

  Left: $d = 0.20$, $Q = (80 \cdot 0 - 10 \cdot 10)/(80 \cdot 0 + 10 \cdot 10) = -1$. Both classifiers are accurate but their few errors never coincide, an ideal regime for ensembling. Right: $d = 0.20$, $Q = (40 \cdot 40 - 10 \cdot 10)/(40 \cdot 40 + 10 \cdot 10) = 1500/1700 \approx 0.88$. The same off-diagonal mass now sits beside a large n_{00}, so Q flips from perfect anti-correlation to strong positive association.

14.2. Comparison Between Two Probabilistic Classifiers

Goal: Compare two binary classifiers using their predicted probabilities rather than hard decisions.

The non-probabilistic methods (Sec. 14.1) reduce each classifier's output to a hard label $\hat{y} \in \{0, 1\}$, discarding the confidence information. When classifiers produce probabilistic outputs

$$p_k = f_{\boldsymbol{\theta}_k}(\mathbf{x}) = \Pr\big(\hat{y} = 1 \mid \mathbf{x}\big),$$

richer comparisons are possible.

14.2.1. Summary

The following methods in Table 14.4 exploit the full predicted probability $p_i \in [0, 1]$ rather than a hard label. They are grouped into three tasks: scoring accuracy (Brier score, calibration curve, paired score difference), significance testing (paired t-test, Wilcoxon, DeLong), and diversity assessment (error correlation, Spearman rank correlation).

Method	Question answered
Scoring accuracy	*How well the predicted probabilities approximate the true labels?*
Brier score (Sec. 14.2.2)	How accurate are each classifier's predicted probabilities?
Calibration curve / ECE (Sec. 14.2.3)	Where in the probability range does calibration break down?
Paired score difference (Sec. 14.2.4)	Which classifier has better calibrated probabilities?
Significance testing	*Whether the observed difference could arise by chance?*
Paired t-test (Sec. 14.2.5)	Is the score difference statistically significant (normal δ_i)?
Wilcoxon signed-rank (Sec. 14.2.6)	Is the score difference significant (non-parametric)?
DeLong's test (Sec. 14.2.7)	Do the classifiers differ in discrimination (AUC)?
Diversity assessment	*Whether the classifiers fail on the same or different samples?*
Error correlation (Sec. 14.2.8)	How correlated are the classifiers' errors (linear)?
Spearman rank correlation (Sec. 14.2.9)	How correlated are the classifiers' errors (non-parametric)?

Table 14.4.: Summary of probabilistic classifier comparison methods.

14.2.2. Brier Score

Goal: Measure the accuracy of predicted probabilities, not just the threshold-independent performance.

AUC (Sec. 12.5) is scale-invariant: it is insensitive to the absolute predicted probability values $\hat{y}_i$ and only evaluates whether positive samples receive higher probabilities than negative ones. The Brier score complements AUC by evaluating how close the predicted probabilities are to the actual outcomes. It is defined as the MSE between the predicted probability and the actual class label:

$$\mathrm{BS} = \frac{1}{M}\sum_{i=1}^{M}(p_i - y_i)^2 \tag{14.12}$$

where $p_i \in [0, 1]$ is the predicted probability and $y_i \in \{0, 1\}$ is the actual label.

Range and reference values: $0 \leq \text{BS} \leq 1$, lower is better.

- BS $= 0$: perfect classifier, assigns probability 1 to the correct class on every sample.
- BS $= 1$: worst case, assigns probability 1 to the wrong class on every sample (confidently wrong).
- BS $= 0.25$: constant prediction $p_i = 0.5$ for all samples, regardless of the label.
- BS $= \pi_1(1-\pi_1)$: *base-rate classifier* that always predicts the marginal positive rate $p_i = \pi_1 = \Pr(y = 1)$ as a probability. This is the optimal constant-probability predictor (it minimizes expected Brier among all constants) and serves as the natural "no-information" probabilistic baseline: on a balanced set it equals 0.25; on an imbalanced set ($\pi_1 = 0.1$) it drops to 0.09, so a small Brier score on imbalanced data is not automatically impressive.
- BS $= \min(\pi_1, 1 - \pi_1)$: *majority-vote classifier* that always predicts the hard majority label ($p_i = 0$ if $\pi_1 < 0.5$, else $p_i = 1$). It is the accuracy-optimal constant classifier but is *worse* than the base-rate baseline as a probabilistic predictor: at $\pi_1 = 0.1$ it gives 0.10 versus 0.09 for the base-rate classifier.
- Skill score: $\text{BSS} = 1 - \text{BS}/\text{BS}_{\text{ref}}$ with $\text{BS}_{\text{ref}} = \pi_1(1 - \pi_1)$ rescales the score so that BSS $= 1$ is perfect, BSS $= 0$ is no better than the base-rate baseline, and BSS < 0 is worse.

Example 14.6: Consider $M = 4$ samples with actual labels and predicted probabilities:

y_i	$\hat{y}_i$	$(p_i - y_i)^2$
1	0.9	0.01
0	0.2	0.04
1	0.7	0.09
0	0.1	0.01

$$\text{BS} = \frac{0.01 + 0.04 + 0.09 + 0.01}{4} = 0.0375 \tag{14.13}$$

The low Brier score indicates well-calibrated probability predictions.

Comparison with AUC:

- AUC measures *discrimination*: do positive samples receive higher predicted probabilities $f_{\mathbf{w}}(\mathbf{x}_i)$ than negative ones? It is scale-invariant and threshold-invariant.
- Brier score measures *calibration*: are the predicted probabilities p_i close to the actual labels y_i? It is sensitive to the absolute probability values.

A classifier can have high AUC (good discrimination) but poor Brier score (poorly calibrated probabilities), or vice versa.

Fig. 14.2 illustrates this distinction. Both classifiers achieve the same AUC (all positive samples are scored higher than all negative samples), but Classifier A predicts probabilities close to the actual labels ($y_i = 0$ or $y_i = 1$), yielding a low Brier score. Classifier B compresses its predictions toward 0.5, degrading calibration and increasing the Brier score, while maintaining the same discrimination ability.

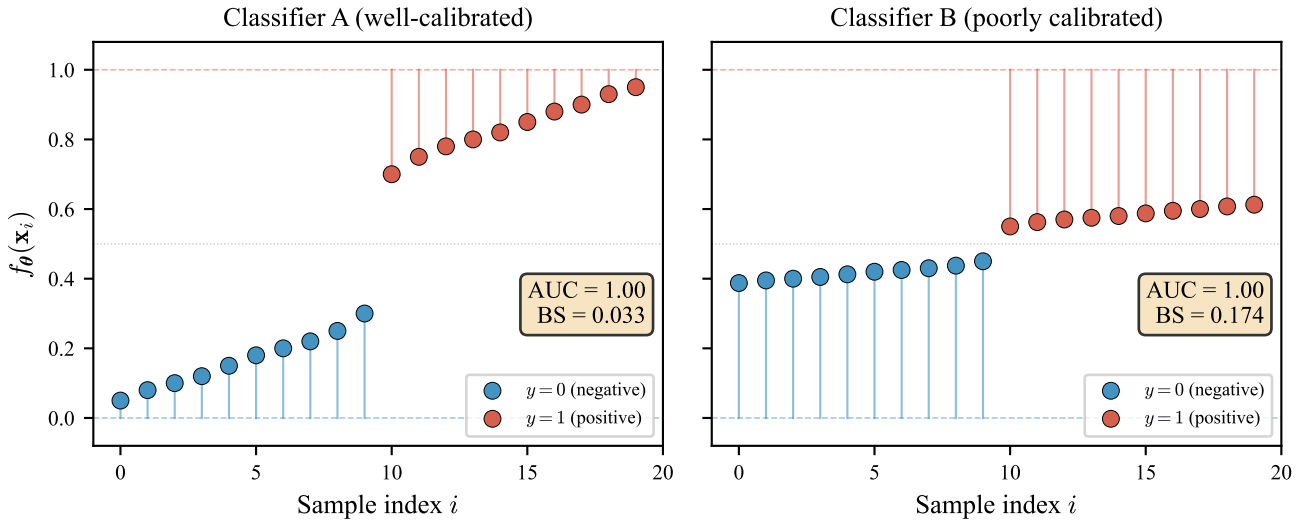


Figure 14.2.: Two classifiers with identical AUC but different Brier scores. Vertical lines show the gap between predicted probability $f_{\mathbf{w}}(\mathbf{x}_i)$ and actual label y_i; longer lines indicate larger per-sample errors. Dashed lines mark the ideal predictions (0 and 1).

14.2.3. Calibration Curve

Goal: Visually diagnose calibration (do predicted probabilities match the observed frequencies of positives?) and quantify the gap between two classifiers via the expected calibration error (ECE).

The Brier score (Sec. 14.2.2) summarizes calibration with a single number. The *calibration curve* (also called *reliability diagram*) shows *where* along the probability range the classifier is miscalibrated.

Construction. The construction is a `probability`-type histogram (Sec. 1.4) of the predicted probabilities $\{p_i\}_{i=1}^{M}$, but with each bin reporting the empirical positive rate $\bar{y}_b$ instead of the relative frequency n_i/N of eq. (1.18).

1. Partition the predicted probabilities $\{p_i\}_{i=1}^{M}$ into B bins $I_1, \ldots, I_B$ over $[0, 1]$ (equal-width or equal-count/quantile).
2. For each bin b, compute the mean predicted probability and the observed positive fraction,

$$\bar{p}_b = \frac{1}{|I_b|} \sum_{i \in I_b} p_i, \qquad \bar{y}_b = \frac{1}{|I_b|} \sum_{i \in I_b} y_i \tag{14.14}$$

3. Plot $\bar{y}_b$ versus $\bar{p}_b$, together with the diagonal $\bar{y} = \bar{p}$ as the reference for perfect calibration.

Interpretation.

- Curve *on* the diagonal: well-calibrated. A predicted probability of 0.7 corresponds to a positive in 70% of such cases.
- Curve *below* the diagonal ($\bar{y}_b < \bar{p}_b$): the classifier is *over-confident*, i.e. predicted probabilities exceed the realized frequencies.

- Curve *above* the diagonal ($\bar{y}_b > \bar{p}_b$): the classifier is *under-confident*.
- Deviations are common (e.g., naive Bayes, SVM with Platt scaling) and can be corrected by post-hoc methods such as *Platt scaling* or *isotonic regression*[2].

Example 14.7: Fig. 14.3 compares two classifiers on a synthetic 1D dataset ($M = 2000$): a logistic regression (well-calibrated by construction) and an over-confident variant obtained by sharpening the logits, $\tilde{p} = \sigma(\alpha \cdot \text{logit}(p))$ with $\alpha = 3$. Because $\tilde{p}$ is a strictly monotonic function of p, the two classifiers order the samples identically: any pair (i, j) with $p_i > p_j$ also satisfies $\tilde{p}_i > \tilde{p}_j$. Since AUC depends only on this ranking and not on the absolute probability values, both classifiers achieve the same AUC (Sec. 12.5). The over-confident one nevertheless produces a calibration curve that bows below the diagonal in the mid-probability range, a markedly higher Brier score, and a higher Expected Calibration Error (both BS and ECE are reported in the figure legend).

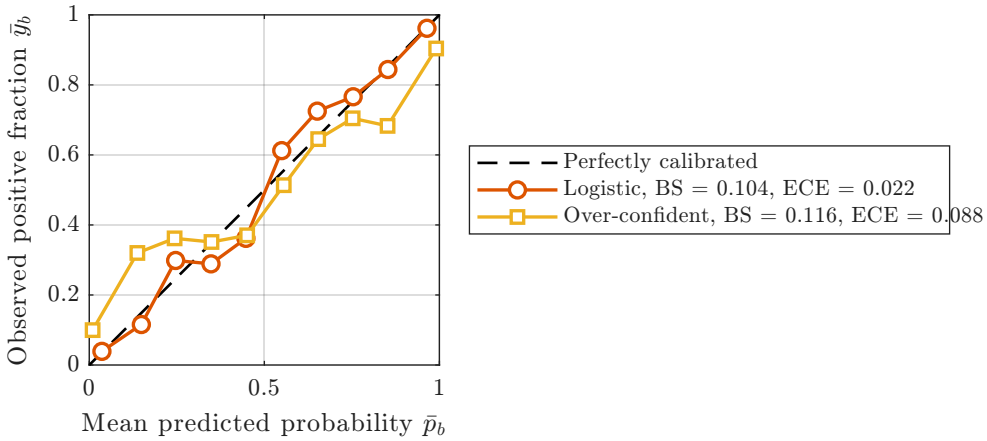


Figure 14.3.: Calibration curve (reliability diagram) for a well-calibrated logistic regression and an over-confident variant. The dashed diagonal marks perfect calibration; points below the diagonal indicate over-confidence. The legend reports the Brier score (BS) and Expected Calibration Error (ECE) of each classifier.

Expected Calibration Error. The calibration curve summarizes to a single scalar by weighting each bin's deviation from the diagonal by its share of samples:

$$\text{ECE}_k = \sum_{b=1}^{B} \frac{|I_b|}{M} \, |\bar{y}_b - \bar{p}_b| \tag{14.15}$$

A lower ECE indicates better calibration. Comparing ECE_1 and ECE_2 complements the Brier score by isolating the calibration component from the refinement (sharpness) component.

Relation to AUC and Brier score:

- AUC (Sec. 12.5) is invariant to any monotonic rescaling of p_i, so a classifier can have $\text{AUC} \approx 1$ while being severely miscalibrated.
- Brier score gives a scalar summary of calibration plus refinement; the calibration curve shows the *shape* of the miscalibration across probability ranges, which is the actionable information for choosing a recalibration method.

[2]In Python: `sklearn.calibration.calibration_curve` and `CalibratedClassifierCV`.

The Brier score decomposes into calibration and refinement (*sharpness*) terms. *Sharpness* refers to how concentrated the predicted probabilities are toward the extremes 0 and 1: a *sharp* classifier commits to confident predictions (most p_i near 0 or 1), while an *unsharp* one hedges with probabilities clustered near the base rate (e.g., 0.5). Two classifiers with similar Brier scores can have very different calibration curves: one may be well-calibrated but unsharp, the other miscalibrated but sharp. The calibration curve reveals this distinction, which the Brier score alone cannot.

Example 14.8: Two probabilistic classifiers are evaluated on $M = 6$ samples with labels y and predicted probabilities p_1, p_2:

i	1	2	3	4	5	6
y_i	1	0	1	0	1	0
$p_{1,i}$	0.90	0.20	0.70	0.30	0.60	0.15
$p_{2,i}$	0.75	0.10	0.85	0.40	0.80	0.25

With $B = 2$ bins, $I_1 = [0, 0.5)$ and $I_2 = [0.5, 1]$:

Bin	$\bar{p}_{b,1}$	$\bar{y}_{b,1}$	$\bar{p}_{b,2}$	$\bar{y}_{b,2}$
[0, 0.5)	0.217	0	0.250	0
[0.5, 1]	0.733	1	0.800	1

$\text{ECE}_1 = \frac{3}{6}|0 - 0.217| + \frac{3}{6}|1 - 0.733| = 0.108 + 0.133 = 0.242.$
$\text{ECE}_2 = \frac{3}{6}|0 - 0.250| + \frac{3}{6}|1 - 0.800| = 0.125 + 0.100 = 0.225.$

Classifier 2 has slightly better calibration. Both classifiers are underconfident in the upper bin ($\bar{p}_b < \bar{y}_b = 1$) and overconfident in the lower bin ($\bar{p}_b > \bar{y}_b = 0$).

14.2.4. Paired Scoring Rule Comparison

Goal: Quantify which classifier produces better-calibrated probabilities on a per-sample basis.

Given a proper scoring rule S , such as the Brier score $S_{\text{BS}}(p, y) = (p - y)^2$ (Sec. 14.2.2) or the CE loss $S_{\text{LL}}(p, y) = -[y \log p + (1-y) \log(1-p)]$ (Sec. 8.4), denote the per-sample score $S_{k,i} = S(p_{k,i}, y_i)$ and compute the score difference:

$$\delta_i = S_{1,i} - S_{2,i} \tag{14.16}$$

where $p_{k,i} = f_{\boldsymbol{\theta}_k}(\mathbf{x}_i)$ is the probability predicted by classifier k for sample i. If $\delta_i > 0$, classifier 2 is better on that sample; if $\delta_i < 0$, classifier 1 is better.

The average score difference summarizes the overall comparison:

$$\bar{\delta} = \frac{1}{M} \sum_{i=1}^{M} \delta_i = \bar{S}_1 - \bar{S}_2 \tag{14.17}$$

where

$$\bar{S}_k = \frac{1}{M} \sum_{i=1}^{M} S_{k,i} \tag{14.18}$$

is the mean score of classifier k. A negative $\bar{\delta}$ favors classifier 1; a positive $\bar{\delta}$ favors classifier 2.

Example 14.9 (Paired Scoring Rule Comparison): Two probabilistic classifiers are evaluated on $M = 6$ test samples using the Brier score $S_{\text{BS}}(p, y) = (p - y)^2$:

y_i	$p_{1,i}$	$S_{\text{BS},1}$	$p_{2,i}$	$S_{\text{BS},2}$	δ_i
1	0.90	0.010	0.75	0.063	-0.053
0	0.20	0.040	0.10	0.010	$+0.030$
1	0.70	0.090	0.85	0.023	$+0.068$
0	0.30	0.090	0.40	0.160	-0.070
1	0.60	0.160	0.80	0.040	$+0.120$
0	0.15	0.023	0.25	0.063	-0.040
	$\bar{S}_1$	0.069	$\bar{S}_2$	0.060	$\bar{\delta} = +0.009$

The small positive $\bar{\delta} = 0.009$ slightly favors classifier 2, but the per-sample differences δ_i alternate in sign, suggesting the advantage is not consistent. A statistical test is needed to determine whether this difference is significant.

14.2.5. Paired t-test on Score Differences

Goal: Test whether the scoring difference between two classifiers is statistically significant, assuming approximately normal score differences.

The paired scoring rule comparison provides a descriptive measure $\bar{\delta}$, but does not indicate whether the difference could have arisen by chance. A paired statistical test addresses this by testing:

- H_0: $\mathbb{E}[\delta_i] = 0$ — the classifiers are equally good.
- H_1: $\mathbb{E}[\delta_i] \neq 0$ — one classifier is systematically better.

The paired t-test on $\{\delta_1, \ldots, \delta_M\}$ yields a test statistic:

$$t = \frac{\bar{\delta}}{s_\delta / \sqrt{M}} \tag{14.19}$$

where s_δ is the *unbiased* sample standard deviation:

$$s_\delta = \sqrt{\frac{1}{M-1} \sum_{i=1}^{M} (\delta_i - \bar{\delta})^2} \tag{14.20}$$

If $p < \alpha$, reject H_0: the scoring difference is statistically significant.

The paired t-test assumes that the δ_i are approximately normally distributed. When this assumption is violated (e.g., small M, heavy-tailed or skewed score distributions), the Wilcoxon signed-rank test (below) provides a non-parametric alternative.

Example 14.9: Back to the previous example (Table in Example 14.9), $\bar{\delta} = 0.009$, $s_\delta = 0.072$, $M = 6$.

$$t = \frac{0.009}{0.072/\sqrt{6}} = \frac{0.009}{0.029} \approx 0.31 \tag{14.21}$$

The corresponding p-value is $p \approx 0.77$. Since $p = 0.77 > 0.05$, we fail to reject H_0. Despite the slight difference in mean Brier score, the two classifiers are not significantly different.

14.2.6. Wilcoxon Signed-Rank Test on Score Differences

Goal: Non-parametric alternative to the paired t-test: test whether the median score difference is zero, without assuming normality.

The Wilcoxon signed-rank test uses the ranks of $|\delta_i|$ rather than their magnitudes, making it robust to outliers and non-normal distributions. The procedure is:

1. Discard any samples where $\delta_i = 0$ (ties). Let M' denote the remaining count.
2. Rank the absolute differences $|\delta_1|, \ldots, |\delta_{M'}|$ from smallest to largest. Assign average ranks to tied values.
3. Compute the signed-rank sums:

$$W^+ = \sum_{\delta_i > 0} R_i, \qquad W^- = \sum_{\delta_i < 0} R_i \tag{14.22}$$

   where R_i is the rank of $|\delta_i|$. The test statistic is $W = \min(W^+, W^-)$.

The hypotheses are:

- H_0: the median of δ_i is zero — the classifiers are equally good.
- H_1: the median of δ_i is not zero — one classifier is systematically better.

Under H_0, positive and negative differences are equally likely at each rank, so $W^+ \approx W^-$. A very small W indicates that one sign dominates the high ranks, i.e., one classifier is consistently better on the samples with the largest differences. If $p < \alpha$, reject H_0.

Example 14.9: Back to the previous example (Table in Example 14.9):

δ_i	$\lvert\delta_i\rvert$	Rank R_i	Sign
$+0.030$	0.030	1	$+$
-0.040	0.040	2	$-$
-0.053	0.053	3	$-$
$+0.068$	0.068	4	$+$
-0.070	0.070	5	$-$
$+0.120$	0.120	6	$+$

$W^+ = 1 + 4 + 6 = 11$, $W^- = 2 + 3 + 5 = 10$, $W = \min(11, 10) = 10$.

For $M' = 6$, the critical value at $\alpha = 0.05$ (two-sided) is $W_{\text{crit}} = 0$. Since $W = 10 > 0$, we fail to reject H_0, consistent with the paired t-test result. The positive and negative ranks are nearly balanced, confirming that neither classifier is systematically better.

14.2.7. DeLong's Test for AUC Comparison

Goal: Test whether two classifiers have significantly different discrimination ability (AUC).

The paired scoring rule test compares calibration (how close probabilities are to the true labels). DeLong's test addresses a complementary question: do the classifiers differ in their ability to *rank* positive samples above negative ones, as measured by the AUC (Sec. 12.5)?

Given two classifiers with AUC_1 and AUC_2 evaluated on the same test set, the hypotheses are:

- H_0: $\mathrm{AUC}_1 = \mathrm{AUC}_2$.
- H_1: $\mathrm{AUC}_1 \neq \mathrm{AUC}_2$.

The DeLong test statistic is:

$$z = \frac{\mathrm{AUC}_1 - \mathrm{AUC}_2}{\sqrt{\mathrm{Var}(\mathrm{AUC}_1 - \mathrm{AUC}_2)}} \tag{14.23}$$

Under H_0, z is approximately standard normal. The variance in the denominator accounts for the correlation between AUC_1 and AUC_2, since both are computed on the *same* test samples and their estimates are not independent[3]. If $p < \alpha$, the classifiers have significantly different discrimination.

> DeLong's test evaluates *discrimination* (ranking ability) while the paired scoring rule test evaluates *calibration* (probability accuracy). Two classifiers can have identical AUC but different Brier scores, or vice versa. Both tests should be considered for a complete comparison.

14.2.8. Error Correlation

Goal: Measure how correlated the errors of two probabilistic classifiers are. It is the probabilistic analog of Yule's Q.

Yule's Q (Sec. 14.1) measures the association between two classifiers' binary correct/incorrect outcomes. For probabilistic classifiers, we can measure the same concept on the continuous per-sample scores $S_{k,i}$: when classifier 1 has a large error, does classifier 2 also have a large error?

The error correlation is the Pearson correlation coefficient between the per-sample scores:

$$r = \frac{\sum_{i=1}^{M}(S_{1,i} - \bar{S}_1)(S_{2,i} - \bar{S}_2)}{\sqrt{\sum_{i=1}^{M}(S_{1,i} - \bar{S}_1)^2}\,\sqrt{\sum_{i=1}^{M}(S_{2,i} - \bar{S}_2)^2}} \tag{14.24}$$

Range: $r \in [-1, 1]$.

- $r \approx 1$: errors are positively correlated — both classifiers fail on the same samples. Combining them provides little benefit.
- $r \approx 0$: errors are uncorrelated — ensembles can potentially average out errors effectively.
- $r < 0$: errors are negatively correlated — the classifiers fail on different samples, which gives high potential for ensembles.

[3] The computation of DeLong's variance is based on placement values and is outside the scope of this chapter. It is available numerically via standard implementations (e.g., `scipy.stats`).

Example 14.9: Back to the previous example (Table in Example 14.9), the per-sample Brier scores are:

i	$S_{\text{BS},1}$	$S_{\text{BS},2}$
1	0.010	0.063
2	0.040	0.010
3	0.090	0.023
4	0.090	0.160
5	0.160	0.040
6	0.023	0.063

Computing the Pearson correlation: $r \approx -0.18$. The weak negative correlation indicates that the classifiers' errors are slightly complementary — when one has a large error, the other tends to have a smaller one. This suggests moderate potential for improvement via ensemble combination (Ch. 15.3).

14.2.9. Spearman Rank Correlation of Scores

Goal: Non-parametric alternative to the error correlation: measure whether two classifiers rank samples by difficulty in the same order, without assuming a linear relationship between their scores.

The Pearson error correlation (Sec. 14.2.8) measures the *linear* association between $S_{1,i}$ and $S_{2,i}$. When the relationship is monotonic but non-linear, or when scores contain outliers, the Spearman rank correlation provides a more robust measure.

Let $R_{k,i}$ denote the rank of $S_{k,i}$ among $\{S_{k,1}, \ldots, S_{k,M}\}$ (with average ranks for ties). The Spearman rank correlation (Sec. 2.5) is the Pearson correlation computed on the ranks:

$$r_s = \frac{\sum_{i=1}^{M}(R_{1,i} - \bar{R})(R_{2,i} - \bar{R})}{\sqrt{\sum_{i=1}^{M}(R_{1,i} - \bar{R})^2}\,\sqrt{\sum_{i=1}^{M}(R_{2,i} - \bar{R})^2}} \tag{14.25}$$

where $\bar{R} = (M+1)/2$ is the mean rank.

Range: $r_s \in [-1, 1]$. The interpretation mirrors the Pearson error correlation:

- $r_s \approx 1$: classifiers find the same samples difficult — limited ensemble benefit.
- $r_s \approx 0$: difficulty rankings are unrelated — good ensemble diversity.
- $r_s < 0$: when one classifier struggles, the other excels — high ensemble potential.

> The Spearman rank correlation relates to the Pearson error correlation as the Wilcoxon test relates to the paired t-test: a non-parametric alternative that is robust to outliers and non-linear score distributions.

Example 14.9: Back to the previous example (Table in Example 14.9). Ranking each classifier's Brier scores (average ranks for ties):

i	$S_{\text{BS},1}$	$R_{1,i}$	$S_{\text{BS},2}$	$R_{2,i}$
1	0.010	1	0.063	4.5
2	0.040	3	0.010	1
3	0.090	4.5	0.023	2
4	0.090	4.5	0.160	6
5	0.160	6	0.040	3
6	0.023	2	0.063	4.5

Computing $r_s \approx -0.18$, consistent with the Pearson result. The classifiers do not rank sample difficulty similarly, confirming modest complementarity.

Summary of Probabilistic Comparisons

For quick reference, Table 14.5 summarizes how to interpret each subrange of the probabilistic comparison metrics.

Method	Range	Subrange	Interpretation
Brier score (Sec. 14.2.2)	[0, 1]	BS ≈ 0	Probabilities track the labels accurately
		BS $\approx \pi_1(1-\pi_1)$	No better than the base-rate (no-information) baseline
		BS $\to 1$	Confidently wrong on most samples
ECE (Sec. 14.2.3)	[0, 1]	ECE ≈ 0	Bin-averaged probabilities match empirical frequencies
		ECE $\gtrsim 0.1$	Visible miscalibration in at least one probability band
Paired score $\bar{\delta}$ (Sec. 14.2.4)	$\mathbb{R}$	$\bar{\delta} < 0$	Classifier 1 scores better (lower loss) on average
		$\bar{\delta} \approx 0$	Classifiers tied on the chosen scoring rule
		$\bar{\delta} > 0$	Classifier 2 scores better on average
Paired t-test p (Sec. 14.2.5)	[0, 1]	$p < 0.05$	Mean score gap is significant (assumes δ_i approximately normal)
		$p \geq 0.05$	No evidence the mean score differs
Wilcoxon p (Sec. 14.2.6)	[0, 1]	$p < 0.05$	Median score gap is significant (non-parametric)
		$p \geq 0.05$	No evidence the median score differs
DeLong p (Sec. 14.2.7)	[0, 1]	$p < 0.05$	AUC (discrimination) differs significantly
		$p \geq 0.05$	No evidence the classifiers rank samples differently
Pearson error corr. r (Sec. 14.2.8)	$[-1, 1]$	$r < 0$	Errors anti-correlated, high ensemble potential
		$r \approx 0$	Errors uncorrelated, good ensemble diversity
		$0 < r < 0.6$	Mildly correlated errors, modest ensemble benefit
		$r \geq 0.6$	Strongly correlated errors, little ensemble benefit
Spearman r_s (Sec. 14.2.9)	$[-1, 1]$	$r_s < 0$	Difficulty rankings reversed, high ensemble potential
		$r_s \approx 0$	Rankings unrelated, good diversity
		$r_s \gtrsim 0.6$	Same samples found hard by both, limited ensemble gain

Table 14.5.: Range and subrange interpretation for the probabilistic comparison metrics. The 0.05 cutoff and the 0.1 ECE rule of thumb follow chapter conventions.

The methods above answer related but distinct questions. The contrasts between them clarify when each one is presented:

- *Brier score (Sec. 14.2.2) vs. ECE (Sec. 14.2.3).* Brier mixes calibration and refinement into one scalar. ECE isolates the calibration component by bin, so it reveals *where* along the probability range miscalibration concentrates; two classifiers with similar Brier can have very different ECE shapes.
- *Paired score $\bar{\delta}$ (Sec. 14.2.4) vs. paired t-test (Sec. 14.2.5).* $\bar{\delta}$ is the descriptive mean gap. The t-test asks whether that gap could plausibly arise by chance; a small $|\bar{\delta}|$ with large per-sample variance can be non-significant, and conversely a tiny but consistent $\bar{\delta}$ on a large M can be highly significant.
- *Paired t-test (Sec. 14.2.5) vs. Wilcoxon (Sec. 14.2.6).* The t-test compares means under approximate normality of the δ_i. Wilcoxon ranks the magnitudes, so it is robust to heavy tails and outliers; the two can disagree when a few extreme δ_i dominate the mean.
- *DeLong (Sec. 14.2.7) vs. paired t-test on Brier (Sec. 14.2.5).* DeLong tests AUC, a ranking property invariant to monotone rescaling. The Brier-difference t-test tests calibrated accuracy. A classifier can keep DeLong's verdict unchanged after a sharpening transform that destroys Brier (Fig. 14.3).
- *Pearson error correlation (Sec. 14.2.8) vs. Spearman rank correlation (Sec. 14.2.9).* Pearson measures the linear association between per-sample error scores; Spearman uses only ranks. They diverge whenever the score-error relationship is monotonic but non-linear, or when a few large errors inflate the Pearson value.
- *Scoring accuracy (Brier, ECE, $\bar{\delta}$) vs. diversity (r, r_s).* The first family asks how good each classifier is individually; the second asks whether their failures overlap. A pair of strong but redundant classifiers ($r \approx 1$) gives less ensemble benefit than a pair of weaker but complementary ones ($r \approx 0$).

14.3. Comparison Between Two Multi-Class Classifiers

Goal: Extend the pairwise comparison methods of Sec. 14.1 and Sec. 14.2 from binary labels $\hat{y} \in \{0, 1\}$ to K classes.

In the binary setting, the joint behaviour of two classifiers is captured by a 2×2 contingency table (Sec. 14.1.2). With $K \geq 2$ classes, the natural object is a $K \times K$ *agreement matrix*, whose entries count samples by the pair of predicted labels.

14.3.1. Summary

Table 14.6 maps each binary comparison method to its multi-class counterpart. Methods marked "same formula" need no modification; per-sample-score methods (Sec. 14.2.4, 14.2.5, 14.2.6, 14.2.8, 14.2.9) apply unchanged because they operate on scalar scores $S_{k,i}$ that do not see the class count K.

Question / aspect	$K = 2$ method	$K > 2$ method
Joint prediction table		
Where do classifiers disagree?	Contingency table (Sec. 14.1.2; uses y)	Agreement matrix (Sec. 14.3.2; no y)
Fraction of disagreements	Disagreement (Sec. 14.1.3)	Disagreement (Sec. 14.3.3; same formula)
Significance testing		
Is the accuracy difference significant?	McNemar (Sec. 14.1.4)	apply on the correct/incorrect 2×2 collapse
Per-pair confusion symmetry	(degenerate at $K = 2$)	Bowker (Sec. 14.3.4)
Equal class-prediction rates	(degenerate at $K = 2$)	Stuart-Maxwell (Sec. 14.3.4)
Distribution-free difference test	permutation McNemar (Sec. 14.1.4)	Omnibus permutation (Sec. 14.3.5)
Agreement and error correlation		
Agreement beyond chance	Cohen's κ (Sec. 14.1.5)	Cohen's κ (Sec. 14.3.6)
Direction of error correlation	Yule's Q (Sec. 14.1.6)	apply binary Q to correct/incorrect 2×2 collapse
Probabilistic scoring		
Probability-vector accuracy	Brier (Sec. 14.2.2)	Multi-class Brier (Sec. 14.3.7)
Calibration diagnosis	Calibration curve / ECE (Sec. 14.2.3)	Top-label / classwise ECE (Sec. 14.3.8)
Discrimination significance (AUC)	DeLong (Sec. 14.2.7)	Multi-class AUC + DeLong, OvR or OvO (Sec. 14.3.9)
Per-sample-score methods (unchanged)		
Paired score difference, t-test, Wilcoxon, Pearson, Spearman	Sec. 14.2.4, 14.2.5, 14.2.6, 14.2.8, 14.2.9	Apply unchanged to the scalar score $S_{k,i}$ (multi-class Brier or cross-entropy); see Sec. 14.3.10

Table 14.6.: Correspondence between binary ($K = 2$) and multi-class ($K > 2$) classifier comparison methods. Rows marked "same formula" need no modification; the per-sample-score row covers several sections collectively because they operate on scalar per-sample scores and do not see K.

14.3.2. Agreement Matrix

Goal: Tabulate the joint predictions of two K-class classifiers on a shared test set.

Let $\hat{y}_{k,i} \in \{1, \dots, K\}$ denote the prediction of classifier k on sample i. The agreement matrix $N = [n_{jk}] \in \mathbb{N}^{K \times K}$ has entries

$$n_{jk} = \sum_{i=1}^{M} \mathbb{1}[\hat{y}_{1,i} = j]\, \mathbb{1}[\hat{y}_{2,i} = k], \qquad \sum_{j,k} n_{jk} = M. \tag{14.26}$$

The diagonal entries n_{jj} count samples on which both classifiers predict class j, and their sum $\sum_{j=1}^{K} n_{jj}$ is the total number of samples on which the two classifiers issue the same label (whether correct or not). The off-diagonal entry n_{jk} with $j \neq k$ counts samples where C_1 predicts class j and C_2 predicts class k, revealing *which* class confusions drive the disagreement.

Confusion, contingency, agreement

Note the differences between:

Confusion matrix	Sec. 12.2	Performance assessment of a classifier
Contingency table	Sec. 14.1.2	Compare performance of two binary classifiers
Agreement matrix	Current section	Compare outcome of two multi-class classifiers

Agreement matrix

The agreement matrix does *not* reference the ground truth y: both axes are classifier *predictions*, $\hat{y}_1$ and $\hat{y}_2$. It is therefore a fundamentally different object from the 2×2 contingency table of Sec. 14.1.2, whose axes are correct/incorrect (i.e. $\hat{y}_k = y$ vs. $\hat{y}_k \neq y$).

- Contingency table (Sec. 14.1.2): *includes* y. Tells you which classifier is more often *right*.
- Agreement matrix (this section): *excludes* y. Tells you how the classifiers *differ from each other*.

One can be computed without labels (e.g. on a live, unlabelled stream); the other cannot. Collapsing the agreement matrix by "predicted label equals y" would recover the contingency table, but the two are used for opposite purposes and should not be confused.

Example 14.10: Two classifiers are evaluated on $M = 200$ samples drawn from $K = 3$ classes (A, B, C). The agreement matrix is:

		Classifier 2 (C_2)			
		A	B	C	Row sum
Classifier 1 (C_1)	A	70	6	4	80
	B	10	55	5	70
	C	8	7	35	50
	Col. sum	88	68	44	200

Table 14.7.: Agreement matrix between two 3-class classifiers on $M = 200$ samples.

The diagonal sums to $70+55+35 = 160$, so the classifiers issue the same label on $160/200 = 80\%$ of samples. The largest off-diagonal cell is $n_{BA} = 10$: among the 40 disagreements, the most frequent pattern is "C_1 says B while C_2 says A."

14.3.3. Disagreement Measure (Multi-Class)

Goal: Quantify the fraction of samples on which two K-class classifiers issue different labels.

The binary formula of Sec. 14.1.3 generalizes directly:

$$\text{Dis} = \frac{1}{M}\sum_{i=1}^{M} \mathbb{1}[\hat{y}_{1,i} \neq \hat{y}_{2,i}] = 1 - \frac{1}{M}\sum_{j=1}^{K} n_{jj}, \tag{14.27}$$

where n_{jj} are the diagonal entries of the agreement matrix (Sec. 14.3.2) that count the samples on which both classifiers predict the same class j.

Example 14.10: Back to the previous example (Table 14.7):

$$\text{Dis} = 1 - \frac{160}{200} = 0.20. \tag{14.28}$$

The classifiers disagree on 20% of the samples.

14.3.4. Bowker's and Stuart-Maxwell Tests (*)

Goal: Diagnose whether two K-class classifiers behave differently from each other on a shared test set, without requiring ground truth y. Two questions are answered separately:

- Do they confuse *specific class pairs* asymmetrically (Bowker)?
- Do they predict each class at *different overall rates* (Stuart-Maxwell)?

Unlike McNemar's test (Sec. 14.1.4), which uses the correct/incorrect contingency table and answers "which classifier is more accurate," Bowker and Stuart-Maxwell operate on the agreement matrix of Sec. 14.3.2, where ground truth y is absent. They cannot rank classifiers by accuracy; they detect whether the two classifiers behave systematically differently from each other. For multi-class accuracy comparison, apply McNemar unchanged to the correct/incorrect 2×2 collapse: each prediction is right or wrong regardless of K.

The actionable outcomes are: (i) detect divergence between two models before labels are available (e.g. champion vs. challenger on live traffic); (ii) catch prior or threshold drift between two versions of a model (Stuart-Maxwell); (iii) localize *which* class pairs drive disagreement, by inspecting the per-pair Bowker contributions before summation (debugging); and (iv) decide whether an ensemble of the two should use symmetric (majority vote) or class-asymmetric weighting.

With K classes there are $K(K-1)/2$ off-diagonal pairs, and the single binary equality $n_{10} = n_{01}$ splits into two distinct, non-equivalent null hypotheses:

- *Cell-wise symmetry.* Every individual pair is balanced: $n_{jk} = n_{kj}$ for all $j < k$. This asks whether each specific class-confusion pattern (e.g. "C_1 predicts A while C_2 predicts B") occurs as often as its mirror ("C_1 predicts B while C_2 predicts A"). Bowker's test addresses this null. Rejecting H_0 means that at least one off-diagonal pair (j, k) is directionally biased:

one classifier systematically substitutes class j for class k more often than the other does the reverse, so the two classifiers do not make the same kinds of confusions in the same proportions.

- *Marginal homogeneity.* For each class j, the row marginal $\sum_k n_{jk}$ (number of times C_1 predicts class j) equals the column marginal $\sum_k n_{kj}$ (number of times C_2 predicts class j). This asks only whether the two classifiers issue each label at the same overall rate, without constraining *which* pairs of classes are confused: an excess of "C_1 says A, C_2 says B" can be cancelled out in the A-marginal by an excess of "C_1 says C, C_2 says A", leaving the marginals balanced while individual off-diagonal pairs are not. Stuart-Maxwell's test addresses this null. Rejecting H_0 means that at least one class is predicted at a different overall rate by the two classifiers, indicating an aggregate bias in class usage (e.g. C_1 predicts class A more often than C_2 does, regardless of which other class C_2 substitutes in its place).

The two coincide at $K = 2$ (where each marginal equation is equivalent to the single off-diagonal equality), but for $K \geq 3$ cell-wise symmetry implies marginal homogeneity and not vice versa: asymmetries on different off-diagonal pairs can cancel in the marginals.

Bowker's test of symmetry. H_0: $n_{jk} = n_{kj}$ for every $j < k$, i.e. the agreement matrix is symmetric about its diagonal:

$$\chi^2_{\mathrm{B}} = \sum_{1 \leq j < k \leq K} \frac{(n_{jk} - n_{kj})^2}{n_{jk} + n_{kj}}, \qquad \mathrm{df} = \frac{K(K-1)}{2}. \tag{14.29}$$

The degrees of freedom $K(K-1)/2$ count the number of off-diagonal pairs above the diagonal, i.e. the number of distinct equalities $n_{jk} = n_{kj}$ being tested; it is the parameter passed to the χ^2 table (or to `scipy.stats.chi2.sf`) to obtain the p-value. At $K = 2$ this collapses to McNemar's statistic (Sec. 14.1.4)[4].

Stuart-Maxwell test of marginal homogeneity. A weaker null only requires that the row marginals match the column marginals:

$$H_0: \sum_{k=1}^{K} n_{jk} = \sum_{k=1}^{K} n_{kj} \quad \text{for all } j. \tag{14.30}$$

The test statistic χ^2_{SM} combines the K marginal differences into a single number using a covariance correction (one marginal is redundant because the totals must match), and is referred to a χ^2 distribution with $\mathrm{df} = K - 1$. Standard implementations such as `SquareTable.homogeneity` compute it directly from N. The p-value is interpreted as in Sec. 14.1.4: reject H_0 when $p < \alpha$.

> Bowker's test asks whether every *individual* confusion pattern $(j, k) \leftrightarrow (k, j)$ is balanced. Stuart-Maxwell asks only whether the two classifiers issue each class *at the same overall rate*. The two tests can disagree: a strongly asymmetric individual pair ($n_{jk} \gg n_{kj}$) can be masked by a compensating asymmetry on another pair, leaving the marginals balanced.

Example 14.10: Back to Example 14.10.

[4] Bowker's test is implemented as `statsmodels.stats.contingency_tables.SquareTable.symmetry`.

Bowker's test (cell-wise symmetry). The three off-diagonal pairs above the diagonal are $(n_{AB}, n_{BA}) = (6, 10)$, $(n_{AC}, n_{CA}) = (4, 8)$, $(n_{BC}, n_{CB}) = (5, 7)$.

$$\chi^2_{\mathrm{B}} = \frac{(6-10)^2}{6+10} + \frac{(4-8)^2}{4+8} + \frac{(5-7)^2}{5+7} = 1.00 + 1.33 + 0.33 = 2.67. \tag{14.31}$$

The three pairs contribute df $= 3$, giving a p-value $p \approx 0.45$. Since $p > 0.05$, we fail to reject H_0: no individual confusion pattern shows a statistically significant directional bias, i.e. the classifiers swap each pair of classes in roughly balanced proportions.

Stuart-Maxwell test (marginal homogeneity). The row marginals of Table 14.7 are $(80, 70, 50)$ for C_1 and the column marginals $(88, 68, 44)$ for C_2, giving differences $\mathbf{d} = (-8, +2, +6)$ (summing to zero, as required). Plugging these into `SquareTable.homogeneity` yields $\chi^2_{\mathrm{SM}} \approx 2.0$ with df $= 2$ and $p \approx 0.37$. We fail to reject H_0: the two classifiers predict each class at statistically indistinguishable overall rates.

Both tests agree in this example: the disagreement on 20% of samples (Sec. 14.3.3) is the kind of fluctuation one would expect from a finite test set even if the two classifiers had no underlying systematic difference. Neither a per-pair bias (Bowker) nor an aggregate class-rate bias (Stuart-Maxwell) is detected.

> Bowker rejection signals *behavioural diversity* between two classifiers, which is a *necessary but not sufficient* condition for ensemble benefit. Two models can fail Bowker (lots of asymmetric label substitution) while still being wrong on the same samples, in which case averaging or voting will not help. To decide whether an ensemble will actually improve accuracy, use error correlation (Sec. 14.2.8) or Yule's Q on the correct/incorrect 2×2 collapse (Sec. 14.1.6).

14.3.5. Omnibus Permutation Test (*)

Goal: Obtain a single distribution-free p-value for whether two K-class classifiers behave differently, as the non-parametric counterpart of the parametric Bowker and Stuart-Maxwell tests.

Just as the permutation test is the non-parametric counterpart of McNemar (Sec. 14.1.4), the omnibus permutation test is the non-parametric counterpart of Bowker and Stuart-Maxwell (Sec. 14.3.4).

The test summarizes the asymmetry of the agreement matrix (Sec. 14.3.2) with a single scalar, for example Bowker's itself or the total off-diagonal asymmetry

$$S = \sum_{1 \le j < k \le K} |n_{jk} - n_{kj}|, \tag{14.32}$$

used directly as a discrepancy score rather than referred to a reference distribution. The null distribution is built under exchangeability of the two classifiers: for each disagreeing sample i (where $\hat{y}_{1,i} \neq \hat{y}_{2,i}$), the ordered prediction pair is swapped, $(\hat{y}_{1,i}, \hat{y}_{2,i}) \to (\hat{y}_{2,i}, \hat{y}_{1,i})$, with probability $1/2$. The agreement matrix is rebuilt, the statistic recomputed, and the procedure repeated T times (Sec. 13.3.1) to obtain the null distribution and the $\frac{1}{T+1}$ p-value. Swapping only disagreeing samples leaves the diagonal n_{jj} fixed, mirroring how the permutation form of McNemar reassigns only the discordant pairs.

Example 14.10: Back to Example 14.10 (Table 14.7). The off-diagonal pairs give

$$S = |6 - 10| + |4 - 8| + |5 - 7| = 4 + 4 + 2 = 10. \tag{14.33}$$

Permuting the 40 disagreeing samples $T = 1000$ times yields a null distribution of S whose right tail places the observed $S = 10$ at $p \approx 0.4$. This agrees with Bowker's asymptotic $p \approx 0.45$ (Sec. 14.3.4): the classifiers swap class pairs in roughly balanced proportions. Unlike Bowker's χ^2, the permutation p-value remains exact even if the cell counts had been too small for the asymptotic approximation.

14.3.6. Cohen's Kappa (Multi-Class)

Goal: Quantify the agreement between two K-class classifiers beyond what would be expected by chance.

The definition of Sec. 14.1.5 extends without change; only the ingredients are computed on the $K \times K$ agreement matrix. The observed agreement is the diagonal mass:

$$p_o = \frac{1}{M} \sum_{j=1}^{K} n_{jj}. \tag{14.34}$$

Define the empirical class-prediction frequency of each classifier from the marginals:

$$p_{1,j} = \frac{1}{M} \sum_{k=1}^{K} n_{jk}, \qquad p_{2,j} = \frac{1}{M} \sum_{k=1}^{K} n_{kj}. \tag{14.35}$$

Under independence, the probability that both classifiers issue the same class j is $p_{1,j}p_{2,j}$, so the expected chance agreement is

$$p_e = \sum_{j=1}^{K} p_{1,j}\, p_{2,j}, \tag{14.36}$$

and Cohen's kappa is $\kappa = (p_o - p_e)/(1 - p_e)$.

Range: $\kappa = 1$ indicates perfect agreement, $\kappa = 0$ indicates agreement no better than chance, and $\kappa < 0$ indicates systematic disagreement. The interpretation is identical to the binary case (Sec. 14.1.5).

Example 14.10: Back to Example 14.10. Row marginals of C_1 give $p_{1,j} = (0.40, 0.35, 0.25)$; column marginals of C_2 give $p_{2,j} = (0.44, 0.34, 0.22)$.

$$\begin{aligned} p_o &= 160/200 = 0.80 \\ p_e &= 0.40 \cdot 0.44 + 0.35 \cdot 0.34 + 0.25 \cdot 0.22 = 0.176 + 0.119 + 0.055 = 0.350 \\ \kappa &= \frac{0.80 - 0.350}{1 - 0.350} = \frac{0.450}{0.650} \approx 0.692. \end{aligned} \tag{14.37}$$

The substantial $\kappa \approx 0.69$ indicates strong agreement beyond chance, well above the slight agreement observed in the binary example (Sec. 14.1.5, $\kappa \approx 0.22$).

14.3.7. Multi-Class Brier Score

Goal: Extend the Brier score (Sec. 14.2.2) to predicted class-probability vectors $\mathbf{p}_i \in [0,1]^K$ with $\sum_{k=1}^{K} p_{i,k} = 1$.

With one-hot labels $\mathbf{y}_i \in \{0,1\}^K$ (where $y_{i,k} = 1$ iff sample i belongs to class k), the multi-class Brier score is the mean squared L_2 distance between $\mathbf{p}_i$ and $\mathbf{y}_i$:

$$\mathrm{BS} = \frac{1}{M} \sum_{i=1}^{M} \sum_{k=1}^{K} \left(p_{i,k} - y_{i,k}\right)^2 . \tag{14.38}$$

Range: $0 \le \mathrm{BS} \le 2$. A perfect classifier ($\mathbf{p}_i = \mathbf{y}_i$) achieves $\mathrm{BS} = 0$. A uniform predictor ($p_{i,k} = 1/K$) achieves $\mathrm{BS} = 1 - 1/K$.

When $K = 2$, the formula above is twice the binary Brier score of Sec. 14.2.2 because both classes contribute identical squared errors. Some libraries (and `sklearn.metrics.brier_score_loss`) report the binary form without the factor of 2; verify the convention before comparing values across sources.

Example 14.11: Three samples with $K = 3$ classes and predicted vectors:

i	true class	$\mathbf{p}_i$	$\lVert\mathbf{p}_i - \mathbf{y}_i\rVert_2^2$
1	A	(0.8, 0.1, 0.1)	$0.04 + 0.01 + 0.01 = 0.06$
2	B	(0.3, 0.5, 0.2)	$0.09 + 0.25 + 0.04 = 0.38$
3	C	(0.2, 0.2, 0.6)	$0.04 + 0.04 + 0.16 = 0.24$

$$\mathrm{BS} = \frac{0.06 + 0.38 + 0.24}{3} = \frac{0.68}{3} \approx 0.227. \tag{14.39}$$

The score lies well below the uniform-predictor baseline $1 - 1/K = 2/3 \approx 0.667$, indicating that the classifier's probability vectors are substantially more informative than a coin flip across classes. The per-sample contributions also point to where it is weakest: sample 2 dominates the average (contribution 0.38 out of 0.68), because the predicted vector $(0.3, 0.5, 0.2)$ assigns only 0.5 to the true class B and leaks 0.3 to class A; samples 1 and 3 are tighter (0.06 and 0.24). Inspecting the per-sample term $\lVert\mathbf{p}_i - \mathbf{y}_i\rVert_2^2$ is the standard way to flag the test points where the model is least confident or most miscalibrated.

14.3.8. Multi-Class Calibration: Top-Label and Classwise ECE

Goal: Extend the calibration curve and the Expected Calibration Error of Sec. 14.2.3 to K classes.

The binary ECE bins predictions by their probability and compares the bin mean to the empirical positive rate. With K classes, two standard reductions to the binary machinery are used.

Top-label ECE Each sample is reduced to its winning probability and predicted class,

$$\hat{p}_i = \max_k p_{i,k}, \qquad \hat{y}_i = \operatorname*{argmax}_k p_{i,k}. \tag{14.40}$$

Bin the values $\{\hat{p}_i\}_{i=1}^{M}$ as in Sec. 14.2.3. In each bin b, compare the mean confidence $\bar{\hat{p}}_b$ to the bin accuracy $\bar{a}_b = (1/|I_b|)\sum_{i\in I_b} \mathbb{1}[\hat{y}_i = y_i]$:

$$\mathrm{ECE}_{\mathrm{top}} = \sum_{b=1}^{B} \frac{|I_b|}{M} \left| \bar{a}_b - \bar{\hat{p}}_b \right|. \tag{14.41}$$

This is the metric most commonly reported in the literature (e.g. for deep network calibration).

Classwise (marginal) ECE Treat each class as its own binary calibration problem with $(p_{i,k},\ \mathbb{1}[y_i = k])$ and average:

$$\mathrm{ECE}_{\mathrm{cw}} = \frac{1}{K}\sum_{k=1}^{K} \mathrm{ECE}^{(k)}, \qquad \mathrm{ECE}^{(k)} = \sum_{b=1}^{B} \frac{|I_b^{(k)}|}{M} \left| \bar{y}_b^{(k)} - \bar{p}_b^{(k)} \right|. \tag{14.42}$$

Both reductions produce reliability diagrams of the same shape as Fig. 14.3: one curve per class for classwise, or a single curve in confidence space for top-label.

Top-label ECE is cheap and inspects only the predicted class, so it ignores miscalibration on the non-winning classes. Classwise ECE catches that residual miscalibration but is noisier per class, especially for imbalanced datasets where minority classes populate few bins.

Example 14.12: A probabilistic classifier is evaluated on $M = 6$ samples with $K = 3$ classes (A, B, C); the true labels are two of each class.

i	y_i	$p_{i,A}$	$p_{i,B}$	$p_{i,C}$	$\hat{p}_i = \max_c p_{i,c}$	$\hat{y}_i$
1	A	0.80	0.15	0.05	0.80	A
2	A	0.45	0.40	0.15	0.45	A
3	B	0.10	0.70	0.20	0.70	B
4	B	0.30	0.50	0.20	0.50	B
5	C	0.20	0.25	0.55	0.55	C
6	C	0.35	0.40	0.25	0.40	B

The classifier is correct on samples 1–5 and wrong on sample 6, so accuracy is $5/6 \approx 83\%$. Use $B = 2$ bins, $I_1 = [0, 0.5)$ and $I_2 = [0.5, 1]$.

Top-label ECE. Bin by $\hat{p}_i$:

Bin	samples	$\bar{\hat{p}}_b$ (mean confidence)	$\bar{a}_b$ (accuracy)
$[0, 0.5)$	$2, 6$	$(0.45 + 0.40)/2 = 0.425$	$1/2 = 0.500$
$[0.5, 1]$	$1, 3, 4, 5$	$(0.80 + 0.70 + 0.50 + 0.55)/4 = 0.6375$	$4/4 = 1.000$

$$\mathrm{ECE}_{\mathrm{top}} = \tfrac{2}{6}\,|0.500 - 0.425| + \tfrac{4}{6}\,|1.000 - 0.6375| = 0.025 + 0.2417 \approx 0.267. \tag{14.43}$$

The high-confidence bin dominates: the classifier is underconfident there (predicts 0.64 but is right 100% of the time).

Classwise ECE. For each class, treat $(p_{i,c},\ \mathbb{1}[y_i = c])$ as a binary calibration problem.

Class c	Bin	samples in bin	$\bar{p}_b^{(c)}$	$\bar{y}_b^{(c)}$	$\lvert\bar{y}_b^{(c)} - \bar{p}_b^{(c)}\rvert$
A	[0, 0.5)	$2, 3, 4, 5, 6$	0.280	0.200	0.080
A	[0.5, 1]	1	0.800	1.000	0.200
B	[0, 0.5)	$1, 2, 5, 6$	0.300	0.000	0.300
B	[0.5, 1]	$3, 4$	0.600	1.000	0.400
C	[0, 0.5)	$1, 2, 3, 4, 6$	0.170	0.200	0.030
C	[0.5, 1]	5	0.550	1.000	0.450

The per-class ECEs are

$$\mathrm{ECE}^{(A)} = \tfrac{5}{6}(0.080) + \tfrac{1}{6}(0.200) = 0.100, \quad \mathrm{ECE}^{(B)} = \tfrac{4}{6}(0.300) + \tfrac{2}{6}(0.400) \approx 0.333, \tag{14.44}$$

$$\mathrm{ECE}^{(C)} = \tfrac{5}{6}(0.030) + \tfrac{1}{6}(0.450) = 0.100, \tag{14.45}$$

and the aggregate is

$$\mathrm{ECE}_{\mathrm{cw}} = \tfrac{1}{3}(0.100 + 0.333 + 0.100) \approx 0.178. \tag{14.46}$$

Comparison. The top-label score (0.267) flags overall underconfidence in the high-probability bin but says nothing about *which* class is responsible. The classwise score (0.178) is smaller but localizes the problem: class B contributes 0.333, more than three times the per-class contribution of A or C, because the model assigns moderate probability mass to B on samples where B is absent (bin 1) and underestimates B on samples where it is correct (bin 2). For deployment, the actionable recommendation is to recalibrate class B specifically rather than the whole classifier.

14.3.9. Multi-Class AUC and DeLong

Goal: Extend AUC-based discrimination comparison (Sec. 14.2.7) to K classes (statistically significant AUC difference).

With K classes, the ROC curve is not unique. Two standard aggregations summarize discrimination in a single scalar.

One-vs-Rest macro-AUC. For each class k compute the binary AUC of $\{(p_{i,k}, \mathbb{1}[y_i = k])\}_i$, then average:

$$\mathrm{AUC}_{\mathrm{OvR}} = \frac{1}{K}\sum_{k=1}^{K} \mathrm{AUC}^{(k)}. \tag{14.47}$$

One-vs-One AUC. For every ordered pair of classes (j, k) with $j \neq k$, compute the AUC restricted to samples whose true label is j or k, using $p_{i,k}$ as the score, and average over the $\binom{K}{2}$ unordered pairs (Hand and Till's M-measure).

DeLong for multi-class. DeLong's test (Sec. 14.2.7) gives a paired variance estimate for the difference between two AUCs. In the multi-class setting it is applied per class on the OvR scores, yielding K p-values that can be combined (e.g. Bonferroni correction across classes), or on $\mathrm{AUC}_{\mathrm{OvR}}$ directly using the per-class DeLong covariances aggregated into the macro variance. Standard implementations (`sklearn.metrics.roc_auc_score` with `multi_class='ovr'` or `'ovo'`) compute the point estimates; dedicated libraries[5] provide the DeLong variance.

Macro-AUC weights classes equally, regardless of frequency, so a poorly discriminated minority class can dominate the score. OvO weights class *pairs* equally and is less sensitive to imbalance, but more expensive. The two can rank a pair of classifiers differently; report the one matching the operational cost structure.

14.3.10. Per-Sample-Score Methods (Carry Over Unchanged)

Goal: Explain why five of the binary comparison methods (paired score difference, paired t-test, Wilcoxon, Pearson error correlation, Spearman) need no rework for $K > 2$.

A comparison method is K-aware only if its formula directly indexes a class label or a class-conditional probability. Methods that reduce each sample to a single scalar $S_{k,i}$ before doing anything else are K-agnostic: once you have a column of scalars per classifier, the test code is the same whether the scalars came from a binary or a multi-class problem.

The scalar score. For probabilistic classifiers the natural per-sample score is either the multi-class Brier (Sec. 14.3.7) or the cross-entropy loss (Sec. 8.4):

$$S_{k,i} = \sum_{c=1}^{K} \left(p_{k,i,c} - y_{i,c}\right)^2 \qquad \text{or} \qquad S_{k,i} = -\sum_{c=1}^{K} y_{i,c} \log p_{k,i,c} \tag{14.48}$$

where $y_{i,c} \in \{0, 1\}$ is the one-hot label and $p_{k,i,c}$ is classifier k's probability for class c on sample i. Both reduce to their binary forms when $K = 2$.

What each method answers and what it sees.

- Paired score difference (Sec. 14.2.4). *Goal:* quantify which classifier produces better-calibrated probabilities, sample by sample. *Input:* the per-sample gap $\delta_i = S_{1,i} - S_{2,i}$; the sign of $\bar{\delta}$ picks the winner.
- Paired t-test (Sec. 14.2.5). *Goal:* test whether the score difference is statistically significant, assuming approximately normal δ_i. *Input:* the vector $\{\delta_i\}_{i=1}^{M}$; tests $\mathbb{E}[\delta_i] = 0$.
- Wilcoxon signed-rank (Sec. 14.2.6). *Goal:* non-parametric significance test on the median score difference, robust to outliers and non-normality. *Input:* ranks of $|\delta_i|$ with their signs.
- Pearson error correlation (Sec. 14.2.8). *Goal:* measure how correlated the two classifiers' errors are (probabilistic analog of Yule's Q); low correlation signals ensemble potential. *Input:* the two columns $\{S_{1,i}\}, \{S_{2,i}\}$; returns $r \in [-1, 1]$.

[5] E.g. `pROC` in R.

- Spearman rank correlation (Sec. 14.2.9). *Goal:* non-parametric error-correlation diagnostic; test whether the two classifiers rank samples by difficulty in the same order. *Input:* ranks of those columns; returns $r_s \in [-1, 1]$.

None of these formulas index a class. The class count K is absorbed into S at the score-construction step and is invisible from then on.

What is gained and what is lost. The same library calls (`scipy.stats.ttest_rel`, `wilcoxon`, `pearsonr`, `spearmanr`) work for any classification task; only the score function changes. The price is that these tests answer only "do the classifiers' overall score distributions differ?" and not "*on which class* does one classifier do worse?" For per-class diagnosis use the classwise ECE (Sec. 14.3.8) or per-class DeLong (Sec. 14.3.9), both of which loop over K explicitly.

Rule of thumb: if a comparison method sees the classifier output only through a per-sample loss $S_{k,i} \in \mathbb{R}$, it is K-agnostic and needs no multi-class version. If it indexes class probabilities $p_{k,i,c}$ or class pairs (j, k), it is K-aware and requires the multi-class treatment given earlier in this section.

Example 14.13: Two probabilistic classifiers are evaluated on $M = 6$ samples with $K = 3$ classes (A, B, C). The per-sample score is the multi-class Brier $S_{k,i} = \sum_{c=1}^{K}(p_{k,i,c} - y_{i,c})^2$.

i	y_i	$\mathbf{p}_{1,i}$	$\mathbf{p}_{2,i}$	$S_{1,i}$	$S_{2,i}$	δ_i
1	A	(0.80, 0.15, 0.05)	(0.60, 0.30, 0.10)	0.065	0.260	−0.195
2	A	(0.45, 0.40, 0.15)	(0.55, 0.30, 0.15)	0.485	0.315	+0.170
3	B	(0.10, 0.70, 0.20)	(0.20, 0.55, 0.25)	0.140	0.305	−0.165
4	B	(0.30, 0.50, 0.20)	(0.25, 0.60, 0.15)	0.380	0.245	+0.135
5	C	(0.20, 0.25, 0.55)	(0.30, 0.20, 0.50)	0.305	0.380	−0.075
6	C	(0.35, 0.40, 0.25)	(0.20, 0.35, 0.45)	0.845	0.465	+0.380
means				$\bar{S}_1 = 0.370$	$\bar{S}_2 = 0.328$	$\bar{\delta} = +0.042$

The mean gap $\bar{\delta} \approx +0.042$ is small and positive, slightly favouring classifier 2 (smaller Brier is better).

Paired t-test (Sec. 14.2.5). With $s_\delta \approx 0.225$ and $M = 6$,

$$t = \frac{0.042}{0.225/\sqrt{6}} \approx 0.46, \qquad p \approx 0.67. \tag{14.49}$$

Fail to reject; the score difference is not significant.

Pearson error correlation (Sec. 14.2.8). Computing on the two score columns,

$$r = \frac{\sum_i (S_{1,i} - \bar{S}_1)(S_{2,i} - \bar{S}_2)}{\sqrt{\sum_i (S_{1,i} - \bar{S}_1)^2}\,\sqrt{\sum_i (S_{2,i} - \bar{S}_2)^2}} \approx 0.75. \tag{14.50}$$

Strong positive error correlation: when one classifier scores poorly on a sample, the other does too. By the Step 2 threshold of the Ensemble Recommendation (Sec. 14.3.11), $r \approx 0.75$ is approaching the $r \geq 0.8$ redundancy zone, so ensembling would likely produce only marginal gains.

Observation. The procedure is byte-for-byte identical to the binary Example 14.9: only the per-sample score function changes from binary Brier $(p_i - y_i)^2$ to multi-class Brier $\sum_c (p_{i,c} - y_{i,c})^2$. The downstream library calls (`scipy.stats.ttest_rel`, `pearsonr`) do not see K.

14.3.11. Ensemble Recommendation

Goal: Decide whether the two compared classifiers should be combined into an ensemble (Sec. 15.3), and if so, whether to use a symmetric (majority vote, score averaging) or asymmetric (weighted, class-conditional) fusion rule. The decision rests on the diagnostics built up in this chapter.

The diagnostics produce four checkpoints. The recommendation is conservative: an ensemble is justified only when all the following conditions hold. Otherwise, deploy the better single classifier.

Both classifiers are individually useful Verify that each classifier outperforms the trivial baseline (constant prediction at the majority class). Then run McNemar (Sec. 14.1.4, applied to the correct/incorrect 2×2 for $K > 2$). If the test rejects *and* the accuracy gap is large (rule of thumb: > 5 percentage points), drop the weaker classifier; ensembling rarely recovers from a wide accuracy gap because the better model's votes are diluted by the weaker one.

The errors are not redundant This is the critical condition: ensembling helps only when the two classifiers fail on *different* samples.

- Hard predictions: Yule's Q on the correct/incorrect 2×2 (Sec. 14.1.6; for $K > 2$, apply to the collapse). $Q \geq 0.8$ signals highly correlated errors, so ensembling will produce marginal gains at best.
- Probabilistic outputs: Pearson error correlation r (Sec. 14.2.8) or Spearman r_s (Sec. 14.2.9). $r \geq 0.8$ is the same red flag; $r \leq 0$ is the ideal case (negatively correlated errors).

If both diagnostics indicate near-perfect correlation, stop: the two classifiers are effectively the same model and an ensemble will not help.

Choose symmetric vs. asymmetric fusion With diversity confirmed, the choice of fusion rule depends on the *shape* of disagreement:

- Bowker (Sec. 14.3.4) fails to reject: disagreements are symmetric across class pairs. Use *symmetric* fusion (majority vote, soft vote / score averaging, Sec. 15.3).
- Bowker rejects: at least one class pair is asymmetrically confused. Use *asymmetric* fusion: weighted averaging (weight by per-class accuracy) or class-conditional routing (send disputed-pair samples to the classifier known to be stronger on that boundary).

Calibration before soft averaging (optional) Soft vote and probability averaging assume the inputs are comparable. Before averaging probabilities, check the calibration of both classifiers (Sec. 14.2.3 for binary, Sec. 14.3.8 for multi-class). If either is poorly calibrated, recalibrate first (Platt scaling, isotonic regression) or fall back to hard voting, which is unaffected by calibration.

Cohen's κ measures agreement on *labels*, not on *errors*. Two classifiers with high κ (they label most samples the same way) can still benefit from ensembling if their few disagreements happen to be precisely on the samples each one gets wrong, so κ is not the ensemble-benefit signal. The signal is Yule's Q or the error correlation, both of which condition on the truth y.

Example 14.14: Apply the recipe to the binary worked example of Sec. 14.1.1 ($n_{11} = 150$, $n_{10} = 25$, $n_{01} = 15$, $n_{00} = 10$).

- McNemar's p-value ≈ 0.11 (Sec. 14.1.4), so the 5-point accuracy gap is not significant; both classifiers stay in play.
- $Q \approx 0.6$ (Sec. 14.1.6), well below the 0.8 redundancy threshold, so diversity is sufficient.
- Bowker reduces to McNemar at $K = 2$ and again fails to reject, so disagreements are symmetric: use majority vote or soft averaging.
- with $K = 2$ and assuming the classifiers are reasonably calibrated, soft averaging is the default; otherwise, fall back to majority vote.

15. Classifier Fusion

Goal: Combine information from multiple data sources or feature sets or classifiers to improve classification performance.

Contents

15.1. Data Concatenation **203**
15.2. Feature Concatenation **204**
15.3. Ensemble of Classifiers **204**
15.3.1. Preface 204
15.3.2. Majority Vote 204
15.3.3. Soft Vote 205
15.3.4. Linearly Weighted Combining 206
15.3.5. Log-Likelihood Ratio (LLR) 206
15.3.6. Learned Combiner (Stacking) 207
15.3.7. General Non-Linear Combining 208
15.3.8. Held-out average log-likelihood gain (*) 208
15.3.9. Summary 208
15.4. Multi-Class Ensemble **209**
15.4.1. Plurality Vote 210
15.4.2. Soft Vote 210
15.4.3. Linearly Weighted Combining 211
15.4.4. Log-Probability Fusion 211
15.4.5. Learned Combiner (Multinomial Stacking) 212
15.4.6. Hard versus Soft Outputs 212
15.5. Sequential Combining **213**
15.5.1. Sequential Log-Odds Update 213
15.5.2. Multiple Independent Sensors 214
15.5.3. Connection to Bayesian Learning (*) 214
15.6. Multiple Instance Learning (MIL) **215**

When multiple data sources are available, there are several strategies for incorporating them into a classification pipeline. Figure 15.1 illustrates five common approaches.

- *Single source* (a): a single dataset is processed through one feature-extraction and classification pipeline.

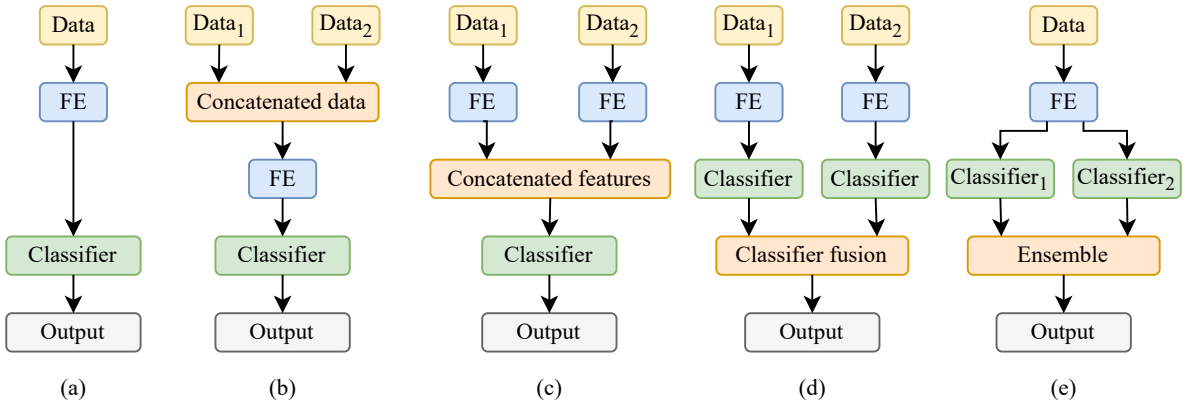


Figure 15.1.: Data-combining strategies: (a) single source, (b) data concatenation (early fusion), (c) feature concatenation (early fusion), (d) classifier fusion (late fusion), (e) classifiers ensemble (late fusion).

- *Data concatenation* (b): raw data from multiple sources are concatenated before feature extraction, producing a single feature vector per sample.
- *Feature concatenation* (c): each data source undergoes independent feature extraction; the resulting feature vectors are concatenated before classification.
- *Classifier fusion* (d): each data source is processed by an independent feature-extraction and classification pipeline; the individual predictions are combined by a fusion rule (Sec. 15.3).
- *Classifiers ensemble* (e): a single data source undergoes feature extraction, then multiple different classifiers are applied in parallel; their predictions are combined by an ensemble rule (e.g., majority voting or averaging) (Sec. 15.3).

Approaches (b) and (c) are collectively termed *early fusion* because they merge information before the classifier; approaches (d) and (e) are *late fusion* because independent classifiers produce separate outputs that are combined afterward.

15.1. Data Concatenation

Goal: Combine (multimodal) raw data from multiple sources before feature extraction.

Given K data sources producing raw vectors $\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_K$ for the same sample, form the concatenated input:

$$\mathbf{d} = [\mathbf{d}_1; \mathbf{d}_2; \ldots; \mathbf{d}_K] \tag{15.1}$$

A single feature-extraction pipeline then maps $\mathbf{d}$ to a feature vector $\mathbf{x}$, which is passed to the classifier (Fig. 15.1b).

- When combining multimodal data (e.g., EEG and EMG, or images from different scanners), each source may have its own acquisition characteristics, noise profile, and feature distribution.
- Before concatenation verify that the sources share a compatible representation (e.g., same sampling rate, resolution, or coordinate system).
- The concatenated vector may be very high-dimensional, increasing the risk of overfitting when M is small relative to N.

15.2. Feature Concatenation

Goal: Combine independently extracted features from multiple sources into a single feature vector.

Each source k is processed by its own feature-extraction pipeline, yielding $\mathbf{x}_k \in \mathbb{R}^{N_k}$. The concatenated feature vector is

$$\mathbf{x} = [\mathbf{x}_1; \mathbf{x}_2; \ldots; \mathbf{x}_K] \in \mathbb{R}^N, \quad N = \sum_{k=1}^{K} N_k \tag{15.2}$$

A single classifier operates on $\mathbf{x}$ (Fig. 15.1c).

Feature concatenation allows each source to use a specialized extraction pipeline suited to its modality. However, the concatenated dimension N grows with K, increasing the risk of overfitting when M is small relative to N.

15.3. Ensemble of Classifiers

Goal: Combine predictions from multiple *binary* classifiers to improve overall performance.

15.3.1. Preface

Given K classifiers, each producing a prediction for the same input $\mathbf{x}$, fusion methods aggregate these predictions into a single output. The methods in this section apply to two settings (Fig. 15.1d,e):

- *Classifiers ensemble*: K different classifiers operate on the same data source. The ensemble benefits from diversity among classifiers; if they make similar errors, combining them provides little gain. Classifier comparison methods (Sec. 14.1 and 14.2.8) can quantify this diversity by measuring:
  - *Yule's Q* (Sec. 14.1.6): measures the association between binary correct/incorrect outcomes. $Q \approx 0$ indicates independent errors, while $Q < 0$ suggests complementary performance.
  - *Error correlation* (Sec. 14.2.8 and 14.2.9): the Pearson/Spearman correlation between per-sample probabilistic scores. $r < 0$ and $r_s < 0$ indicate that the classifiers' errors are negatively correlated, which indicates a high ensemble potential.
- *Classifier fusion*: a single classifier is applied to K different data sources (e.g., multimodal sensors). When the sources originate from different domains, a domain consistency check (Sec. 13.2.2) should verify distributional compatibility before fusion. Classifier fusion methods are a subset of the ensemble methods; applicability is summarized in Sec. 15.3.9.

15.3.2. Majority Vote

Goal: Combine predictions by selecting the most common class.

Condorcet's jury theorem

Condorcet's jury theorem (1785) is a political science theorem about the relative probability of a given group of individuals arriving at a correct decision. The theorem suggests a decision-making scenario with two options, where one option is assumed to be correct. Given a group of n voters making independent choices with $p > \frac{1}{2}$ of making the correct choice, the theorem states that the majority vote is most likely to select the correct option. Moreover, for $n \to \infty$ the probability of the group reaching the correct decision by majority vote approaches 1.

Definition

Each classifier k produces a binary prediction $\hat{y}_k \in \{0, 1\}$. The fused prediction is

$$\hat{y}_{\text{fused}} = \begin{cases} 1 & \sum_{k=1}^{K} \hat{y}_k > K/2 \\ 0 & \text{otherwise} \end{cases} \tag{15.3}$$

Choosing odd K avoids ties.

15.3.3. Soft Vote

Goal: Combine probabilistic classifier outputs by averaging their predicted probabilities.

Each probabilistic classifier k outputs $p_k = f_k(\mathbf{x}) = \Pr(\hat{y} = 1 \mid \mathbf{x})$. The soft vote averages these probabilities:

$$p_{\text{soft}} = \frac{1}{K} \sum_{k=1}^{K} p_k \tag{15.4}$$

The fused decision is

$$\hat{y}_{\text{fused}} = \begin{cases} 1 & p_{\text{soft}} \geq 0.5 \\ 0 & p_{\text{soft}} < 0.5 \end{cases} \tag{15.5}$$

Example 15.1: Three classifiers produce probabilities $p_1 = 0.9$, $p_2 = 0.4$, $p_3 = 0.45$ for a given sample.

- *Majority vote*: $\hat{y}_1 = 1$, $\hat{y}_2 = 0$, $\hat{y}_3 = 0$. Fused prediction: $\hat{y}_{\text{fused}} = 0$ (two out of three vote for class 0).
- *Soft vote*: $p_{\text{soft}} = (0.9+0.4+0.45)/3 = 0.583$. Fused prediction: $\hat{y}_{\text{fused}} = 1$ (average probability exceeds 0.5).

The high confidence of classifier 1 tips the soft vote toward class 1, whereas majority vote ignores this confidence.

For the special case of $K = 2$, if the classifiers disagree ($\hat{y}_1 \neq \hat{y}_2$), the fused prediction follows the more confident classifier:

$$\hat{y}_{\text{fused}} = \begin{cases} \hat{y}_1 & |p_1 - 0.5| > |p_2 - 0.5| \\ \hat{y}_2 & |p_2 - 0.5| > |p_1 - 0.5| \end{cases} \tag{15.6}$$

If one classifier is very confident (e.g., $p_k = 0.95$) while the others are weakly opposed (e.g., $p_j = 0.45$), the confident vote contributes more to the average than a single hard vote would.

15.3.4. Linearly Weighted Combining

Goal: Combine probabilistic classifier outputs using a weighted average.

Each probabilistic classifier k outputs $p_k = f_k(\mathbf{x})$. Assign non-negative weights $w_k \geq 0$ with $\sum_{k=1}^{K} w_k = 1$. The fused probability is

$$p_{\text{fused}} = \sum_{k=1}^{K} w_k \, p_k \tag{15.7}$$

The fused decision is

$$\hat{y}_{\text{fused}} = \begin{cases} 1 & p_{\text{fused}} \geq 0.5 \\ 0 & p_{\text{fused}} < 0.5 \end{cases} \tag{15.8}$$

A common approach is to normalize validation accuracies [17]:

$$w_k = \frac{a_k}{\sum_{j=1}^{K} a_j} \tag{15.9}$$

where a_k is the validation accuracy (or other performance metric, e.g., F_1) of classifier k. Equal weights $w_k = 1/K$ recover simple averaging.

> Alternatively, weights can be learned by minimizing the Brier score or cross-entropy loss on a validation set, treating $\{w_k\}$ as optimization variables subject to $w_k \geq 0$ and $\sum_k w_k = 1$. However, this approach involves a significant computation complexity.

15.3.5. Log-Likelihood Ratio (LLR)

Goal: Combine probabilistic classifier outputs using log-odds (Sec. 8.6).

Each probabilistic classifier k outputs $p_k = f_k(\mathbf{x}) = \Pr\big(\hat{y} = 1 \mid \mathbf{x}\big)$. Recall that the logit (log-odds) is given by

$$\text{logit}_k = \ln \frac{p_k}{1 - p_k} \tag{15.10}$$

Under the assumption that the K classifiers (or classifications) are conditionally independent given the true label, the fused log-odds is the sum of individual logits:

$$\text{logit}_{\text{fused}} = \sum_{k=1}^{K} \text{logit}_k = \sum_{k=1}^{K} \ln \frac{p_k}{1 - p_k} = \ln \prod_{k=1}^{K} \frac{p_k}{1 - p_k} \tag{15.11}$$

The fused probability is recovered by applying the sigmoid: $p_{\text{fused}} = \sigma(\text{logit}_{\text{fused}})$.

When the classes are imbalanced, the log-prior odds should not be dropped. The general fusion formula is

$$\text{logit}_{\text{fused}} = \ln \frac{M_1}{M_0} + \sum_{k=1}^{K} \text{logit}_k \tag{15.12}$$

where M_0 and M_1 are the class counts. The log-prior term acts as a constant bias that shifts the decision boundary toward the minority class. When the classes are balanced ($M_0 = M_1$), the bias vanishes and the standard formula is recovered.

The fused decision is

$$\hat{y}_{\text{fused}} = \begin{cases} 1 & \text{logit}_{\text{fused}} \geq 0 \quad (p_{\text{fused}} \geq 0.5) \\ 0 & \text{logit}_{\text{fused}} < 0 \end{cases} \tag{15.13}$$

With imbalanced priors, the log-prior bias in $\text{logit}_{\text{fused}}$ effectively shifts the decision boundary away from 0.5 toward the minority class.

LLR fusion requires probabilistic classifiers and assumes independence between classifier errors. When the independence assumption is violated, the fused score may be overconfident.

Proof. Why Conditional Independence Leads to Summing Log-Odds? By Bayes' rule in odds form, the posterior odds given evidence $\mathbf{x}_k$ from classifier k are

$$\frac{\Pr(y{=}1 \mid \mathbf{x}_k)}{\Pr(y{=}0 \mid \mathbf{x}_k)} = \frac{\Pr(\mathbf{x}_k \mid y{=}1)}{\Pr(\mathbf{x}_k \mid y{=}0)} \cdot \frac{\Pr(y{=}1)}{\Pr(y{=}0)} \tag{15.14}$$

where $\frac{\Pr(\mathbf{x}_k|y=1)}{\Pr(\mathbf{x}_k|y=0)}$ is likelihood ratio$_k$. When the K classifiers are conditionally independent given the true label, the joint likelihood ratio factorizes:

$$\frac{\Pr(\mathbf{x}_1, \ldots, \mathbf{x}_K \mid y{=}1)}{\Pr(\mathbf{x}_1, \ldots, \mathbf{x}_K \mid y{=}0)} = \prod_{k=1}^{K} \frac{\Pr(\mathbf{x}_k \mid y{=}1)}{\Pr(\mathbf{x}_k \mid y{=}0)} = \prod_{k=1}^{K} \text{likelihood ratio}_k \tag{15.15}$$

Multiplying by the prior odds and taking the logarithm converts the product into a sum. Absorbing the log-prior into a constant (or assuming equal priors), the fused log-odds reduces to the sum of individual LLRs.

15.3.6. Learned Combiner (Stacking)

Goal: Learn optimal combination weights from data, rather than assuming equal weights or independence.

The previous fusion methods use fixed rules. When the classifiers' outputs are probabilistic, a stronger ensemble can be obtained by fitting a logistic regression model on the classifiers' log-odds:

$$\text{logit}(p_{\text{fused}}) = \beta_0 + \sum_{k=1}^{K} \beta_k \, \text{logit}_k \tag{15.16}$$

where β_0 is a bias term and β_k are learned coefficients. The parameters $\{\beta_0, \beta_1, \ldots, \beta_K\}$ are fit on a validation set, and the stacked model is evaluated on a separate test set to avoid overfitting.

The fused probability is recovered via the sigmoid: $p_{\text{fused}} = \sigma\big(\beta_0 + \sum_k \beta_k \, \text{logit}_k\big)$.

When $\beta_0 = 0$ and $\beta_k = 1$ for all k, the learned combiner reduces to LLR fusion. The learned coefficients allow the model to down-weight redundant or poorly calibrated classifiers.

15.3.7. General Non-Linear Combining

Goal: Use a non-linear model to combine base classifier predictions, capturing complex interactions between them.

The stacking approach can be generalized by replacing the linear logistic regression model with any non-linear classifier (e.g., a neural network, random forest, or gradient boosting machine). The fused prediction is formulated as:

$$p_{\text{fused}} = g_{\boldsymbol{\phi}}(p_1, \ldots, p_K) \tag{15.17}$$

where $g_{\boldsymbol{\phi}}$ is the non-linear combiner parameterized by $\boldsymbol{\phi}$. As with the linear learned combiner, the parameters $\boldsymbol{\phi}$ are fit on a validation set, and the stacked model is evaluated on a separate test set.

> While non-linear combiners can capture complex dependencies between base classifiers, they are more prone to overfitting than linear combiners, especially if the validation set is small.

15.3.8. Held-out average log-likelihood gain (*)

The ensemble gain can be quantified using the log-likelihood. Recall (Sec. 8.4) that the per-sample log-likelihood for a probabilistic classifier with output $p_i = f_{\mathbf{w}}(\mathbf{x}_i)$ is

$$\log p(y_i \mid \mathbf{x}_i) = y_i \log p_i + (1 - y_i) \log(1 - p_i) \tag{15.18}$$

The held-out average log-likelihood gain of the ensemble over the best base model is

$$\Delta\bar{\ell} = \frac{1}{M_{\text{test}}} \sum_{i=1}^{M_{\text{test}}} \left[\log p_{\text{fused}}(y_i \mid \mathbf{x}_i) - \log p_{\text{best}}(y_i \mid \mathbf{x}_i)\right] \tag{15.19}$$

where p_{fused} and p_{best} are the predicted probabilities of the stacked model and the best individual classifier, respectively, evaluated on the test set.

Positive $\Delta\bar{\ell}$ indicates that the ensemble provides better calibrated probabilities than the best base model. Equivalently, $\Delta\bar{\ell}$ equals the reduction in BCE loss: $\Delta\bar{\ell} = \mathcal{L}_{\text{best}} - \mathcal{L}_{\text{fused}}$.

15.3.9. Summary

The methods above apply to both *classifiers ensemble* of K classifiers and *classifier fusion* of interference from K different data sources (or noisy measurements $\mathbf{x}_1, \ldots, \mathbf{x}_K$ of the same quantity). In the classifier-fusion setting, the independence assumption required for LLR fusion corresponds to independent measurement noise, and methods that assign different learned parameters per classifier (Linearly Weighted Combining, Learned Combiner) are not applicable, since there is only one classifier. The method summary is presented in Table 15.1.

Cross-validation of ensembles When evaluating an ensemble with cross-validation, the procedure depends on whether the fusion rule has learned parameters:

- *Fixed-rule methods* (Majority vote, Soft vote, LLR fusion): no parameters are learned from data. A standard outer CV loop suffices. In each fold, train the K base classifiers on the training partition, apply the fixed fusion rule, and evaluate on the held-out test partition.
- *Learned-parameter methods* (Weighted combining, Learned combiner): the combiner weights (w_k or β_k) must be fitted on data that is separate from both the base-classifier training data and the test data. Within each outer CV fold, further split the training partition into a *base-training* set (to train the K classifiers) and a *validation* set (to fit the combiner weights). The fused model is then evaluated on the outer test fold (Sec. 4.5).

Possible data leakage

Fitting combiner weights on the same data used to train the base classifiers is a form of data leakage (Sec. 13.2.1): the base classifiers' predictions on their own training samples are overconfident, leading to inflated ensemble performance estimates.

Method	Input type	Key idea	Classifier fusion	Validation set
Majority vote	Hard ($\hat{y}_k \in \{0, 1\}$)	Most common class	Yes	No
Soft vote	Probabilistic (p_k)	Average of probabilities	Yes	No
Weighted combining	Probabilistic (p_k)	Weighted average of probabilities	No	Yes
LLR fusion	Probabilistic (p_k)	Sum of log-odds (assumes independence)	Yes	No
Learned combiner	Probabilistic (p_k)	Logistic regression on log-odds	No	Yes
Non-linear combiner	Any	Non-linear model on outputs	No	Yes

Table 15.1.: Summary of classifier fusion and ensemble methods. All the methods are applicable for classifiers ensemble. "Classifier fusion" indicates applicability when the same classifier is applied to K independent data sources. "Validation set" indicates whether the method requires a held-out validation set to fit combiner parameters.

15.4. Multi-Class Ensemble

Goal: Extend the binary fusion rules of Sec. 15.3 to C-class classification.

Consider K base classifiers and $C \geq 2$ classes. Each classifier k outputs either a hard label $\hat{y}_k \in \{1, \ldots, C\}$ or a probability vector $\mathbf{p}_k = (p_{k,1}, \ldots, p_{k,C})$ with $p_{k,c} \geq 0$ and $\sum_{c=1}^{C} p_{k,c} = 1$.

The fused output is either a single label $\hat{y}_{\text{fused}}$ or a probability vector $\mathbf{p}_{\text{fused}}$. The methods below are direct lifts of their binary counterparts; Table 15.2 summarizes them upfront.

Method	Input	Fused output	Decision rule	Validation set
Plurality vote	Hard $\hat{y}_k$	Vote counts per class	arg max counts	No
Soft vote	Probabilistic $\mathbf{p}_k$	Mean probability vector	$\arg\max_c \bar{p}_c$	No
Weighted combining	Probabilistic $\mathbf{p}_k$	Weighted mean vector	$\arg\max_c \sum_k w_k p_{k,c}$	Yes
Log-probability fusion	Probabilistic $\mathbf{p}_k$	Softmax of summed log-probabilities	$\arg\max_c \sum_k \log p_{k,c}$	No
Learned combiner	Probabilistic $\mathbf{p}_k$	Multinomial logistic on log-probabilities	$\arg\max_c$ of softmax output	Yes
Non-linear combiner	Any	Any model with softmax head	$\arg\max_c$ of model output	Yes

Table 15.2.: Multi-class ensemble methods. Each row is the C-class analogue of the corresponding binary rule in Table 15.1.

15.4.1. Plurality Vote

Goal: Multi-class analogue of majority vote: select the class with the largest number of votes.

Each classifier k produces a hard label $\hat{y}_k \in \{1, \ldots, C\}$. The fused prediction is

$$\hat{y}_{\text{fused}} = \arg \max_{c \in \{1,\ldots,C\}} \sum_{k=1}^{K} \mathbb{1}[\hat{y}_k = c] \tag{15.20}$$

Condorcet's theorem extends to multi-class with the threshold $p > 1/C$: if each classifier is correct with probability $p > 1/C$ and errors are independent, the plurality vote converges to the correct class as $K \to \infty$.

Ties are more frequent than in the binary case, especially when K is small relative to C. Common tie-breakers: (i) weight votes by per-classifier validation accuracy, (ii) fall back to soft vote (Sec. 15.4.2), or (iii) prefer the class with the smaller index (deterministic but arbitrary).

15.4.2. Soft Vote

Goal: Average the probability vectors of probabilistic classifiers.

Each classifier k outputs $\mathbf{p}_k$. The fused probability vector is the elementwise mean:

$$\mathbf{p}_{\text{soft}} = \frac{1}{K}\sum_{k=1}^{K}\mathbf{p}_k, \qquad \hat{y}_{\text{fused}} = \arg\max_c p_{\text{soft},c} \tag{15.21}$$

Soft vote averages probability scales across classifiers; if any base classifier is poorly calibrated, its overconfident scores dominate the average. Verify per-class calibration (Sec. 14.3.8) before averaging.

15.4.3. Linearly Weighted Combining

Goal: Weight each classifier's probability vector by its validation performance.

Assign non-negative weights w_k with $\sum_k w_k = 1$. The fused probability vector and decision are

$$\mathbf{p}_{\text{fused}} = \sum_{k=1}^{K} w_k\,\mathbf{p}_k, \qquad \hat{y}_{\text{fused}} = \arg\max_c\ p_{\text{fused},c} \tag{15.22}$$

A common choice is $w_k = a_k / \sum_j a_j$ where a_k is the validation accuracy (or macro-F_1) of classifier k.

When Bowker's test (Sec. 14.3.4) rejects symmetry, the per-class error rates differ across classifiers. In that case, class-specific weights $w_{k,c}$ (re-normalized so $\sum_c p_{\text{fused},c} = 1$) can route disputed class pairs to the classifier known to be stronger on that boundary. See Sec. 14.3.11, Step 3.

15.4.4. Log-Probability Fusion

Goal: Multi-class analogue of LLR fusion (Sec. 15.3.5): sum log-probabilities under conditional independence.

Under the assumption that the K classifiers are conditionally independent given the true class, the joint log-posterior factorizes as

$$\log \Pr\big(y = c \mid \mathbf{x}_1, \ldots, \mathbf{x}_K\big) \ \propto\ \log \pi_c + \sum_{k=1}^{K} \log p_{k,c} \tag{15.23}$$

where $\pi_c = \Pr(y = c)$ is the class prior. Normalizing by the softmax across classes yields

$$p_{\text{fused},c} = \frac{\pi_c \prod_{k=1}^{K} p_{k,c}}{\sum_{c'=1}^{C} \pi_{c'} \prod_{k=1}^{K} p_{k,c'}} \tag{15.24}$$

The fused decision is

$$\hat{y}_{\text{fused}} = \arg\max_c \left[\log \pi_c + \sum_{k=1}^{K} \log p_{k,c}\right] \tag{15.25}$$

When classes are balanced ($\pi_c = 1/C$), the prior term is a constant that drops out of the arg max. The binary case (Sec. 15.3.5) is recovered by taking the log-ratio against class 0.

If any classifier assigns $p_{k,c} \approx 0$ to a class, $\log p_{k,c} \to -\infty$ and that class is permanently excluded by a single overconfident vote. Floor probabilities at ε (e.g., $\varepsilon = 10^{-6}$): $p_{k,c} \leftarrow \max(p_{k,c}, \varepsilon)$. This is the multi-class form of the overconfidence warning for binary LLR fusion.

15.4.5. Learned Combiner (Multinomial Stacking)

Goal: Fit a multinomial logistic regression on the base classifiers' log-probabilities.

The fused log-posterior is

$$\log \Pr(y = c \mid \cdot) \;\propto\; \beta_{0,c} + \sum_{k=1}^{K} \sum_{c'=1}^{C} \beta_{k,c,c'} \log p_{k,c'} \tag{15.26}$$

followed by a softmax across c. Parameters $\{\beta_{0,c}, \beta_{k,c,c'}\}$ are fitted on a held-out validation set and the stacked model is evaluated on a separate test set.

The log-probability fusion of Sec. 15.4.4 is the special case $\beta_{0,c} = \log \pi_c$, $\beta_{k,c,c} = 1$, and $\beta_{k,c,c'} = 0$ for $c' \neq c$. The learned coefficients allow the stacked model to down-weight redundant or poorly calibrated classifiers, and the off-diagonal terms ($c' \neq c$) capture systematic class confusions in individual base classifiers.

Non-linear combiners (Sec. 15.3.7) apply unchanged: replace the multinomial logistic head with any classifier that accepts the stacked probability vectors $(\mathbf{p}_1, \ldots, \mathbf{p}_K)$ as input and produces a C-dimensional probability vector.

15.4.6. Hard versus Soft Outputs

The rules above produce a probability vector $\mathbf{p}_{\text{fused}}$, from which one can take arg max (top-1) or report the top-k predictions ranked by $p_{\text{fused},c}$. If only hard labels $\hat{y}_k$ are available from the base classifiers, only plurality vote (and a degenerate weighted vote where each classifier contributes a one-hot indicator) applies.

Example 15.2: Three classifiers produce probability vectors for a $C = 3$ classification task:

Classifier	$p_{k,1}$	$p_{k,2}$	$p_{k,3}$	$\hat{y}_k$
$k = 1$	0.55	0.40	0.05	1
$k = 2$	0.50	0.45	0.05	1
$k = 3$	0.10	0.85	0.05	2

Assume balanced priors ($\pi_c = 1/3$).

- *Plurality vote*: class 1 receives 2 votes, class 2 receives 1 vote. $\hat{y}_{\text{fused}} = 1$.
- *Soft vote*: $\mathbf{p}_{\text{soft}} = (0.383, 0.567, 0.050)$. $\hat{y}_{\text{fused}} = 2$.
- *Log-probability fusion*: $\sum_k \log p_{k,1} = -2.40$, $\sum_k \log p_{k,2} = -2.18$, $\sum_k \log p_{k,3} = -8.99$. $\hat{y}_{\text{fused}} = 2$.

The confident vote of classifier 3 for class 2 tips both soft vote and log-probability fusion toward class 2, while plurality vote ignores the confidence and follows the count.

15.5. Sequential Combining

Goal: Sequential processing of independent noisy measurements.

The ensemble methods in the previous section assume that all K classifier outputs (or measurements) are available simultaneously. In many practical settings, however, measurements arrive one at a time (streaming data), and a decision can/must be updated after each new observation.

15.5.1. Sequential Log-Odds Update

Consider a single classifier applied to a sequence of K independent noisy measurements $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_K$ of the same underlying quantity. After measurement $\mathbf{x}_t$, the classifier outputs the probability $p_t = f_{\mathbf{w}}(\mathbf{x}_t)$.

In the batch LLR fusion, all measurements are combined at once:

$$\text{logit}_{\text{fused}} = \sum_{k=1}^{K} \ln \frac{p_k}{1-p_k} \tag{15.27}$$

The same result can be obtained *sequentially*. Define the accumulated log-odds after t measurements as

$$L_t = L_{t-1} + \underbrace{\ln \frac{p_t}{1-p_t}}_{\text{logit}_t}, \quad L_0 = 0 \tag{15.28}$$

After each update, the fused probability is

$$p_{\text{fused},t} = \sigma(L_t) = \frac{1}{1+e^{-L_t}} \tag{15.29}$$

The sequential formulation produces identical results to the batch LLR fusion: $L_K = \text{logit}_{\text{fused}}$. Its advantage is that the decision can be monitored and potentially made before all K measurements are collected.

> The initialization $L_0 = 0$ corresponds to an uninformative (equal) prior, i.e., $p_{\text{fused},0} = \sigma(0) = 0.5$. A non-equal prior $\pi_1 = \Pr(y=1)$ can be incorporated by setting $L_0 = \ln \frac{\pi_1}{1-\pi_1}$.

A key advantage of the sequential formulation is that the decision can be made as soon as sufficient evidence has accumulated, without waiting for all K measurements.

Fixed-threshold decision At each step t, compare the accumulated log-odds to a threshold:

$$\hat{y}_t = \begin{cases} 1 & L_t \geq \tau_1 \\ 0 & L_t \leq \tau_0 \\ \text{continue} & \tau_0 < L_t < \tau_1 \end{cases} \tag{15.30}$$

where $\tau_1 > 0$ and $\tau_0 < 0$ are decision thresholds. If L_t falls between the two thresholds, no decision is made and the next measurement is collected.

Confidence-based stopping Equivalently, the decision can be expressed in terms of the fused probability $p_{\text{fused},t} = \sigma(L_t)$. Stop when the confidence exceeds a desired level γ:

$$\text{stop at time } t^* = \min\left\{t : p_{\text{fused},t} \geq \gamma \;\text{ or }\; p_{\text{fused},t} \leq 1-\gamma\right\} \tag{15.31}$$

where $\gamma \in (0.5, 1)$ is the confidence threshold. For example, $\gamma = 0.95$ means the decision is made when the fused probability exceeds 95% for either class.

Example 15.3: A classifier is applied to $K = 4$ sequential noisy measurements with outputs $p_1 = 0.7$, $p_2 = 0.8$, $p_3 = 0.6$, $p_4 = 0.75$.

t	p_t	logit_t	L_t	$p_{\text{fused},t} = \sigma(L_t)$
1	0.7	$\ln\frac{0.7}{0.3} = 0.847$	0.847	0.700
2	0.8	$\ln\frac{0.8}{0.2} = 1.386$	2.233	0.903
3	0.6	$\ln\frac{0.6}{0.4} = 0.405$	2.638	0.933
4	0.75	$\ln\frac{0.75}{0.25} = 1.099$	3.737	0.977

- *Fixed-threshold stopping* ($\tau_1 = 2$, $\tau_0 = -2$): the decision $\hat{y} = 1$ is reached at $t = 2$, since $L_2 = 2.233 > \tau_1$. Measurements p_3, p_4 are not needed.
- *Confidence-based stopping* ($\gamma = 0.95$): With a strict confidence threshold $\gamma = 0.95$, the decision is delayed to $t = 4$ ($p_{\text{fused},4} = 0.977 > 0.95$). A higher threshold requires more evidence but yields a more reliable result (97.7% confidence vs. 90.3%).

15.5.2. Multiple Independent Sensors

The previous subsection considers repeated measurements from a single classifier. In many applications, N different independent sensors (or classifiers) observe the same underlying state, each producing its own probability estimate. In this case:

1. Since the sensors are independent, the log-odds combine by summation (LLR fusion).
2. Sequential update (Sec. 15.5.1) is performed with combined log-odds.

15.5.3. Connection to Bayesian Learning (*)

Goal: Relate the sequential log-odds update to the Bayesian belief update framework.

The sequential log-odds update is a special case of Bayesian belief updating.

Belief: A *belief* $\pi(H_i)$ is a probability assigned to hypothesis H_i, representing the agent's confidence that H_i is the true state of the world. The belief vector satisfies $\pi(H_i) \geq 0$ and $\sum_i \pi(H_i) = 1$.

Consider a binary hypothesis set $\{H_0, H_1\}$ (e.g., $y = 0$ vs. $y = 1$) with prior beliefs $\pi_{\text{old}}(H_0)$ and $\pi_{\text{old}}(H_1)$. After observing new evidence x, the posterior is updated using Bayes' rule:

$$\pi_{\text{new}}(H_i) = \frac{\pi_{\text{old}}(H_i) \cdot L(H_i; x)}{\sum_j \pi_{\text{old}}(H_j) \cdot L(H_j; x)} \tag{15.32}$$

where $L(H_i; x)$ is the likelihood of the observation x under hypothesis H_i.

In the sequential setting, the posterior after observation x_t becomes the prior for observation x_{t+1}. Taking the log-ratio of the two hypotheses after t observations yields

$$\ln \frac{\pi(H_1)}{\pi(H_0)} = \underbrace{\ln \frac{\pi_0(H_1)}{\pi_0(H_0)}}_{\text{log-prior odds}} + \sum_{k=1}^{t} \underbrace{\ln \frac{L(H_1; x_k)}{L(H_0; x_k)}}_{\text{log-likelihood ratio}_k} \tag{15.33}$$

This is exactly the sequential log-odds update with $L_0 = \ln \frac{\pi_0(H_1)}{\pi_0(H_0)}$ and $\text{logit}_k = \ln \frac{L(H_1;x_k)}{L(H_0;x_k)}$.

When a probabilistic classifier outputs $p_k = f_{\mathbf{w}}(\mathbf{x}_k)$, its logit $\ln \frac{p_k}{1-p_k}$ serves as an estimate of the log-likelihood ratio. The sequential log-odds update therefore implements approximate Bayesian inference using classifier outputs as surrogate likelihood ratios.

Example 15.4: A sensor classifies objects as planes (H_1) or non-planes (H_0). The prior belief is $\pi_{\text{old}}(H_1) = 0.3$, $\pi_{\text{old}}(H_0) = 0.7$. A new measurement x has likelihoods $L(H_1; x) = 0.8$ and $L(H_0; x) = 0.2$. Applying Eq. (15.32):

$$\begin{aligned} \pi_{\text{new}}(H_1) &= \frac{0.3 \times 0.8}{0.3 \times 0.8 + 0.7 \times 0.2} = \frac{0.24}{0.38} \approx 0.632 \\ \pi_{\text{new}}(H_0) &= \frac{0.7 \times 0.2}{0.38} \approx 0.368 \end{aligned} \tag{15.34}$$

A single observation shifted the belief from 30% to 63.2% in favor of "plane." This posterior becomes the prior for the next measurement.

15.6. Multiple Instance Learning (MIL)

Goal: Classify *bags* of instances under weak supervision: only the bag label is observed during training, while the labels of the individual instances remain hidden.

In MIL, the i-th training example is a bag of n_i instances,

$$B_i = \{\mathbf{x}_{i,1}, \mathbf{x}_{i,2}, \ldots, \mathbf{x}_{i,n_i}\}, \tag{15.35}$$

with a single observed bag label $Y_i \in \{0, 1\}$. Each instance has a latent label $y_{i,j} \in \{0, 1\}$ that is not available during training. The *standard MIL assumption* relates the two:

$$Y_i = \max_{j=1,\ldots,n_i} y_{i,j}, \tag{15.36}$$

i.e., a bag is positive if and only if at least one of its instances is positive. This setting reduces annotation cost whenever labeling every instance is impractical (e.g., a whole-slide pathology image labeled only as "tumor/no tumor," with the individual image patches left unlabeled).

Two learning strategies are common: (i) train an instance-level classifier $f(\mathbf{x}_{i,j}) = p_{i,j}$ and aggregate the per-instance probabilities into a bag probability P_i; or (ii) train a bag-level classifier that maps the whole set B_i directly to P_i. Both strategies ultimately depend on an aggregation rule

$$P_i = g(p_{i,1}, p_{i,2}, \ldots, p_{i,n_i}), \tag{15.37}$$

which plays the same role that the fusion rules of Sec. 15.3 play for ensembles of classifiers, except the index now runs over *instances within one bag* rather than *classifiers on one sample*. MIL is therefore ensemble aggregation applied along the instance axis.

The *aggregated bag prediction* P_i acts as the bridge between the micro-level instances and the macro-level bag. It represents the model's overall confidence that the entire bag belongs to the positive class ($Y_i = 1$). Because the true labels of individual instances are hidden, the model must combine its per-instance predictions into this single collective score so the loss can be calculated against the only available ground truth: the bag label.

Four aggregation rules appear frequently:

$$\begin{aligned} P_i^{\max} &= \max_j p_{i,j}, \\ P_i^{\text{mean}} &= \tfrac{1}{n_i} \sum_{j=1}^{n_i} p_{i,j}, \\ P_i^{\text{noisy-OR}} &= 1 - \prod_{j=1}^{n_i} (1 - p_{i,j}), \\ P_i^{\text{att}} &= \sum_{j=1}^{n_i} \alpha_{i,j}\, p_{i,j}, \qquad \alpha_{i,j} = \frac{\exp\!\left(\mathbf{w}^\top \phi(\mathbf{x}_{i,j})\right)}{\sum_{j'} \exp\!\left(\mathbf{w}^\top \phi(\mathbf{x}_{i,j'})\right)}. \end{aligned} \tag{15.38}$$

Max-pooling is the direct probabilistic counterpart of Eq. (15.36). Mean-pooling is the instance-axis analog of soft vote combining (Sec. 15.3.3). Noisy-OR is the analog of LLR fusion (Sec. 15.3.5) under the assumption that instance predictions are conditionally independent. Attention pooling learns the mixing weights $\alpha_{i,j}$ from the data and plays the role of a learned combiner (Sec. 15.3.7).

Example 15.5 (The Security Guard Analogy): Imagine a security guard tasked with determining if a 5-room building (the bag) is dangerous ($Y_i = 1$). The guard sweeps each room (the instances) and assigns a threat probability $p_{i,j}$ to each:

- Room 1: 1% (0.01)
- Room 2: 2% (0.02)
- Room 3: 95% (0.95) – Weapon found
- Room 4: 0% (0.00)
- Room 5: 2% (0.02)

How the guard reports the overall threat level of the building (P_i) depends entirely on the aggregation rule:

- *Max-pooling* ($P_i = 0.95$): The guard reports the building is 95% dangerous. The building is only as safe as its most dangerous room. This perfectly aligns with the standard MIL assumption.
- *Mean-pooling* ($P_i = 0.20$): The guard averages the threat levels and reports the building is 20% dangerous. Because there are four relatively safe rooms, the severe threat in Room 3 gets dangerously diluted.
- *Noisy-OR* ($P_i = 1 - (0.99)(0.98)(0.05)(1.00)(0.98) \approx 0.952$): The guard calculates the probability that *at least one* room is dangerous, assuming independent events. The combined probability remains very high.

Under the standard MIL assumption, a single strongly positive instance should determine the bag label, so max-pooling and noisy-OR are consistent aggregators. Mean-pooling mirrors Soft Vote and tends to dangerously under-weight rare but decisive positive instances.

Because individual instance labels $y_{i,j}$ are completely absent during training, a standard per-instance loss cannot be computed directly. Instead, training an instance-level classifier relies on one of three main paradigms:

End-to-End Backpropagation

The instance-level classifier (parameterized by $\mathbf{w}$) and the aggregation function g are optimized simultaneously. Because instance-level labels are unknown, the loss is evaluated strictly at the bag level. The aggregated bag prediction P_i is compared to the true bag label Y_i using standard Binary Cross-Entropy (Sec. 8.4):

$$\mathcal{L} = -\left[Y_i \log P_i + (1 - Y_i) \log(1 - P_i)\right] \tag{15.39}$$

During backpropagation, the error signal must pass through the aggregation function g to reach the instance-level predictions $p_{i,j}$. By the chain rule, g acts as a structural bottleneck that dictates exactly *how* the bag-level gradient is distributed among the individual instances. The choice of aggregation directly determines this gradient flow:

- *Max-pooling*: The max operation acts as a hard mathematical switch. The gradient routes entirely to the single instance that yielded the maximum probability. The remaining $n_i - 1$ instances in the bag receive a gradient of zero, meaning their parameters receive no learning signal from this forward pass.
- *Mean-pooling*: The gradient is divided evenly by n_i and passed to all instances in the bag. Every instance is updated equally, which means noisy or non-informative instances receive the exact same learning signal as the truly predictive ones.
- *Attention-pooling*: The gradient is scaled proportionally by the learned attention weights $\alpha_{i,j}$. The model dynamically routes stronger learning signals to the instances it deems most relevant (high α), while suppressing the updates for instances treated as background noise (low α).

Label Inheritance

A heuristic approach where every instance inside a bag is simply assigned the bag's label ($y_{i,j} = Y_i$). A standard binary classifier is then trained on the instances directly. This introduces significant noise into positive bags (since a positive bag may contain many negative instances), but it is computationally cheap and often used as a pre-training step to initialize weights.

Iterative Pseudo-Labeling

An alternating expectation-maximization (EM) style approach to refine instance predictions:

1. Train an initial classifier (often using Label Inheritance).
2. Use this classifier to score all instances in the training bags. Assign hard "pseudo-labels" based on these scores (e.g., in a positive bag, label the top-scoring instance as 1 and the rest as 0).
3. Retrain the classifier using these new pseudo-labels, and repeat the process until convergence.

Iterative pseudo-labeling is vulnerable to a confirmation-bias failure mode: the classifier's own predictions become the next round's training targets, so the loop can converge to a self-

consistent solution with no genuine predictive power while still reporting high training and cross-validation accuracy. This is the same spurious-signal pathology that the sanity checks of Chapter 13 are designed to detect. After pseudo-labeling converges, the final model should be validated with:

- A *permutation test on the bag labels* (Sec. 13.3.1): shuffle only the outer Y_i, re-run the full pseudo-labeling loop, and verify that the bag-level accuracy collapses to the chance level J_{chance} (Eq. 13.2).
- A *random feature baseline* (Sec. 13.3.4): replace the instance features $\mathbf{x}_{i,j}$ with noise of the same dimensions and confirm that the pseudo-labeling loop no longer produces above-chance bag accuracy.

A pseudo-labeling pipeline that passes both checks provides meaningful evidence that the recovered instance labels reflect genuine signal rather than a self-reinforced artifact.

16. Vanilla NN

Goal: One-layer NN.

16.1. Definition

16.1.1. Single neuron

Goal: Relation between the generalized linear model (logistic regression) and a single neuron.

A single neuron is the neural-network realization of the generalized linear model $\hat{y}_i = g(\mathbf{w}^T \mathbf{x}_i)$ introduced in the logistic regression chapter (Eq. (8.1)).

Single neuron: A single neuron computes its output in two steps (Fig. 16.1):

$$a = w_1 x_1 + w_2 x_2 + \cdots + w_N x_N + b = \mathbf{w}^T \mathbf{x} + b \tag{16.1}$$

$$z = g(a) \tag{16.2}$$

where

- x_i are input features, $i = 1, \ldots, N$
- w_i are learnable weights and $\mathbf{w}$ is weight vector,
- b is an explicit bias term that is separated from the weight vector,
- a is intermediate linear combination (pre-activation) in (16.1),
- $g(\cdot)$ is activation function, e.g. $\sigma(\cdot)$, or identity,
- z is output after applying the activation function (neuron output or post-activation)

When $g = \sigma$ (sigmoid), the single neuron reduces to logistic regression.

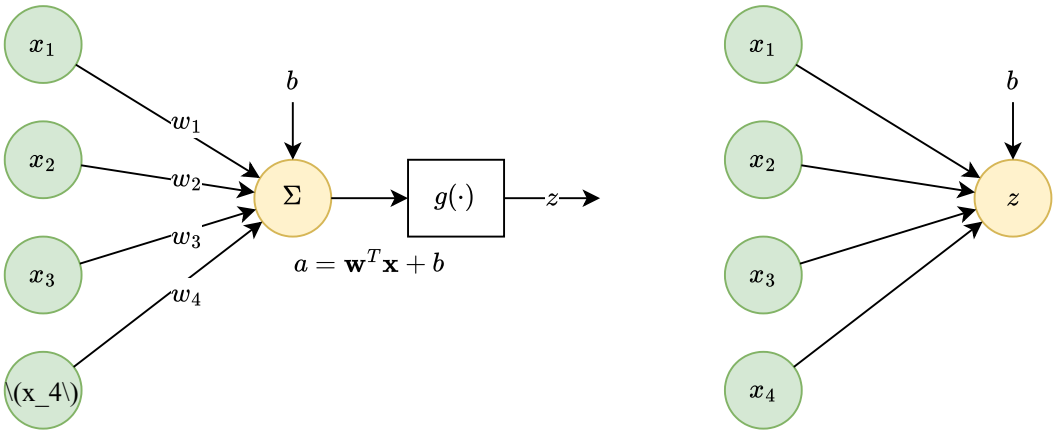
> In the ML part of this book the design matrix $\mathbf{X} \in \mathscr{R}^{M \times N}$ has samples as rows. In DL notation the transposed form $\mathbf{X}^T$ is commonly used, so that samples are columns.

16.1.2. Layered representation

Goal: From a single neuron to a multi-layer network: matrix formulation for one sample and for the entire dataset.

A single neuron in a layer is indexed by its layer number $[k]$. Applying the single-neuron definition to every neuron in a layer yields the vector form:

$$\mathbf{a}^{[k]} = \left(\mathbf{W}^{[k]}\right)^T \mathbf{z}^{[k-1]} + \mathbf{b}^{[k]} \tag{16.3}$$



(a) Fully annotated (b) Compact

Figure 16.1.: Single neuron: fully annotated (left) and compact (right) representation.

$$\mathbf{z}^{[k]} = g_k\left(\mathbf{a}^{[k]}\right) \tag{16.4}$$

For the one-layer network in Fig. 16.2a ($k = 1$, four inputs, five hidden neurons) the pre-activation vector expands as

$$\underbrace{\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix}}_{\mathbf{a}^{[1]}} = \underbrace{\begin{bmatrix} w_{11} & w_{21} & w_{31} & w_{41} \\ w_{12} & w_{22} & w_{32} & w_{42} \\ w_{13} & w_{23} & w_{33} & w_{43} \\ w_{14} & w_{24} & w_{34} & w_{44} \\ w_{15} & w_{25} & w_{35} & w_{45} \end{bmatrix}}_{\left(\mathbf{W}^{[1]}\right)^T} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}}_{\mathbf{x}} + \underbrace{\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{bmatrix}}_{\mathbf{b}^{[1]}} \tag{16.5}$$

Multi-layer model (single sample):

$$\mathbf{a}^{[k]} = \left(\mathbf{W}^{[k]}\right)^T \mathbf{z}^{[k-1]} + \mathbf{b}^{[k]} \tag{16.6}$$

$$\mathbf{z}^{[k]} = g_k\left(\mathbf{a}^{[k]}\right) \tag{16.7}$$

$$\mathbf{z}^{[0]} = \mathbf{x}_i \tag{16.8}$$

When the entire dataset is processed at once, each column of $\mathbf{Z}^{[k]}$ corresponds to one sample.

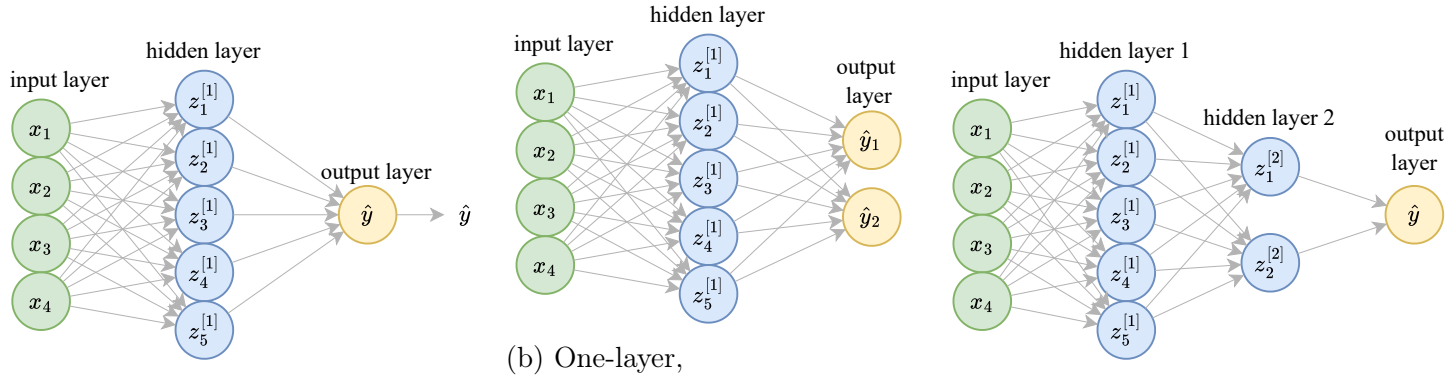
Multi-layer model (all dataset):

$$\mathbf{Z}^{[k]} = g_k\left(\mathbf{W}^{[k]}\mathbf{Z}^{[k-1]} + \mathbf{b}^{[k]}\right) \tag{16.9}$$

$$\mathbf{Z}^{[0]} = \mathbf{X} \tag{16.10}$$

Output is the value of $\mathbf{Z}^{[k]}$ at the output layer.

Fig. 16.2 illustrates three common architectures. A one-layer network with a single output neuron (Fig. 16.2a) performs scalar regression or binary classification. Adding output neurons (Fig. 16.2b) enables multi-target regression or multi-class classification. Stacking hidden layers (Fig. 16.2c) increases the model capacity, allowing the network to learn hierarchical feature representations.



(a) One-layer, single output

(b) One-layer, multi-output

(c) Two hidden layers

Figure 16.2.: NN architectures: (a) single output, (b) multiple outputs, (c) multiple hidden layers.

The number of hidden layers and the number of neurons per layer are *hyperparameters* chosen before training begins. Increasing the number of layers (depth) and of neurons per layer (width) enlarge the model complexity and the number of trainable parameters, so they must be balanced against the available training data to avoid overfitting.

Most of the calculations are matrix addition and multiplication. Recently applied activation functions (e.g., ReLU) also require similar operations.

16.2. Activation Functions

Goal: Introduce nonlinearity into the network to enable learning of complex patterns.

Activation functions are essential components of neural networks:

- Every activation function must be differentiable (or piecewise differentiable) to support back-propagation (Sec. 16.5 below).
- A linear activation function $g(z) = z$ reduces the entire network to linear regression, regardless of depth.
- The choice of activation function affects convergence speed, gradient flow, and computational cost.

Table 16.1 summarizes the common activation functions and their derivatives.

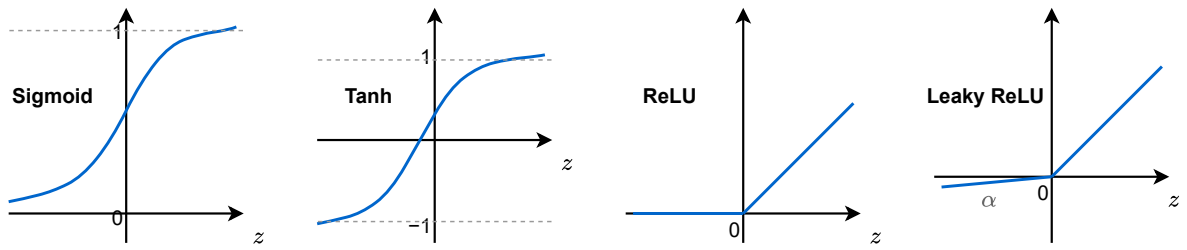


Figure 16.3.: Common activation functions: sigmoid, tanh, ReLU, and Leaky ReLU.

Table 16.1.: Common activation functions and their derivatives.

Name	Function	Range	Derivative
Sigmoid	$f(z) = \dfrac{1}{1+e^{-z}}$	$[0, 1]$	$f'(z) = f(z)(1 - f(z))$
Tanh	$f(z) = \dfrac{e^z - e^{-z}}{e^z + e^{-z}}$	$[-1, 1]$	$f'(z) = 1 - f^2(z)$
ReLU	$f(z) = \max(0, z)$	$[0, \infty)$	$f'(z) = \begin{cases} 1 & z > 0 \\ 0 & z < 0 \end{cases}$
Leaky ReLU	$f(z) = \max(\alpha z, z)$	$(-\infty, \infty)$	$f'(z) = \begin{cases} 1 & z \geq 0 \\ \alpha & z < 0 \end{cases}$

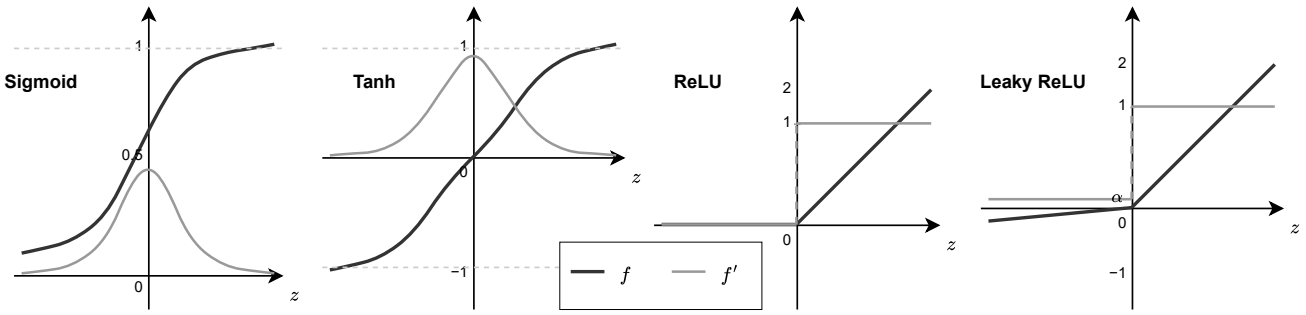


Figure 16.4.: Derivatives of activation functions.

Sigmoid

The sigmoid function maps any real value to the interval $(0, 1)$, making it suitable for probabilistic interpretation.

- Outputs can be interpreted as probabilities.
- Suffers from vanishing gradients: for large $|z|$, the derivative approaches zero, slowing learning.
- Computationally expensive due to the exponential operation.
- Not zero-centered, which can slow convergence.

Tanh

The hyperbolic tangent is a scaled and shifted sigmoid, $\tanh(z) = 2\sigma(2z) - 1$.

- Zero-centered output in $[-1, 1]$, which often improves convergence.
- Still suffers from vanishing gradients for large $|z|$.
- Computationally expensive.

ReLU

The Rectified Linear Unit is currently the most widely used activation function.

- Faster convergence than sigmoid or tanh due to non-saturating gradients for $z > 0$.
- Computationally efficient: only requires a threshold comparison.
- Unbounded output can lead to exploding activations.
- "Dying ReLU" problem: neurons with negative inputs have zero gradient and stop learning.
- Sensitive to weight initialization.

Leaky ReLU

Addresses the dying ReLU problem by allowing a small gradient for negative inputs.

- Uses a small slope α (typically $\alpha = 0.01$) for $z < 0$.
- Prevents neurons from becoming permanently inactive.

16.3. Softmax Layer

Goal: Output layer for multi-class classification with a single-output class.

In multi-class classification with K classes, the target label is represented as a *one-hot* vector.

One-hot encoding: A categorical label $c \in \{1, \dots, K\}$ is encoded as a binary vector $\mathbf{y} \in \{0, 1\}^K$ with exactly one element equal to 1:

$$y_j = \begin{cases} 1 & j = c \\ 0 & j \neq c \end{cases} \qquad j = 1, \dots, K \tag{16.11}$$

For example, with $K = 4$ classes, label $c = 3$ is encoded as $\mathbf{y} = (0, 0, 1, 0)^T$.

The softmax function produces a probability distribution over K classes that can be compared directly against this one-hot target.

Softmax function: Given a probabilistic score s_i for each class, the softmax is defined by

$$\hat{s}_i = f_S(s_i) = \frac{\exp(s_i)}{\sum_{j=1}^{K} \exp(s_j)} \quad i = 1, \dots, K \tag{16.12}$$

The corresponding loss (categorical cross entropy) is given by

$$\mathcal{L} = -\frac{1}{K} \sum_{j=1}^{K} \log(\hat{s}_i) \tag{16.13}$$

For example, $(1, 2, 8) \to (0.001, 0.002, 0.997)$.

Probability simplex: The output vector $\hat{\mathbf{s}} = (\hat{s}_1, \dots, \hat{s}_K)^T$ has nonnegative entries summing to 1, i.e. it lies in the *probability simplex*

$$\Delta^{K-1} = \left\{ \mathbf{p} \in \mathbb{R}^K : \; p_i \geq 0, \; \sum_{i=1}^{K} p_i = 1 \right\}. \tag{16.14}$$

The exponent $K - 1$ is the intrinsic dimension of the simplex: the constraint $\sum_{i=1}^{K} p_i = 1$ removes one degree of freedom, so only $K - 1$ entries can vary independently. For example:

- $K = 2$: Δ^1 is the line segment from $(1, 0)$ to $(0, 1)$, 1-dimensional.
- $K = 3$: Δ^2 is the triangle with vertices $(1, 0, 0), (0, 1, 0), (0, 0, 1)$, 2-dimensional.
- $K = 4$: Δ^3 is a tetrahedron, 3-dimensional.

ArgMax It may be considered as an approximation of arg max operator, when all the values are close to 0 except a one value that is close to 1.

Sigmoid Sigmoid is a special case of softmax function with $s_1 = 0$ and $K = 2$.

16.4. Loss Function

Goal: Select the loss function that matches the task and the output-layer activation.

The loss function measures the discrepancy between predictions and targets. The choice depends on the task:

- **Regression** – Mean Squared Error (MSE):

$$\mathcal{L}_{\text{MSE}} = \frac{1}{M} \sum_{i=1}^{M} (y_i - \hat{y}_i)^2 \tag{16.15}$$

  Typically paired with a linear (identity) output activation. See Ch. 6 for derivation and properties.
- **Binary classification** – Binary Cross-Entropy (BCE):

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{M} \sum_{i=1}^{M} \Big[y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i) \Big] \tag{16.16}$$

  Paired with a sigmoid output activation. See Sec. 8.4 for derivation.
- **Multi-class classification** – Categorical Cross-Entropy (CCE):

$$\mathcal{L}_{\text{CCE}} = -\frac{1}{M} \sum_{i=1}^{M} \sum_{j=1}^{K} y_{ij} \log \hat{y}_{ij} \tag{16.17}$$

  Paired with a softmax output activation.

Table 16.2 summarizes the common task–loss–activation combinations.

Table 16.2.: Common task–loss–activation combinations.

Task	Loss	Output activation
Regression	MSE	Linear (identity)
Binary classification	BCE	Sigmoid
Multi-class classification	CCE	Softmax

16.5. Back-propagation

16.5.1. Concept

Goal: Compute the gradient of the loss with respect to every weight in the network; use it for gradient-descent training (Sec. 3.4).

Because every layer applies differentiable operations, the chain rule gives the partial derivative of the loss with respect to any weight:

$$\frac{\partial \mathcal{L}(y,\hat{y})}{\partial w_{ij}^{[k]}} = \frac{\partial \mathcal{L}(y,\hat{y})}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_{ij}^{[k]}} \tag{16.18}$$

Gradient descent iteratively updates every weight in the negative-gradient direction. For non-convex losses the procedure converges to a local minimum. The training loop is:

1. Initialize weights.
2. Forward propagation $\Rightarrow$ evaluate $\mathcal{L}()$.
3. Back-propagation: compute $\frac{\partial \mathcal{L}(y,\hat{y})}{\partial w_{ij}^{[k]}}$ for all weights.
4. Update weights by GD.
5. Return to step 2.

16.5.2. General Example

Goal: Walk through the chain rule for a single-layer network with MSE loss and sigmoid activation.

Consider MSE loss with a $\frac{1}{2}$ scaling factor for convenience:

$$\mathcal{L}(\mathbf{y},\hat{\mathbf{y}}) = \frac{1}{2M}\sum_{i=1}^{M}(y_i-\hat{y}_i)^2 \tag{16.19}$$

Its derivative with respect to the prediction is

$$\frac{\partial \mathcal{L}(\mathbf{y},\hat{\mathbf{y}})}{\partial \hat{y}_i} = -\frac{1}{M}(y_i-\hat{y}_i) \tag{16.20}$$

The output layer applies a sigmoid activation, $\hat{y}_i = \sigma(a_i^{[l]})$, whose derivative is

$$\frac{\partial \hat{y}_i}{\partial a_i^{[l]}} = \sigma(a_i^{[l]})\Big(1-\sigma(a_i^{[l]})\Big) \tag{16.21}$$

Combining via the chain rule:

$$\frac{\partial \mathcal{L}(\mathbf{y},\hat{\mathbf{y}})}{\partial a_i^{[l]}} = \frac{\partial \mathcal{L}(\mathbf{y},\hat{\mathbf{y}})}{\partial \hat{y}_i}\frac{\partial \hat{y}_i}{\partial a_i^{[l]}} \tag{16.22}$$

The pre-activation is $a_i^{[l]} = \sum_j w_{ij}^{[l]} z_j^{[l-1]} + b_i^{[l]}$, so $\frac{\partial a_i^{[l]}}{\partial w_{ij}^{[l]}} = z_j^{[l-1]}$. The full gradient is therefore

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial w_{ij}^{[l]}} = \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial a_i^{[l]}} \frac{\partial a_i^{[l]}}{\partial w_{ij}^{[l]}} \tag{16.23}$$

This pattern extends to deeper networks: the chain rule propagates the gradient backward through each layer.

16.6. Learning

16.6.1. Stochastic and Mini-Batch Gradient Descent

Basic gradient descent (Section 3.4) applies a single, fixed learning rate to every parameter. In bigger/deep networks this can be inefficient. The modifications below address these issues.

Recall that batch GD computes the gradient over all M samples, mini-batch GD uses a random subset of size $B \ll M$, and the special case $B = 1$ is stochastic gradient descent (SGD). These concepts, together with the notion of an epoch, are defined in Section 3.4.

Mini-batch GD algorithm:

- Initialize weights $\mathbf{w}$ and learning rate α.
- Repeat until stop condition is met:
  For each epoch:
  - Randomly shuffle the dataset.
  - Partition into mini-batches $(\mathbf{X}_i, \mathbf{y}_i)$.
  - For each mini-batch, update

$$\mathbf{w}_{i+1} = \mathbf{w}_i - \alpha \nabla \mathcal{L}_{\mathbf{w}}(\mathbf{y}_i, \hat{\mathbf{y}}_i)$$

- Each mini-batch is representative of the overall dataset patterns.
- The noise in mini-batch gradients allows the optimizer to escape shallow local minima.
- Smaller batches may improve generalization [13].

Advanced optimizers Several methods improve upon basic SGD by adapting the learning rate per parameter. Notable examples include Momentum, RMSProp, and Adam (Adaptive Moment Estimation). Adam is currently the default choice in most deep-learning frameworks.

16.6.2. Learning control

Goal: Trade-off between:

Too low learning rate	slow convergence
Too high learning rate	does not converge to a minimum
Too few epochs	under-fitting
Too many epochs	overfitting

Two hyperparameters govern the training loop (Table 16.3):

- the learning rate,
- and the stopping criterion.

Table 16.3.: Learning-control strategies.

Hyperparameter	Strategy	Description
Learning rate	Constant	Pre-defined fixed value
	Exponential decay	Decrease as a function of the epoch number
	Loss-based	Reduce when the loss plateaus for several epochs
Stopping	Fixed budget	Pre-defined number of epochs
	Early stopping	Halt when validation loss stops improving

Instead of a fixed learning rate, α may vary as a function of the iteration number k,

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \alpha_k \nabla \mathcal{L}_{\mathbf{w}}$$

Common strategies include constant rate, step decay (reduce α by a factor every few epochs), and exponential decay $\alpha_k = \alpha_0 e^{-\lambda k}$.

16.7. Weight Initialization

Problem with ReLU For zero initialization, the gradients are zero. For example, we initialize all the biases to 0 and the weights with some constant α. If we forward propagate an input $\mathbf{x}$ in this network, the output of all hidden units will be $g(\mathbf{w}^T\mathbf{x})$. All hidden units will have identical influence on the cost, which will lead to identical gradients. Thus, all neurons will evolve symmetrically throughout training, effectively preventing different neurons from learning different things.

The common approach is to use some random initialization with either Gaussian or uniform distribution with some distribution parameters. Examples are Xavier, Glorot (Gaussian distribution) and He (Gaussian/uniform distribution) initializations [11]. Different methods use different distribution parameters according to number of neurons in current and previous layer.

Xavier initialization An example of initialization is Xavier initialization (or one of its derived methods), that for every layer l uses

$$\begin{aligned} \mathbf{W}^{[l]} &\sim N(\mu = 0, \sigma^2 = \frac{1}{n^{[l-1]}}) \\ b^{[l]} &= 0 \end{aligned} \tag{16.24}$$

where $n^{[l-1]}$ is the number of neuron in layer $l-1$ (previous layer).

16.8. Summary

- Data

  - Train/test and validation for hyper-parameters evaluation
  - Batch size
- NN model
  - Neural layers: number of layers, number of neurons in each level, activation functions
  - Special layers
  - Output layer that matches loss function
  - Complexity grows exponentially with additional layers, $O(2^k)$
- Loss
  - Minimization goal according to the task, e.g. regression (mse) or classification (ce)
- Metric
  - Performance evaluation, e.g. rmse for regression or accuracy for classification
- Learning algorithm: Adam, ...
  - Learning rate
  - Hyper-parameters (very rare)
  - Learning rate control, e.g. exponential decrease.
  - Early stopping by loss or metric convergence
  - Number of epochs (GD steps)

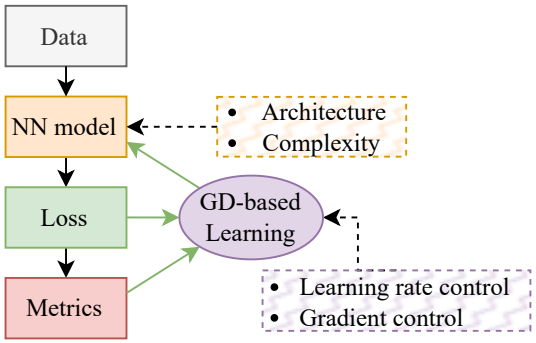


Figure 16.5.: Summary of VNN design choices.

17. From Machine Learning to Deep Learning

Goal: Frame where neural networks fit among learning algorithms and motivate the shift to deep learning.

Contents

17.1. Three Paradigms of Learning **229**
17.2. Prominent Historical Points **230**
17.2.1. Deep Blue (1997) 230
17.2.2. The Netflix Prize (2006–2009) 231
17.2.3. The ImageNet Challenge (2010–2017) 231
17.3. Enablers **232**
17.3.1. The Data Explosion 232
17.3.2. Hardware: GPUs and TPUs 233
17.3.3. Scaling Behaviour 233
17.4. Applications and Architectures **234**

17.1. Three Paradigms of Learning

Predictive systems can be grouped into three families along an axis of how much of the feature engineering is delegated to the learner (Fig. 17.1).

Rule-based The mapping from input to output is hand-crafted by a domain expert; no parameters are estimated from data. Examples: classical computer-vision filters, fire-control logic in automatic weapon systems, expert rule engines in medicine, and, at its most elaborate, IBM's *Deep Blue* chess machine (Sec. 17.2.1).

Classical machine learning A human designs a feature representation and a learning algorithm fits the mapping from features to output. Examples: signal classification from engineered features (amplitude, dominant frequency, spectral entropy), random-forest credit scoring on tabular features.

Deep learning Both the feature representation and the mapping are learned end-to-end from raw data. Examples: large language models such as ChatGPT, diffusion-based image generators, end-to-end speech recognition.

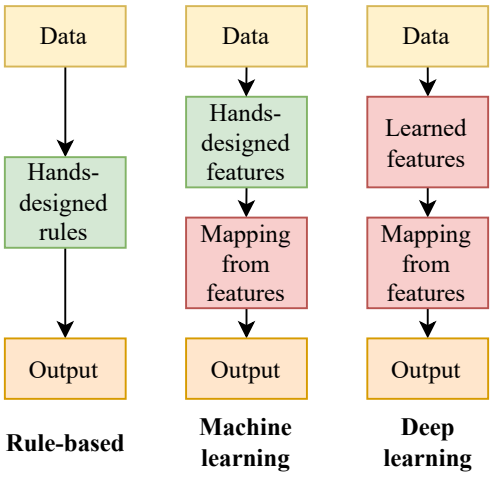


Figure 17.1.: Three learning paradigms, ordered by how much of the pipeline is learned from data rather than designed by hand.

17.2. Prominent Historical Points

Three landmarks, one per paradigm of Sec. 17.1, mark the change in ML/DL. The first is a rule-based system at the peak of what hand-crafted engineering can reach; the second is a public competition won by classical machine learning just before deep learning broke through; the third is a competition overturned by it.

17.2.1. Deep Blue (1997)

On May 11, 1997, in New York, IBM's chess machine *Deep Blue* defeated the reigning world champion Garry Kasparov $3\frac{1}{2}$–$2\frac{1}{2}$ in a six-game match, the first time a reigning champion lost a classical match to a machine. Kasparov had beaten an earlier version 4–2 in Philadelphia the year before. The machine was a 30-node IBM RS/6000 SP parallel computer carrying 480 custom chess chips, and it evaluated roughly 2×10^8 board positions per second.

What matters here is not the hardware but where the chess knowledge came from. Deep Blue searched the game tree and scored the resulting positions with an evaluation function of about 8,000 hand-written features, whose weights were tuned by hand with the help of grandmasters, supported by an opening book of about 4,000 positions and exhaustive tables for all endings with five pieces or fewer. Nothing in the system was estimated from data in the sense used in the rest of this book: every piece of chess knowledge was placed there by a human, and the machine supplied only search speed (Table 17.1).

Table 17.1.: The three landmarks of this section, and how the work was split between the human and the data.

Landmark	Paradigm	Written by a human	Learned from data
Deep Blue (1997)	Rule-based	all the chess knowledge	nothing
BellKor (2009)	Classical ML	the features and the models	how to combine them
AlexNet (2012)	Deep learning	the network layout	the features themselves

This is the ceiling of the rule-based paradigm rather than a failure of it. Hand-written rules reach superhuman quality when the domain is fully observable, discrete, defined by exact rules, and judged by a compact success criterion. They give no route to perception tasks, where no expert can write down the features in the first place, which is the saturation level drawn for rule-based systems in Fig. 17.2. Game playing itself eventually crossed that ceiling from the other side: *AlphaGo* (2016) and *AlphaZero* (2017) replaced the hand-tuned evaluation with a network trained by self-play, leaving nothing but the rules of the game hand-coded.

17.2.2. The Netflix Prize (2006–2009)

A public competition to predict user ratings of films from previous ratings alone, with no auxiliary information about users or films. The goal was to beat the in-house Netflix algorithm (*Cinematch*) by 10 %, with a grand prize of \$1,000,000 and several \$50,000 progress prizes along the way. The dataset contained 100,480,507 ratings given by 480,189 users to 17,770 movies on a discrete scale of 1–5; predictions were allowed to be real-valued. Each (user, movie) entry was timestamped, but no demographic, textual, or visual information about users or films was provided.

Performance was measured by the root-mean-square error

$$\mathrm{RMSE} = \sqrt{\frac{1}{M}\sum_{i=1}^{M}(\hat{r}_i - r_i)^2},$$

on a held-out test set. The Cinematch baseline of 0.9525 had to be reduced to 0.8572. The contest ran from October 2, 2006 to July 26, 2009 and attracted about 2,000 teams from over 150 countries.

Within a few weeks of launch, an independent contestant (Simon Funk) publicly described a stochastic-gradient form of *matrix factorization* that beat Cinematch by several percent; for the rest of the contest, latent-factor models stayed at the core of every leading entry. The grand prize was awarded to the team *BellKor's Pragmatic Chaos*, a late-stage merger of three groups, whose final submission combined more than a hundred classical predictors (matrix factorization, restricted Boltzmann machines, gradient-boosted trees, and k-nearest-neighbor models) through linear blending. A rival team, *The Ensemble*, matched the score numerically but submitted twenty minutes later and lost on the tie-breaking rule. Deep architectures had not yet emerged as competitive predictors.

The lasting influence of the prize was less on the algorithms themselves than on the surrounding practices: large-scale public benchmarks with held-out test sets, public leaderboards, and team-merger dynamics later codified by Kaggle. Two cautionary footnotes also followed: the winning ensemble was never deployed in production because its engineering cost outweighed the rating-quality gain over a simpler in-house model, and a planned sequel competition was canceled in 2010 after researchers showed that supposedly anonymized users in the released dataset could be re-identified by cross-referencing public ratings on other sites.

17.2.3. The ImageNet Challenge (2010–2017)

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) was an annual computer-vision competition held from 2010 to 2017. The ILSVRC subset (ImageNet-1K) contains 1,000 classes with

1,281,167 training, 50,000 validation, and 100,000 test images; the larger ImageNet-21K contains 14,197,122 images across 21,841 classes. Performance was reported using both top-1 and top-5 error, where the *top-5 error* is the fraction of test images for which the correct label is not among the model's five most probable predictions (formal definitions in Sec. 18.2.1).

On September 30, 2012, the convolutional neural network *AlexNet* broke from the classical Fisher-vector pipelines that had won the previous years, enabled by GPU training of a deep CNN (Table 17.2). The jump is widely credited with launching the modern deep-learning era: within five years deep models dominated the leaderboard, error fell below the human reference of about 5 %, and the competition was retired.

Table 17.2.: Selected ILSVRC top-5 classification results. 2011 and the 2012 runner-up still rely on hand-engineered features; AlexNet replaces them with a learned deep representation.

Year	Entry	Approach	Top-1 err. (%)	Top-5 err. (%)
2011	XRCE (winner)	hand-crafted features	~ 50	25.8
2012	ISI (runner-up)	hand-crafted features	~ 46	26.2
2012	*AlexNet* (winner)	8-layer CNN + **GPU debut**	37.5	15.3
2014	GoogLeNet (winner)	22-layer CNN	~ 26	6.7
2015	ResNet (winner)	152-layer CNN	~ 19	3.57
2017	SENet (winner)	CNN + attention	~ 17	2.25

The challenge was retired after 2017 because it had stopped discriminating between top entries: 29 of the 38 submitting teams had top-5 error below 5 %, and the leading entries had fallen to roughly 2 %, comparable to the estimated label-noise floor of the test set itself, so further gains were no longer measuring image-recognition quality. The community moved on to harder benchmarks (detection, segmentation, larger and noisier web-scale corpora, and self-supervised pre-training).

17.3. Enablers

What changed between the Netflix Prize and the ImageNet jump was not the theoretical idea of deep networks (it had been around for decades), but three practical enablers that arrived together: the data needed to train an over-parametrised model (Sec. 17.3.1), the hardware to fit it in a tractable wall-clock (Sec. 17.3.2), and the empirical observation that deep networks keep improving as both grow (Sec. 17.3.3).

17.3.1. The Data Explosion

A modern megapixel camera produces 1920×1080 pixels, each described by three colour channels (RGB) of 1 byte (8 bit) each. At a typical frame rate of 30 fps, the uncompressed stream is

$$1920 \times 1080 \times 3 \times 30 \;\approx\; 2 \times 10^8 \,\text{bytes/s} \;\approx\; 200\,\text{MB/s}.$$

A single hour of raw HD video therefore exceeds 700 GB; multiply by millions of cameras, microphones, and clickstreams across the internet and the corpus available for training is enormous.

17.3.2. Hardware: GPUs and TPUs

Training and inference in neural networks consist almost entirely of dense matrix and tensor operations, which parallelise trivially across thousands of cores. GPUs are designed for exactly this workload (Table 17.3).

Table 17.3.: High-level comparison between CPU and GPU for neural-network workloads.

Parameter	**CPU**	**GPU (NVIDIA)**	**Note**
Single-thread speed	Fast	Slow	Latency-optimised vs. throughput-optimised
Parallel cores	8–64	$\sim 10^4$	GPU has many simple cores
Execution model	Independent threads (MIMD)	Same program, many data items (SIMT)	GPU well-suited to tensor ops
Cache	Hardware-managed, multi-level	Programmer-managed shared memory	GPU memory hierarchy is explicit
Memory	DDR, large, slower	HBM, 16–80 GB on-board, much faster bandwidth	GPU memory dominates throughput
Power	~ 65 W	150–700 W	Datacentre GPUs need dedicated cooling
Multi-device scaling	Limited	NVLink / InfiniBand fabrics	Routine scaling to thousands of GPUs
Precision	FP64 standard	FP32 / FP16 / BF16 / FP8	Lower precision $\Rightarrow$ higher throughput

Two GPU-specific features were decisive for deep learning:

- *Tensor Cores* (NVIDIA Volta and later): dedicated units that perform a fused mixed-precision matrix-multiply-accumulate in a single cycle, giving an order-of-magnitude throughput advantage on the operations that dominate training.
- *Mature software stack* (CUDA, cuDNN, cuBLAS), which is the base to all the major deep-learning frameworks (PyTorch, TensorFlow, JAX).

An example of further specialization is Google's *Tensor Processing Unit* (TPU), an application-specific integrated circuit (ASIC) built around a large array matrix multiplier and optimized exclusively for tensor workloads. NVIDIA, AMD, and several startups now ship comparable accelerators.

17.3.3. Scaling Behaviour

Data and hardware would be of little use if the model could not benefit from them. The defining property of deep networks is that they keep improving as both grow.

Scalability of a model: A model is *scalable* if both of the following hold:

- **Statistical scalability:** test-set performance keeps improving as the training set grows.
- **Computational scalability:** the training procedure can absorb the larger dataset at a tolerable wall-clock and memory cost (e.g., through parallelization across GPUs).

A linear or otherwise low-capacity model has an asymptotic performance ceiling: once enough data is available to estimate its parameters reliably, further samples bring diminishing returns because the hypothesis class itself is too restrictive. Deep networks behave differently: when the model is enlarged in tandem with the dataset, performance keeps improving across many orders of magnitude in dataset size. This empirical observation is the operational definition of "deep learning at scale" (Fig. 17.2), and it is what made the AlexNet jump in Sec. 17.2 possible.

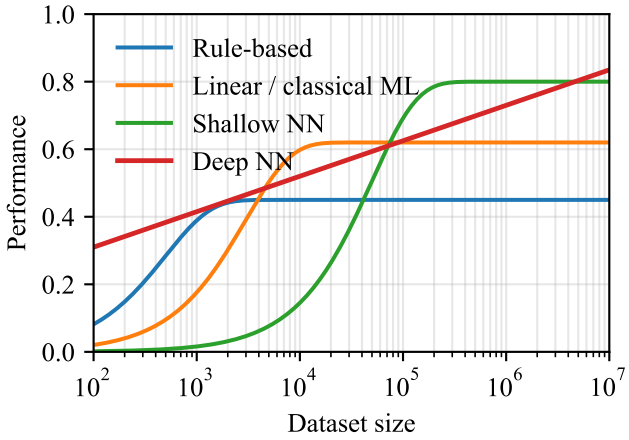


Figure 17.2.: Schematic performance vs. dataset size. Rule-based, linear, and shallow models saturate at successively higher levels; deep networks continue to improve as data and capacity grow together.

17.4. Applications and Architectures

Neural networks are flexible enough to tackle problems across very different data modalities. Table 17.4 maps a few canonical pairings of data type, task, and architecture; the rest of this part of the book introduces these architectures one by one.

Table 17.4.: Canonical pairings of data modality, task, and neural-network architecture.

Learning type	**Data modality**	**Example**	**Task**	**Architecture**
Supervised	Tabular	Home / car description	Price prediction	Fully-connected NN
	Sequential (time)	Audio waveform	Speech transcription	Recurrent NN
	Sequential (text)	Sentence	Translation	Recurrent NN / Transformer
	Image (2D)	Photographs	Image classification	Convolutional NN
Unsupervised	Image (2D)	Unlabelled images	Representation learning	Contrastive / self-supervised NN
	Image (2D)	Noisy images	Denoising	Autoencoder

18. Computer Vision Metrics

Goal: Present main computer-vision tasks and the corresponding solutions.

Contents

18.1. Benchmark datasets **237**
18.2. Image Classification **237**
18.2.1. Performance metrics 238
18.2.2. Practical recipe 239
18.2.3. Loss (*) 240
18.3. Image Segmentation **240**
18.3.1. Pixel Accuracy 242
18.3.2. Intersection over Union 244
18.3.3. Dice coefficient 246
18.3.4. Hausdorff distance 247
18.3.5. Panoptic quality 248
18.3.6. Instance segmentation 250
18.3.7. Dice Loss 251
18.4. Object Detection **254**
18.4.1. Performance metrics 255
18.4.2. Architecture 262
18.4.3. Notes 264

Notation Throughout this chapter, an input image $\mathbf{X} \in \mathbb{R}^{H \times W \times C_{\text{in}}}$ has spatial dimensions:

- H (height, in pixels) and
- W (width, in pixels), with
- $C_{\text{in}} \in \{1, 3\}$ channels (grayscale or RGB).

C denotes the number of target classes (excluding the background class 0 when present).

The chapter covers three computer-vision tasks:

- Section 18.2 *(Image Classification)*: assign a single class label to the whole image.
- Section 18.3 *(Image Segmentation)*: assign a class label to every pixel.
- Section 18.4 *(Object Detection)*: locate and classify a variable number of objects with bounding boxes.

All three assume a class list fixed at training time. The open-set alternative, where an image is mapped to an embedding instead of a label, is the subject of the next chapter (Ch. 19).

18.1. Benchmark datasets

Throughout this chapter, metric conventions are tied to the public benchmarks that introduced them. The three datasets below cover detection, instance/panoptic segmentation, and urban-scene semantic segmentation; ImageNet, used in classification, is covered separately in Sec. 17.2.3.

Pascal VOC The PASCAL Visual Object Classes challenge ran from 2005 to 2012 and was the first widely-adopted detection/segmentation benchmark. The standard VOC2007 / VOC2012 splits contain $\sim$ 11,000 images annotated with 20 object classes (person, vehicles, animals, indoor objects) plus a background class. VOC introduced two conventions that this chapter inherits: the lenient IoU threshold $\tau = 0.5$ for box matching (Sec. 18.4.1) and 11-point interpolated AP for the pre-2010 editions, replaced by all-point interpolation from 2010 onward. VOC numbers are not directly comparable to COCO numbers because the threshold and interpolation conventions differ.

COCO Common Objects in Context (Lin et al., 2014) is the de-facto modern benchmark for detection, instance segmentation and panoptic segmentation. It contains $\sim$ 330,000 images with 80 "thing" classes (animals, vehicles, people, household objects) for detection / instance segmentation and a further 91 "stuff" classes (sky, road, grass) used by the panoptic split. COCO introduced the stricter mAP@[.5:.95] convention (averaging over ten IoU thresholds, Eq. 18.24), the $\text{AP}_{\text{small/medium/large}}$ size breakdown that exposes the small-object weakness of IoU, and the mask-AP variant that reuses the detection recipe with mask IoU. Most pretrained detection/segmentation checkpoints in this chapter are trained on COCO.

Cityscapes Cityscapes (Cordts et al., 2016) is the standard semantic-segmentation benchmark for urban driving: 5,000 images with fine pixel-level annotations and a further $\sim$ 20,000 with coarse annotations, drawn from 50 European cities, evaluated on 19 classes (road, sidewalk, car, pedestrian, sign, ...) plus a *void / ignore* label for ambiguous or out-of-vocabulary pixels. The void label is the convention to remember: it must be excluded from both numerator and denominator of every metric, otherwise mIoU is silently inflated.

18.2. Image Classification

Goal: Map an input image to a single class label.

Image classification is the canonical computer-vision task. The model $f_\theta(\mathbf{X})$ returns a C-length probabilistic vector on the simplex Δ^{C-1} (Sec. 16.3); the prediction is $\widehat{y} = \text{argmax}_c[f_\theta(\mathbf{X})]$.

Two sub-flavors are worth distinguishing:

- *Single-label* (multi-class) classification: exactly one of C classes per image. The output layer is a softmax (Sec. 16.3) of size C and the loss is categorical cross-entropy. This is the focus of the present section.
- *Multi-label* classification: any subset of C tags per image (e.g. scene attributes, chest-X-ray findings). The softmax is replaced by C independent sigmoids and the loss is the sum of binary cross-entropies, one per class. The architectural pattern is otherwise identical.

18.2.1. Performance metrics

Classification reuses the binary/multi-class machinery of Chapter 12: the confusion matrix and everything derived from it. The image-level specifics are top-n accuracy/error (Sec. 18.2.1) and the class-averaged metrics introduced below.

Top-n accuracy and error

A probabilistic multi-class classifier outputs a vector of class probabilities $\hat{\mathbf{p}}_i = (\hat{p}_{i,1}, \dots, \hat{p}_{i,C})$. Ordinary accuracy uses only the top-1 prediction $\arg\max_c \hat{p}_{i,c}$. With many classes, or with semantically similar classes (e.g. ImageNet has 1000 classes including several breeds of dog), the top-1 prediction is often a near-miss while the true class still appears among the most probable few. Allowing the classifier to return a short list of candidates and counting it correct whenever the true class lies in the list gives the top-n family of metrics.

Top-n accuracy: A prediction is counted as correct if the true class is among the n classes with the highest predicted probability. Let $\mathcal{T}_n(\hat{\mathbf{p}}_i)$ denote the set of indices of the n largest entries of $\hat{\mathbf{p}}_i$. Then

$$\text{Top-}n \text{ accuracy} = \frac{1}{M} \sum_{i=1}^{M} \mathbb{1}\left[y_i \in \mathcal{T}_n(\hat{\mathbf{p}}_i)\right], \tag{18.1}$$

where $\mathbb{1}[\cdot]$ is the indicator function (Sec. B.2). Top-1 accuracy coincides with ordinary accuracy. Top-n accuracy is non-decreasing in n and reaches 100% at $n = C$. Top-5 is the conventional companion to top-1 on ImageNet-scale benchmarks.

Top-n error: The complement of top-n accuracy: the fraction of samples for which the true class is *not* among the n most probable predictions,

$$\text{Top-}n \text{ error} = 1 - \text{Top-}n \text{ accuracy} = \frac{1}{M} \sum_{i=1}^{M} \mathbb{1}\left[y_i \notin \mathcal{T}_n(\hat{\mathbf{p}}_i)\right]. \tag{18.2}$$

ImageNet leaderboards (Sec. 17.2.3) conventionally report top-5 *error*, because once accuracy reaches the high 90 % range, differences are easier to read on the error scale (e.g. 5.1 % vs. 3.6 % rather than 94.9 % vs. 96.4 %).

Example 18.1: A 5-class image classifier produces the following probability vectors on four test images (true-class probability in bold):

i	true	$\hat{p}_1$	$\hat{p}_2$	$\hat{p}_3$	$\hat{p}_4$	$\hat{p}_5$	top-1	top-3
1	2	0.10	***0.55***	0.20	0.10	0.05	yes	yes
2	4	*0.40*	0.30	0.15	**0.10**	0.05	no	no
3	3	0.05	0.10	**0.30**	*0.45*	0.10	no	yes
4	1	***0.50***	0.20	0.15	0.10	0.05	yes	yes

For sample 3, the top-3 predicted classes are $\{4, 3, 2\}$, which contains the true class 3, so the prediction counts as correct under top-3. Aggregating:

$$\text{Top-1 accuracy} = \frac{2}{4} = 50\%, \qquad \text{Top-3 accuracy} = \frac{3}{4} = 75\%.$$

The gap reveals that the classifier often ranks the correct class highly even when its top-1 pick is wrong, suggesting that the model has captured useful structure that a stricter accuracy measure obscures.

Class-averaged metrics

Macro-F_1 and balanced accuracy (Sec. 12.6.2): per-class F_1 or recall averaged uniformly across classes. Mandatory on long-tailed datasets where top-1 is dominated by the majority classes.

18.2.2. Practical recipe

Table 18.1 summarises a representative set of available models. Three families recur:

- *Mobile/edge* (MobileNetV3, EfficientNet-B0): ≤ 10 M parameters, optimised for latency on CPU/mobile.
- *General-purpose* (ResNet-50, EfficientNet-B4, ConvNeXt-Tiny, ViT-B/16): 20–90 M parameters, the default starting point when no hard latency budget applies.
- *High-capacity* (ConvNeXt-Large, ViT-L/16 and bigger): hundreds of millions of parameters.

To quantify model size and computational complexity, we use parameter counts (associated with memory footprint) and multiply-accumulate operations (MACs, associated with latency and computational cost).

Multiply-accumulate operation (MAC): A *multiply-accumulate operation* (MAC) computes the product of two numbers and adds that product to an accumulator:

$$a \leftarrow a + (b \times c). \tag{18.3}$$

In the context of deep learning, a MAC operation (consisting of one multiplication and one addition) represents the fundamental computation in dot products, matrix multiplications, and convolutions. A single MAC is approximately equivalent to two floating-point operations (FLOPs):

$$\text{FLOPs} \approx 2 \times \text{MACs}. \tag{18.4}$$

Doubling parameters buys a few accuracy points

An *accuracy point* here means one absolute percentage point on the Top-1 column of Table 18.1 (e.g. moving from 76.1% to 77.1% is a gain of one point). The gap between ResNet-50 and ConvNeXt-Large is roughly 8 points despite an 8× parameter increase. Pick the smallest model whose accuracy clears the application threshold.

Table 18.1.: Common backbones for image classification. Parameter counts, multiply-accumulate operations (MACs) at 224×224, and ImageNet-1k (Sec. 17.2.3) top-1 accuracy are reference figures from the original papers; absolute numbers shift with the training recipe but the ranking is stable.

Backbone	Params [M]	MACs [G]	Top-1 [%]
MobileNetV3-Large	5.5	0.22	75.2
EfficientNet-B0	5.3	0.39	77.1
ResNet-18	11.7	1.8	69.8
ResNet-50	25.6	4.1	76.1
EfficientNet-B4	19	4.2	82.9
ConvNeXt-Tiny	28	4.5	82.1
ViT-B/16	86	17.6	81.8
ConvNeXt-Base	89	15.4	83.8
ConvNeXt-Large	198	34.4	84.3

18.2.3. Loss (*)

For single-label classification with one-hot target $\mathbf{y} \in \{0,1\}^C$ (Sec. 9.1.3) and softmax probabilities $\mathbf{p} = g_\phi(\mathbf{z}) \in \Delta^{C-1}$ (the probability simplex; Sec. 16.3), the categorical cross-entropy is

$$\mathcal{L}_{\text{CE}} \;=\; -\sum_{c=1}^{C} y_c \log p_c \;=\; -\log p_y, \tag{18.5}$$

where y denotes the index of the true class. Two refinements are routine:

Focal loss

The cross-entropy contribution of an example with predicted true-class probability p_y is $-\log p_y$, which remains non-negligible even when p_y is already large. With many easy examples (the typical regime when one class dominates), their summed gradient swamps the contribution of the few hard ones. Focal loss multiplies CE by a modulating factor that vanishes as $p_y \to 1$,

$$\mathcal{L}_{\text{focal}} \;=\; -(1-p_y)^\gamma \log p_y, \tag{18.6}$$

where $\gamma \geq 0$ controls the steepness: $\gamma = 0$ recovers CE and larger γ flattens the loss in the well-classified region (Fig. 18.1). Concretely, at $p_y = 0.9$ (an easy example), CE is $-\log 0.9 \approx 0.105$ while focal with $\gamma = 2$ scales it by $(1-0.9)^2 = 0.01$ down to ≈ 0.00105, a $100\times$ down-weighting. At $p_y = 0.1$ (a hard example), the factor $(1-0.1)^2 = 0.81$ leaves the loss almost untouched, so the gradient is redirected toward the hard examples.

18.3. Image Segmentation

Goal: Per-pixel classification; assigning a class index to *every* pixel.

Segmentation maps an input image to a label map of the same spatial size,

$$f_\theta : \mathcal{X} \to \{0,1,\ldots,C\}^{H\times W},$$

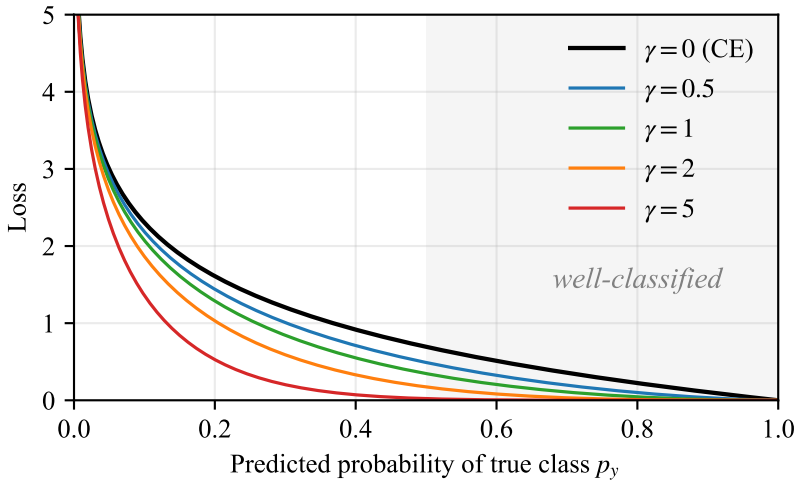


Figure 18.1.: Focal loss $-(1-p_y)^\gamma \log p_y$ vs. predicted probability of the true class, for $\gamma \in \{0, 0.5, 1, 2, 5\}$. $\gamma = 0$ is cross-entropy. Larger γ suppresses the loss on well-classified examples ($p_y > 0.5$, shaded) while leaving hard examples (p_y small) nearly unchanged.

Throughout this subsection:

- $\mathbf{M} \in \{0, 1, \ldots, C\}^{H \times W}$ is the ground-truth label map,
- $\hat{\mathbf{M}} = f_\theta(\mathbf{X})$ the predicted label map, and
- $|\cdot|$ denotes pixel count.

Background and foreground

One class index is reserved for the *background* ($c = 0$): pixels that do not belong to any object of interest (sky, empty tissue, unlabelled regions). All other classes ($c = 1, \ldots, C$) are *foreground* classes (tumour, road, vehicle, sheep), so C counts the foreground classes only and the total number of class indices is $C + 1$.

In a binary problem ($C = 1$) there is a single foreground class and $y_i \in \{0, 1\}$ indicates whether pixel i belongs to it. With multiple foreground classes, each is treated separately when computing per-class losses and metrics: class c is its own foreground ($y_i^{(c)} = 1$) and every other label, including the background (one-vs-rest), is its background ($y_i^{(c)} = 0$). Foreground classes are typically the minority, and the pixel fraction each one occupies drives most of the design choices below (loss, metrics, sampling).

Objectives

Image segmentation comes in three flavors, distinguished by what each pixel of the output carries. It is useful to split classes into

- *stuff* (uncountable regions: sky, road, grass) and
- *things* (countable objects: sheep, car, person).

The corresponding objectives are:

- *Semantic segmentation*: every pixel gets a single class label $c \in \{0, \dots, C\}$. Two sheep standing side by side are merged into one connected `sheep` region; the model cannot tell them apart.
- *Instance segmentation*: every *thing* pixel gets a pair (class, instance id), so the two sheep receive different ids; stuff pixels (sky, road) are left unlabelled.
- *Panoptic segmentation*: every pixel gets a pair (class, instance id). Things receive a unique id per object; each stuff class shares one id. It is a strict superset of the two above.

Example 18.2 (Two sheep on a road): Consider a 2×4 toy image (see also Fig. 18.2) with sky in the top row, road in the bottom row, and two sheep occupying the bottom-left and bottom-right pixels. Let class indices be `sky` $= 0$, `road` $= 1$, `sheep` $= 2$. The three flavors produce

$$\underbrace{\begin{pmatrix} 0 & 0 & 0 & 0 \\ 2 & 1 & 1 & 2 \end{pmatrix}}_{\text{semantic}}, \quad \underbrace{\begin{pmatrix} \varnothing & \varnothing & \varnothing & \varnothing \\ (2,1) & \varnothing & \varnothing & (2,2) \end{pmatrix}}_{\text{instance}}, \quad \underbrace{\begin{pmatrix} (0,1) & (0,1) & (0,1) & (0,1) \\ (2,1) & (1,1) & (1,1) & (2,2) \end{pmatrix}}_{\text{panoptic}}.$$

Semantic loses the count (one `sheep` region or two cannot be recovered from the label map); instance recovers the count but ignores sky and road; panoptic labels every pixel and distinguishes the two sheep.

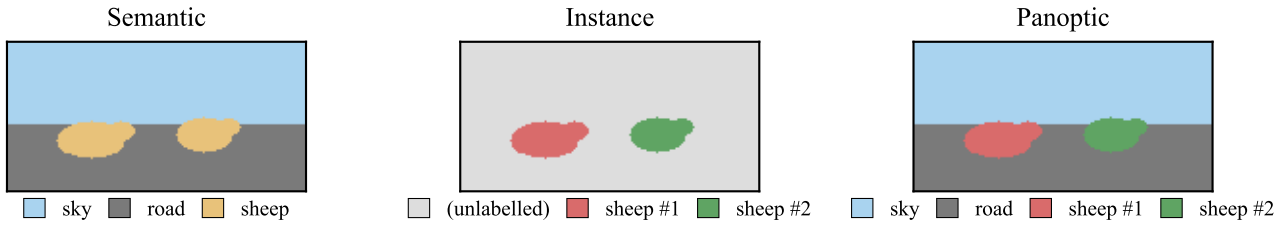


Figure 18.2.: The three segmentation flavours on the same synthetic scene. Semantic merges the two sheep into a single class region; instance separates them but leaves stuff classes (sky, road) blank; panoptic assigns a (class, instance id) pair to every pixel.

> Note, per-pixel annotation is expensive.

Counter-indications for segmentation:

- If a single image-level label suffices, use a classifier that cheaper to label, train and run (Sec. 18.2).
- If approximate locations of multiple objects are enough, use detection (Sec. 18.4).

18.3.1. Pixel Accuracy

Pixel-level confusion matrix A multi-class confusion matrix $N \in \mathbb{N}^{(C+1)\times(C+1)}$ (the per-pixel analogue of the per-sample matrix of Sec. 12.6.1),

$$N_{cc'} = \sum_{i,j} \mathbb{1}\left[M_{ij} = c\right] \, \mathbb{1}\left[\hat{M}_{ij} = c'\right], \tag{18.7}$$

counts how many pixels of true class c are predicted as c'. Every per-pixel metric below is a simple function of N.

Figure 18.3.: The raw street scene used as the running example for the dense-prediction tasks below: a view over a residential neighborhood. Figure 18.4 shows its semantic segmentation and Fig. 18.10 its object detection.

Pixel accuracy Pixel accuracy is the diagonal sum normalized by total pixels,

$$\mathrm{PA} \;=\; \frac{\sum_{c} N_{cc}}{\sum_{c,c'} N_{cc'}}, \tag{18.8}$$

treating each pixel as an equal-weight classification example.

Per-class pixel accuracy The per-class pixel accuracy is the fraction of class-c pixels recovered,

$$\mathrm{PA}_c \;=\; \frac{N_{cc}}{\sum_{c'} N_{cc'}} \;=\; \frac{\mathrm{TP}_c}{\mathrm{TP}_c + \mathrm{FN}_c}, \tag{18.9}$$

which is the recall on class c. The denominator $\sum_{c'} N_{cc'}$ is the row-c sum of the confusion matrix, i.e. the number of pixels whose true label is c, counted across every predicted class c'.

Mean pixel accuracy

$$\mathrm{mPA} \;=\; \frac{1}{C+1}\sum_{c} \mathrm{PA}_c \;=\; \frac{1}{C+1}\sum_{c} \frac{N_{cc}}{\sum_{c'} N_{cc'}} \tag{18.10}$$

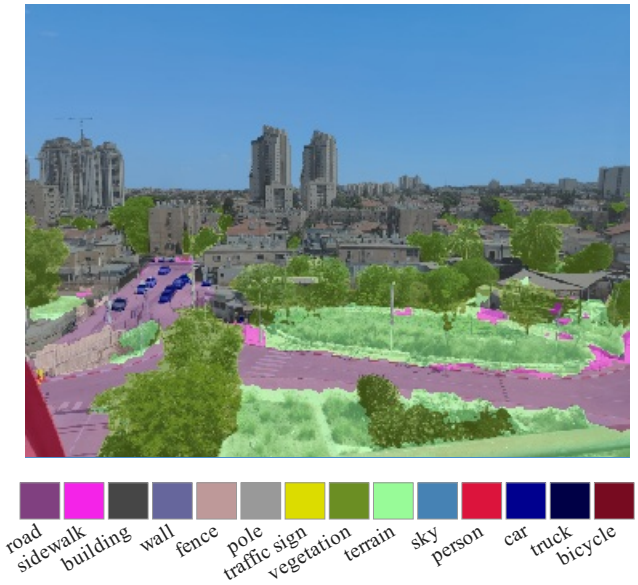


Figure 18.4.: Semantic segmentation of the street scene of Fig. 18.3: a Cityscapes-pretrained SegFormer assigns every pixel one of 19 classes (the legend shows those present). Unlike detection (Fig. 18.10), there are no instances: all cars share the one `car` color, and *stuff* classes (sky, building, road, vegetation, terrain) are labeled too. Note the `road`/`sidewalk`/`terrain` distinctions, which a bounding box cannot express.

averages PA_c equally over the $C+1$ classes ($+1$ stands for the background class). It is the macro-averaged per-class recall (Sec. 12.6.2), less biased by class frequency than PA.

18.3.2. Intersection over Union

Definition

For two regions A and B (pixel sets, or, in Sec. 18.4, bounding boxes) the *Intersection over Union* (IoU, also called the Jaccard index) is

$$\mathrm{IoU}(A,B) \;=\; \frac{|A\cap B|}{|A\cup B|} \;=\; \frac{|A\cap B|}{|A|+|B|-|A\cap B|}, \tag{18.11}$$

with $\mathrm{IoU}\in[0,1]$ (1 for perfect overlap, 0 for disjoint regions).

> Why IoU and not pixel accuracy? IoU penalises both false positives (predicted pixels outside the ground truth) and false negatives (ground-truth pixels missed). Pixel accuracy on a sparse foreground class can stay near 1 even when the model predicts everything as background; IoU collapses to 0 in that case.

IoU per class

For class c, treating "class c" vs. "not class c" as a binary problem, apply the IoU definition (Eq. 18.11) with $B=\{\,\text{pixels with } M_{ij}=c\,\}$ (the actual class-c region) and $A=\{\,\text{pixels with } \hat{M}_{ij}=c\,\}$ (the predicted class-c region). The matrix entries read out the three set cardinalities directly:

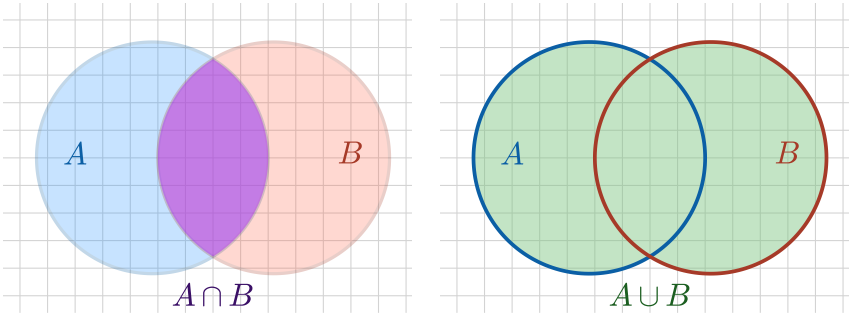


Figure 18.5.: Intersection over union for two binary masks. In this example, $\text{IoU} = |A \cap B|/|A \cup B| \approx 0.22$

- $|A \cap B| = N_{cc} = \text{TP}_c$ — pixels that are class c in both the truth and the prediction (diagonal entry);
- $|B| = \sum_{c'} N_{cc'} = \text{TP}_c + \text{FN}_c$ — row-c sum, all actual class-c pixels;
- $|A| = \sum_{c'} N_{c'c} = \text{TP}_c + \text{FP}_c$ — column-c sum, all predicted class-c pixels.
- $|A \cup B| = |A| + |B| - |A \cap B|$ - inclusion-exclusion formula.

Substituting into Eq. 18.11 gives

$$\text{IoU}_c \;=\; \frac{N_{cc}}{\sum_{c'} N_{cc'} + \sum_{c'} N_{c'c} - N_{cc}} \;=\; \frac{\text{TP}_c}{\text{TP}_c + \text{FP}_c + \text{FN}_c}. \tag{18.12}$$

Mean IoU

The *mean IoU* averages over classes,

$$\text{mIoU} \;=\; \frac{1}{C+1} \sum_{c=0}^{C} \text{IoU}_c, \tag{18.13}$$

and is the de-facto reporting metric. A frequency-weighted variant is

$$\text{fwIoU} = \sum_{c} f_c \, \text{IoU}_c \tag{18.14}$$

with f_c the pixel frequency of class c (the fraction of all pixels whose true label is c).

Median IoU

The *median IoU* is a robust alternative,

$$\text{medIoU} \;=\; \underset{c=0,\ldots,C}{\text{median}} \; \text{IoU}_c, \tag{18.15}$$

useful on long-tailed benchmarks: a single rare class whose IoU collapses to near zero can drag mIoU down disproportionately, while medIoU is unaffected by such per-class outliers. It is sometimes reported alongside mIoU rather than instead of it.

What counts as a good (m)IoU? There is no universal cutoff, but on standard benchmarks:

- mIoU < 0.5: typically weak; usually a sign of class imbalance or that boundary errors dominate.
- 0.5–0.7: typical range for hard datasets.
- 0.7–0.8: strong; current SOTA.
- > 0.9: near-perfect, realistic only for easy or binary masks.

Per-class IoU varies widely within a single model: background and large classes routinely sit at 0.9+, while thin or rare classes (poles, pedestrians, lesions) often languish at 0.3–0.5 even for SOTA. This spread is why mIoU is always reported alongside per-class values.

18.3.3. Dice coefficient

The Dice coefficient (the F_1 score in its binary form) is

$$\mathrm{Dice}(A, B) \;=\; \frac{2|A \cap B|}{|A| + |B|} \;=\; \frac{2\,\mathrm{TP}}{2\,\mathrm{TP} + \mathrm{FP} + \mathrm{FN}}. \tag{18.16}$$

Dice and IoU are monotone transforms of each other,

$$\mathrm{Dice} \;=\; \frac{2\,\mathrm{IoU}}{1 + \mathrm{IoU}}, \qquad \mathrm{IoU} \;=\; \frac{\mathrm{Dice}}{2 - \mathrm{Dice}}, \tag{18.17}$$

so they rank methods identically; Dice gives a higher score for the same overlap (Fig. 18.6).

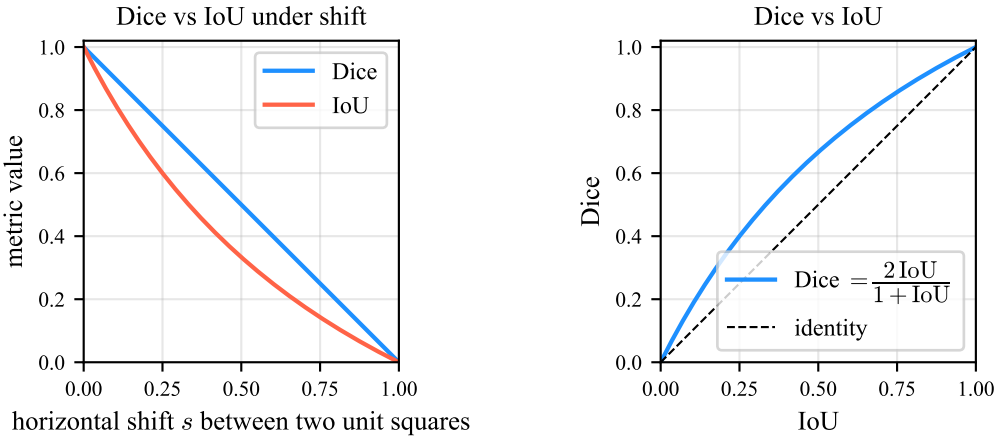


Figure 18.6.: Dice and IoU compared. *Left:* both metrics as a function of the horizontal shift s between two unit squares, where $s \in [0, 1]$ is measured in units of the square side ($s = 0$ full overlap, $s = 1$ a full-width offset leaving no overlap); Dice decays linearly while IoU drops faster. *Right:* Dice plotted directly against IoU via (18.17), lying above the identity line so Dice always reports a higher score for the same overlap.

In medical imaging Dice is the dominant choice; in autonomous driving and general computer vision IoU is. The choice is conventional, so do not report both as if they were independent evidence.

On datasets where one class covers most pixels (typical for medical or satellite imagery) PA is misleading: predicting all-background scores near 1 in PA but 0 in mIoU. Always report mIoU or per-class IoU on imbalanced data.

Example 18.3 (Pixel metrics on a 4×4 image): A binary segmentation problem (foreground vs. background) on a 4×4 image. Ground truth $\mathbf{M}$ has the foreground (class 1) in the top-left 2×2 block; the prediction $\hat{\mathbf{M}}$ has it shifted one pixel to the right:

$$M = \begin{array}{|c|c|c|c|}\hline 1 & 1 & 0 & 0 \\ \hline 1 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline \end{array} \qquad \hat{M} = \begin{array}{|c|c|c|c|}\hline 0 & 1 & 1 & 0 \\ \hline 0 & 1 & 1 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 \\ \hline \end{array}$$

Cells are shaded by the class-1 outcome: TP, FP, FN, TN. Compute pixel accuracy, and the IoU and Dice for each class.

Solution: For the foreground (class 1) the per-pixel counts are TP $= 2$, FP $= 2$, FN $= 2$, TN $= 10$. Then

$$\begin{aligned} \text{PA} &= (2 + 10)/16 = 0.75, \\ \text{IoU}_1 &= 2/(2 + 2 + 2) = 1/3 \approx 0.33, \\ \text{Dice}_1 &= 2 \cdot 2/(2 \cdot 2 + 2 + 2) = 1/2 = 0.50. \end{aligned}$$

For the background (class 0) the roles swap: its $\text{TP}_0 = 10$ are the class-1 true negatives, while the class-1 FP and FN become $\text{FN}_0 = 2$ and $\text{FP}_0 = 2$. Then

$$\begin{aligned} \text{IoU}_0 &= 10/(10 + 2 + 2) = 5/7 \approx 0.71, \\ \text{Dice}_0 &= 2 \cdot 10/(2 \cdot 10 + 2 + 2) = 5/6 \approx 0.83, \end{aligned}$$

so mIoU $= (\text{IoU}_0 + \text{IoU}_1)/2 \approx 0.52$. PA looks reasonable, but IoU exposes that only one third of the foreground union is correctly predicted; the large background inflates both PA and the background's own scores, which is why the foreground IoU (or mIoU) is the honest summary. Dice falls between PA and IoU as expected from (18.17).

18.3.4. Hausdorff distance

IoU and Dice score *area* overlap and are dominated by the interior of large objects, so a prediction can score well even while its boundary wanders far from the truth in a few places. The *Hausdorff distance* measures exactly that: the largest gap between the two boundaries (Fig. 18.7). It is the distance-based complement to the overlap metrics above and the standard companion to Dice in medical imaging, where contour carries clinical information (cf. the boundary metrics in Sec. 18.3.3). This distance exists in multiple variants.

The Hausdorff distance is a length, not a ratio in $[0, 1]$: it has no fixed scale and is comparable across results only at a fixed image resolution and voxel spacing.

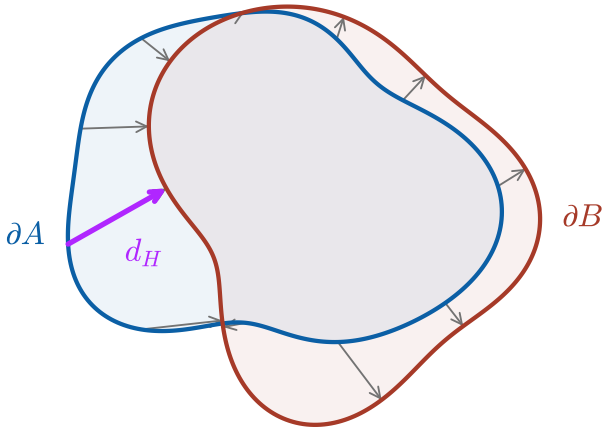


Figure 18.7.: Hausdorff distance between a predicted contour ∂A (blue) and a ground-truth contour ∂B (red).

18.3.5. Panoptic quality

Panoptic quality (PQ) grades a result at the level of *whole segments* rather than individual pixels. It is built in two stages, as follows.

Step 1: match segments Pair a predicted segment p with a ground-truth segment g only when their mask overlap clears $\mathrm{IoU}(p, g) > 0.5$.

The threshold of 0.5 is what keeps the matching clean: no two predictions can both exceed 0.5 IoU with the same ground truth, so each ground truth matches at most one prediction and the pairing is unique. The matching sorts every segment into three *counts* (segment-level, not pixels):

- TP – matched pairs (p, g);
- FP – predicted segments with no matching ground truth (spurious detections);
- FN – ground-truth segments with no matching prediction (misses).

Step 2: grade with two questions A good panoptic result must answer two independent questions, and PQ measures each with its own factor.

- *Did the model find the right segments?* This is *recognition quality* (RQ), the F_1 score over segments, treating each segment as one classification example:

$$\mathrm{RQ} \;=\; \frac{|\mathrm{TP}|}{|\mathrm{TP}| + \frac{1}{2}|\mathrm{FP}| + \frac{1}{2}|\mathrm{FN}|}.$$

  The $\frac{1}{2}$ weights split each unmatched segment evenly between FP and FN, the standard F_1 form.
- *When it does find a segment, how tightly does the mask fit?* This is *segmentation quality* (SQ), the mean IoU over the matched pairs only (unmatched segments do not enter):

$$\mathrm{SQ} \;=\; \frac{\sum_{(p,g)\in \mathrm{TP}} \mathrm{IoU}(p, g)}{|\mathrm{TP}|}.$$

Step 3: combine Panoptic quality is the product of the two factors,

$$\mathrm{PQ} \;=\; \underbrace{\frac{\sum_{(p,g)\in \mathrm{TP}} \mathrm{IoU}(p,g)}{|\mathrm{TP}|}}_{\mathrm{SQ}} \;\times\; \underbrace{\frac{|\mathrm{TP}|}{|\mathrm{TP}| + \tfrac{1}{2}|\mathrm{FP}| + \tfrac{1}{2}|\mathrm{FN}|}}_{\mathrm{RQ}}. \tag{18.18}$$

Because it is a product, a high PQ demands both: detect the right segments (high RQ) *and* outline them tightly (high SQ). A model that detects everything with sloppy masks, or produces a few perfect masks while missing most objects, scores poorly.

Example 18.4 (Panoptic quality on a small image): A panoptic prediction returns four segments $\hat{I}_1, \ldots, \hat{I}_4$ on an image whose ground truth contains three segments I_1, I_2, I_3. The pairwise mask IoUs between candidate pairs are

$$\mathrm{IoU}(\hat{I}_1, I_1) = 0.82, \quad \mathrm{IoU}(\hat{I}_2, I_2) = 0.61, \quad \mathrm{IoU}(\hat{I}_4, I_3) = 0.34,$$

and all other pairs have IoU $= 0$. The matching at threshold 0.5 is shown in Fig. 18.8. Compute SQ, RQ and PQ.

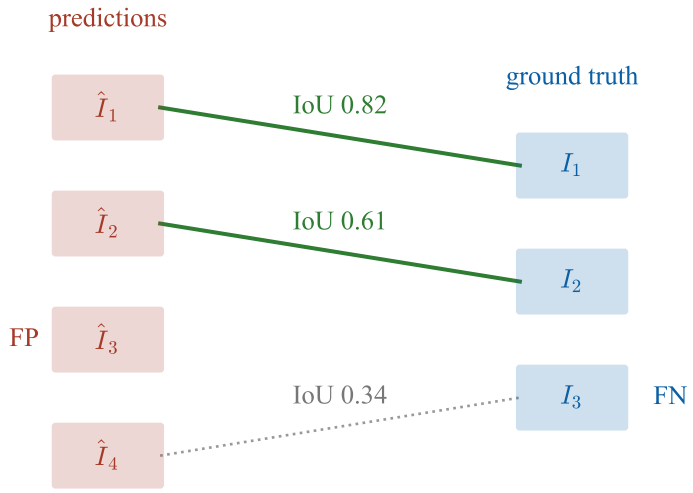


Figure 18.8.: Matching predicted instances $\hat{I}_k$ to ground-truth instances I_k by mask IoU at threshold 0.5. Solid green lines mark TP pairs (IoU above 0.5); the dotted gray line is a below-threshold candidate that fails to match. Unmatched predictions become FP, unmatched ground truths become FN.

Solution: 1. Match segments

Apply the 0.5 threshold. Only $(\hat{I}_1, I_1)$ at 0.82 and $(\hat{I}_2, I_2)$ at 0.61 clear IoU > 0.5, so they form the TP set; $(\hat{I}_4, I_3)$ at 0.34 does not match. Reading off the figure, the counts are

$$|\mathrm{TP}| = 2, \qquad \underbrace{|\mathrm{FP}| = 2}_{\hat{I}_3,\, \hat{I}_4 \text{ unmatched}}, \qquad \underbrace{|\mathrm{FN}| = 1}_{I_3 \text{ unmatched}}.$$

2. Grade with two questions

Recognition first, then mask fit:

$$\mathrm{RQ} = \frac{|\mathrm{TP}|}{|\mathrm{TP}| + \frac{1}{2}|\mathrm{FP}| + \frac{1}{2}|\mathrm{FN}|} = \frac{2}{2 + \frac{1}{2}\cdot 2 + \frac{1}{2}\cdot 1} = \frac{2}{3.5} \approx 0.571,$$
$$\mathrm{SQ} = \frac{\mathrm{IoU}(\hat{I}_1, I_1) + \mathrm{IoU}(\hat{I}_2, I_2)}{|\mathrm{TP}|} = \frac{0.82 + 0.61}{2} = 0.715.$$

3. Combine

   Multiply the two factors (Eq. 18.18),

$$\mathrm{PQ} = \mathrm{SQ} \times \mathrm{RQ} \approx 0.715 \times 0.571 \approx 0.408.$$

   The matched masks overlap fairly well on average (SQ ≈ 0.72), but two spurious predictions and a missed ground truth drag RQ down to 0.57, and the product punishes the model in both dimensions at once.

What counts as a good PQ? There is no universal cutoff, but on standard panoptic benchmarks:

- PQ < 0.3: typically weak; usually means many segments are unmatched or masks are loose.
- 0.3–0.4: typical range for hard datasets.
- 0.4–0.55: strong.
- 0.55–0.65: SOTA.
- > 0.7: near-perfect, realistic only for easy or binary cases.

Because PQ = SQ × RQ, a low PQ alone is ambiguous: report SQ and RQ separately to localise the problem (loose masks vs. poor detection).

18.3.6. Instance segmentation

Evaluation reuses the detection recipe of Sec. 18.4.1 below, with mask IoU (Eq. 18.11 applied to per-instance binary masks) replacing box IoU. The standard mask-AP metric is the COCO (Sec. 18.1) mAP averaged over IoU thresholds $\{0.5, 0.55, \ldots, 0.95\}$ (mAP@[.5:.95], Eq. 18.24) applied to masks instead of boxes.

Inference hyperparameters Two deployment-time knobs sit between the network output and the reported metric:

- *Score threshold* τ_{score}: the confidence below which predicted segments are dropped before any TP/FP/FN counting. For computing mAP, keep τ_{score} low (e.g. 0.05) so the precision–recall curve is sampled across the full operating range (Sec. 18.4.1). For visualisation or deployment, raise it (e.g. 0.5–0.7) so only confident segments are shown.
- *Mask threshold* τ_{mask}: the per-pixel probability cutoff that binarises the soft mask into $\{0, 1\}$ at inference, $\hat{M}_{ij} = \mathbb{1}[p_{ij} > \tau_{\text{mask}}]$ (the same 0.5 choice that defines the hard-mask Dice loss in Sec. 18.3.3 below). Lower it to grow masks (boosts recall, may merge instances); raise it to shrink them (boosts precision, may fragment).

Both τ_{score} and τ_{mask} change the reported mask IoU and PQ even with the network frozen, because both metrics are computed on the binarised, score-filtered prediction. Reporting numbers without disclosing the two thresholds is a common reproducibility gap.

18.3.7. Dice Loss

Definition

The baseline is per-pixel cross-entropy summed over the $H \times W$ grid, identical to a classification CE applied independently at every pixel. On imbalanced foreground (small lesions on large background, thin roads on large satellite tiles), CE alone underweights the minority class and the model collapses toward the background.

The *Dice loss* addresses this directly. In the binary case (background and foreground, $C = 1$),

$$\mathcal{L}_{\text{Dice}} \;=\; 1 \;-\; \frac{2\sum_i p_i\, y_i}{\sum_i p_i + \sum_i y_i}, \tag{18.19}$$

where $p_i \in [0, 1]$ is the predicted foreground probability and $y_i \in \{0, 1\}$ the ground-truth indicator at pixel i (with i ranging over all HW pixels). Each of the three sums has a direct interpretation:

- $\sum_i y_i = |B|$ counts the foreground pixels in the ground-truth mask,
- $\sum_i p_i$ is the soft analogue of $|A|$: the expected number of foreground pixels predicted by the model, where each pixel contributes its probability rather than a $\{0, 1\}$ vote, and
- $\sum_i p_i\, y_i$ measures their soft overlap (the predicted probability summed over the true foreground pixels).

Two variants share this formula and differ only in the input.

- The *soft Dice loss* is $\mathcal{L}_{\text{Dice}}$ as written above: the inputs are the continuous probabilities $p_i \in [0, 1]$, so the loss is differentiable in the model parameters and can drive training.
- The *hard-mask Dice loss* replaces p_i with the thresholded mask $\hat{p}_i = \mathbb{1}[p_i > 0.5] \in \{0, 1\}$ before the same formula is evaluated; the three sums then become the cardinalities $|B|$, $|A|$ and $|A \cap B|$, recovering the set-based Dice coefficient of Eq. 18.16 (Sec. 18.3.3). Training therefore uses the soft form (gradient everywhere); evaluation reports the hard form (matches the set-based coefficient that humans interpret).

Motivation (*)

Because Dice definition in 18.17 makes it a strictly increasing function of IoU, minimising $\mathcal{L}_{\text{Dice}}$ is equivalent to maximising IoU: any drop in the loss is a guaranteed gain in IoU on the same prediction, so the loss and the evaluation metric rank predictions in the same order.

The second advantage is robustness to class imbalance. Let $f = \frac{1}{HW}\sum_i y_i$ denote the *foreground fraction* (the share of pixels of foreground). For per-pixel cross-entropy with a constant-probability predictor $p_i \equiv p$ (every pixel's predicted foreground probability p_i is the same scalar p, so the loss is a function of one number instead of HW), the optimum is $p^* = f$: as $f \to 0$, CE rewards the model for predicting all-background and the foreground gradient vanishes. Dice loss has no such collapse, because its ratio is scale-invariant in the foreground set: doubling $\sum_i y_i$ and $\sum_i p_i y_i$ together leaves

$\mathcal{L}_{\text{Dice}}$ unchanged, so a tiny foreground is not automatically a small loss. Figure 18.9 shows both behaviours side by side at three foreground fractions:

- $f = 0.5$ – balanced (a roughly 50/50 split, e.g. generic binary classification). The CE optimum sits at $p^* = 0.5$ (middle dotted line in panel a); CE and soft Dice agree on direction, both descending together as $p \to 1$. The hard-mask Dice loss in panel b drops from 1 (all-background, $p \leq 0.5$) to $(1 - f)/(1 + f) = 1/3$ (all-foreground, $p > 0.5$) – a clear improvement, though Dice= 2/3 is still far from a good prediction.
- $f = 0.1$ – mildly imbalanced (typical foreground objects in natural images). The CE optimum has slid left to $p^* = 0.1$: a model that admits only 10% foreground confidence is already CE-optimal, while soft Dice keeps decreasing all the way to $p = 1$ and its gradient still asks for more foreground confidence. Hard-mask Dice plateaus at $(1 - f)/(1 + f) \approx 0.82$ on $p > 0.5$.
- $f = 0.01$ – severely imbalanced (small lesions in medical scans, thin roads in satellite tiles). The CE optimum collapses against the left axis: predicting all-background is essentially optimal and the foreground gradient is negligible. Soft Dice is still monotone-decreasing in p, but the hard-mask Dice loss only drops to ≈ 0.98 even after thresholding everything to foreground – which is why training uses the soft form rather than thresholding inside the loss.

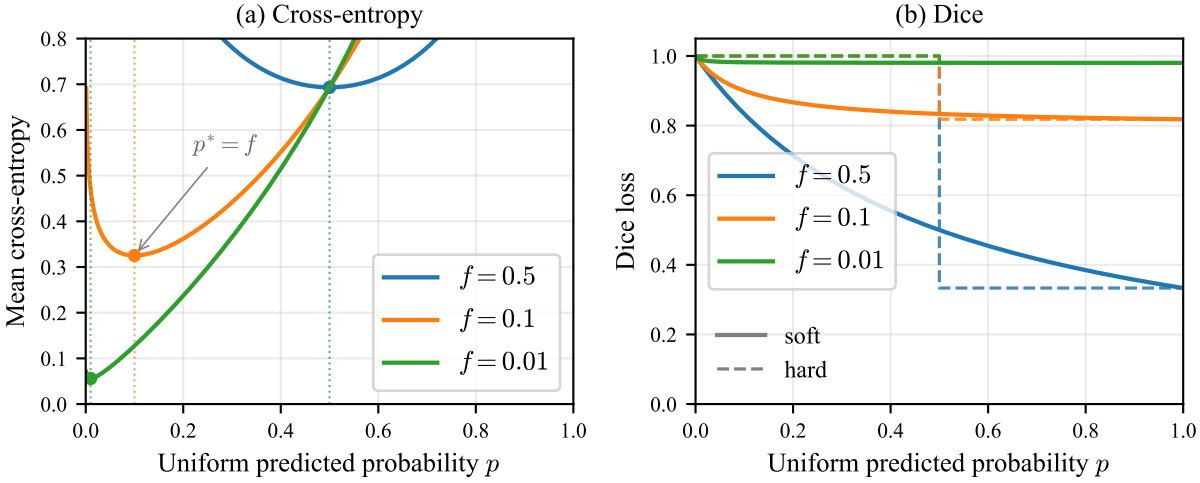


Figure 18.9.: Constant-probability predictor $p_i \equiv p$ at three foreground fractions f. (a) CE has its minimum at $p^* = f$ (dotted lines), which drifts to 0 as f shrinks. (b) Soft Dice loss (solid) decreases monotonically in p at every f, so its gradient still pulls predictions toward foreground when f is small; the hard-mask Dice loss (dashed) is a step at $p = 0.5$ and provides no usable gradient.

Example 18.5 (Dice loss on three predictors): Take the 4×4 ground-truth mask of the pixel-metrics example (Example 18.3), foreground in the top-left 2×2 block ($f = 4/16 = 0.25$). Compute the hard-mask Dice loss for three predictions: (A) all-background $\hat{M} \equiv 0$; (B) the foreground shifted one pixel to the right (TP=FP=FN= 2); (C) an exact match. For each, also report pixel accuracy (PA, Eq. 18.8, the share of correctly labelled pixels): does PA tell the degenerate predictor (A) apart from the partially-correct one (B), and does it agree with Dice on which prediction is worst?

Solution: With Dice $= \frac{2\,\text{TP}}{2\,\text{TP}+\text{FP}+\text{FN}}$ and $\mathcal{L}_{\text{Dice}} = 1 - \text{Dice}$:

(A)	$\text{TP} = 0,\ \text{FP} = 0,\ \text{FN} = 4:$	$\mathcal{L}_{\text{Dice}} = 1 - 0/4 = 1,$	$\text{PA} = 12/16 = 0.75,$
(B)	$\text{TP} = 2,\ \text{FP} = 2,\ \text{FN} = 2:$	$\mathcal{L}_{\text{Dice}} = 1 - 4/8 = 0.5,$	$\text{PA} = 12/16 = 0.75,$

(C) $\mathrm{TP} = 4,\ \mathrm{FP} = 0,\ \mathrm{FN} = 0:\qquad \mathcal{L}_{\mathrm{Dice}} = 0,\qquad\qquad \mathrm{PA} = 1.$

The degenerate all-background predictor (A) still scores PA $= 0.75$ because 12 of 16 pixels are background, yet $\mathcal{L}_{\mathrm{Dice}} = 1$ flags it as the worst possible solution. (A) and (B) are indistinguishable to pixel accuracy but separated by 0.5 in Dice loss. This is precisely the imbalance regime where CE alone leaves the model stuck at (A); adding Dice to the loss restores a useful gradient.

Multi-class case. With C foreground classes and a softmax probability vector output $\mathbf{p}_i \in [0,1]^{C+1}$ at every pixel and the ground truth is the one-hot vector $\mathbf{y}_i$. The Dice loss is computed per class against its one-vs-rest indicator and then averaged,

$$\mathcal{L}_{\mathrm{Dice}}^{\mathrm{multi}} \;=\; 1 \;-\; \frac{1}{C}\sum_{c=1}^{C} \frac{2\sum_i p_i^{(c)} y_i^{(c)}}{\sum_i p_i^{(c)} + \sum_i y_i^{(c)} + \varepsilon}, \tag{18.20}$$

where $p_i^{(c)}$ is the softmax probability of class c at pixel i, $y_i^{(c)} \in \{0,1\}$ is the one-hot ground-truth indicator, and ε is a small constant (~ 1) that prevents division by zero on images where a class is absent ($\sum_i y_i^{(c)} = 0$). The background class $c = 0$ is excluded from the sum.

Practical recipe. The standard combined loss is $\mathcal{L} = \mathcal{L}_{\mathrm{CE}} + \mathcal{L}_{\mathrm{Dice}}$ (binary or multi-class as appropriate), which keeps the per-pixel calibration of CE and the imbalance robustness of Dice.

Summary

- Evaluating segmentation on tiles that overlap training tiles. Large images (satellite scenes, histology slides, whole-slide MRI) do not fit in GPU memory and are cropped into smaller patches (tiles), often with deliberate overlap so objects on a tile boundary appear whole in some neighbouring tile. If train/test tiles are then drawn from the *same* source image (e.g. a random tile-level split), the test tiles share pixels with training tiles and the model has effectively memorised the answer for that region. The resulting IoU is dramatically inflated relative to performance on a truly unseen image. The fix is to split at the *image* level, not the tile level: all tiles from a given source image go to exactly one of train/val/test.
- Forgetting the *void* or *ignore* label (e.g. on Cityscapes, Sec. 18.1). Void pixels must be excluded from both numerator and denominator of every metric.
- Computing per-image mIoU and then averaging over images. The accepted convention is to accumulate the confusion matrix over the entire test set first and compute mIoU once.

Evaluate performance by mIoU on a held-out test set, accumulating the confusion matrix once over the whole set, and report per-class IoU on imbalanced data.

18.4. Object Detection

Goal: Predict a variable-length list of (box, class, score) tuples per image: for each detected object, an axis-aligned bounding box $b = (x_1, y_1, x_2, y_2)$, a class label $y \in \{1, \dots, C\}$, and a confidence score $s \in [0, 1]$.

Detection sits at the end of a natural progression of image-level tasks, each adding one more degree of output complexity:

- *Image classification* (Sec. 18.2)
  - Dataset provides one label per image, 1 image $\to$ 1 class.
  - Model output is a probability simplex Δ^C over the C classes.
- *Classification with localization* (currently not discussed in-depth in this chapter):
  - Dataset provides one label *and* one bounding box for a single dominant object, 1 image $\to$ 1 class + 1 box.
  - Model output is a probability simplex Δ^C and four box coordinates. The output shape is still fixed, so a single regression head bolted onto a classifier suffices.
- *Object detection* (this section)
  - Dataset provides a variable-length list of (class, box) annotations per image, 1 image $\to$ N (class + box) with N unknown in advance.
  - Model output is a *set* of (class, box, score) tuples, where the score $s \in [0, 1]$ is a per-(box, class) confidence used to rank the prediction against others on the precision-recall curve (Sec. 18.4.1).
  - Makes the output *variable-length* (an image may contain zero, one or many objects).
  - Couples a regression (box coordinates) and a classification (class + objectness), evaluated jointly.

Bounding-box principle

A *bounding box* is the smallest axis-aligned rectangle that fully encloses an object in the image plane. Two equivalent four-number parameterizations are used:

- *Corner form* $b = (x_1, y_1, x_2, y_2)$: top-left and bottom-right pixel coordinates, with $x_1 < x_2$ and $y_1 < y_2$. Convenient for IoU and matching (Sec. 18.4.1).
- *Center-size form* $b = (x_c, y_c, w, h)$: box center and width/height, with each coordinate normalized to $[0, 1]$ by dividing x_c, w by the image width and y_c, h by the image height. Normalization makes the four numbers resolution-independent: the same box parameters describe the object before and after resizing a 1920×1080 image to the 640×640 training crop, so one regression target stays valid across the multi-scale training augmentations and the variable camera resolutions seen at inference. The parameterization also decouples location (x_c, y_c) from scale (w, h).

The box carries no orientation (no rotation) and no shape information beyond the rectangle; tighter outlines require instance segmentation (Sec. 18.3). A box is considered correct, against a ground-truth box of the same class, when their overlap exceeds an IoU threshold (Sec. 18.4.1).

When to use it Detection is the right tool when you must *locate and count* multiple objects of possibly overlapping categories in a scene, and a bounding box is precise enough (otherwise use

Figure 18.10.: Object detection on a real street scene (a COCO-pretrained YOLOX detector): each yellow box is one detected vehicle. The detector localizes and counts the parked cars, returning a (box, class, score) tuple per object; the confidence scores rank the boxes on the precision–recall curve of Sec. 18.4.1. Far-away buildings, road and trees are not "thing" classes and are left undetected, which is the job of segmentation (Sec. 18.3).

instance segmentation). Counter-indications:

- A single dominant object per image → classifier (Sec. 18.2).
- Pixel-precise contours required → segmentation (Sec. 18.3).
- Open-set identification (who is this person?) → Siamese / embedding approach (Ch. 19).

18.4.1. Performance metrics

An object detector (the model being evaluated) outputs a *set* of scored boxes per image, and the metric has to collapse that set into a single comparable number. It does so through a four-stage pipeline, each stage feeding the next:

1. *Match and label*: pair each predicted box with a ground-truth box by bounding-box IoU, label it TP or FP, and count the missed ground truths as FN.

2. *Sweep*: vary the confidence cut-off to trace a precision–recall curve.
3. *Integrate*: collapse that curve into one average precision (AP) at a fixed IoU threshold.
4. *Average*: average AP over classes (mAP) and over IoU thresholds (mAP@[.5:.95]).

The subsections below walk these stages in order; non-maximum suppression is an inference-time aside (it reuses the same IoU test) inserted between Stages 1 and 2.

Box matching and TP/FP/FN labeling

Matching is done one class at a time and repeated identically for each class (Stage 4 then averages across classes).

Goal:
- Match each predicted box to a ground-truth box by IoU.
- Label every prediction TP or FP and count the missed ground truths as FN.

The detector's class-c predictions are a set of boxes $\{\hat{b}_i\}$ with confidence scores $\{s_i\}$, which must be scored against the class-c ground-truth boxes $\{b_j\}$. Matching turns these two box sets into TP, FP, and FN counts.

When does a box count as correct? Overlap is measured by the bounding-box IoU (Eq. 18.11, Fig. 18.11): a prediction can match a ground-truth box only if their IoU clears a threshold $\tau \in [0, 1]$ (Pascal VOC uses $\tau = 0.5$; a stricter τ demands tighter boxes).

The matching rule Several predictions may overlap the same object, so to keep the count unambiguous they are matched greedily in order of confidence:

1. Sort the predictions by descending confidence scores $\{s_i\}$ and walk down the list.
2. Each $\hat{b}_i$ claims the still-unclaimed ground-truth box b_j with which its $\text{IoU}(\hat{b}_i, b_j) \geq \tau$ is highest.

This sorts every box into one of three outcomes:

- *True positive* (TP): the claimed box clears $\text{IoU}(\hat{b}_i, b_j) \geq \tau$, and that ground truth leaves the pool.
- *False positive* (FP): the prediction wins no valid claim, either because it overlaps no ground truth above τ (a *misplaced box*) or because the ground truth it overlaps was already taken by a higher-scoring prediction (a *duplicate detection*).
- *False negative* (FN): a ground-truth box left unclaimed once every prediction has been processed.

Why there is no TN This reduces single-image detection to TP, FP, FN counts per class, almost the binary classification setting of Chapter 12. The missing fourth cell is TN: a true negative would be a correctly emitted "no object here" decision, but the negative space is every possible box in the image plane (infinitely many), so TN is undefined and not counted. This is why detection metrics use precision and recall (which need only TP, FP, FN) and never accuracy or specificity (which need TN).

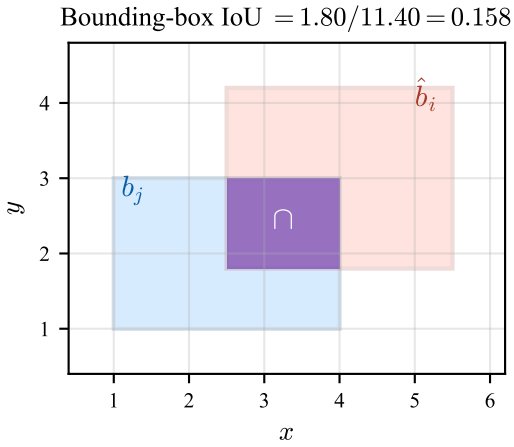


Figure 18.11.: Bounding-box IoU geometry: ground-truth box b_j (blue) and predicted box $\hat{b}_i$ (red), with their intersection shaded purple. IoU is the purple area divided by the area of the blue-red union.

What IoU threshold counts as a good match?

- $\tau = 0.5$: the lenient default; tolerates loose boxes and is what Pascal VOC (Sec. 18.1) adopted.
- $\tau = 0.75$: strict; requires tight localization.
- $\tau \in \{0.5, 0.55, \ldots, 0.95\}$: averaging the metric over this range (rather than picking one τ) is the COCO (Sec. 18.1) convention.

For a single image, IoU ≥ 0.5 usually looks visually correct; IoU ≥ 0.9 requires near-pixel-perfect box edges and is rare in practice.

Example 18.6 (Box matching at IoU $\tau = 0.5$): A small image (Fig. 18.12) contains three ground-truth boxes b_1, b_2, b_3 and four predictions $\hat{b}_1, \ldots, \hat{b}_4$ sorted by descending confidence. The matrix of pairwise IoUs (zero entries omitted) is

	score	b_1	b_2	b_3
$\hat{b}_1$	0.95	0.87	·	·
$\hat{b}_2$	0.80	0.77	·	·
$\hat{b}_3$	0.65	·	0.82	·
$\hat{b}_4$	0.40	·	·	0.83

Apply the greedy descending-confidence rule at $\tau = 0.5$: classify each prediction as TP or FP and identify any FN.

Solution: 1. Walk the score-sorted list and match

Dispatch each prediction in descending-confidence order, letting it claim its highest-IoU still-unclaimed ground truth:

rank	prediction	score	best GT	IoU	decision
1	$\hat{b}_1$	0.95	b_1	0.87	TP – claims b_1
2	$\hat{b}_2$	0.80	b_1	0.77	FP – duplicate, b_1 already claimed
3	$\hat{b}_3$	0.65	b_2	0.82	TP – claims b_2
4	$\hat{b}_4$	0.40	b_3	0.83	TP – claims b_3

2. Final counts: TP= 3, FP= 1, FN= 0 (every ground truth was claimed). The per-rank labels TP, FP, TP, TP are exactly the input of Example 18.7 below, which integrates them into AP.

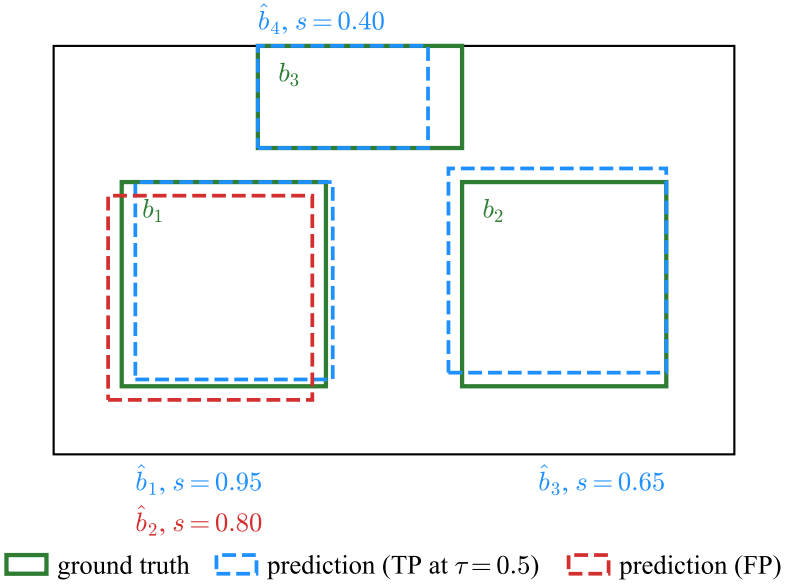


Figure 18.12.: Three ground-truth boxes (solid green) and four predicted boxes (dashed) sorted by confidence. At $\tau = 0.5$ both $\hat{b}_1$ and $\hat{b}_2$ would match b_1, but greedy descending-confidence matching keeps only the higher-scoring $\hat{b}_1$; $\hat{b}_2$ is suppressed as a duplicate and counted as FP.

Non-Maximum Suppression (NMS)

Goal: Remove duplicate predictions at inference time, so the detector returns one box per object instead of a cluster of near-identical ones.

Table 18.2 contrasts box matching and NMS.

At *inference time* the detector must return a clean box list rather than a labeled one, so it deletes the duplicates outright. *Non-maximum suppression* (NMS, used by one-stage detectors such as YOLO, Sec. 18.4.2) does this with the *same* greedy descending-confidence sweep as the matching rule, except each prediction is compared against the boxes already kept rather than against the ground truth:

1. Sort the class-c predictions by descending confidence scores $\{s_i\}$, as in Stage 1.
2. Pop the highest-scoring box $\hat{b}^*$ and keep it; remove every remaining prediction $\hat{b}_i$ with $\mathrm{IoU}(\hat{b}_i, \hat{b}^*) > \tau_{\mathrm{NMS}}$. Predictions with $\mathrm{IoU}(\hat{b}_i, \hat{b}^*) \le \tau_{\mathrm{NMS}}$ overlap $\hat{b}^*$ too little to be duplicates and stay in the pool as candidates for separate objects.

	NMS	Box matching + labeling
When	inference time (every run)	evaluation time (computing metrics)
Needs ground truth?	no	yes
IoU compares	prediction vs. prediction ($\hat{b}_i$ vs. $\hat{b}^*$)	prediction vs. ground truth ($\hat{b}_i$ vs. b_j)
Threshold	τ_{NMS}	τ
Produces	the clean predicted-box list	TP/FP/FN counts

Table 18.2.: NMS versus box matching: complementary steps, not alternatives. NMS runs at inference on the predictions alone and deduplicates them; matching runs at evaluation against the ground truth and consumes the box list NMS produced.

3. Repeat on what is left.

The NMS threshold τ_{NMS} (commonly 0.5–0.7) trades the two failure modes: too low merges genuinely adjacent objects (two pedestrians become one box), too high leaves duplicates. Figure 18.13 illustrates the effect on a two-sheep scene.

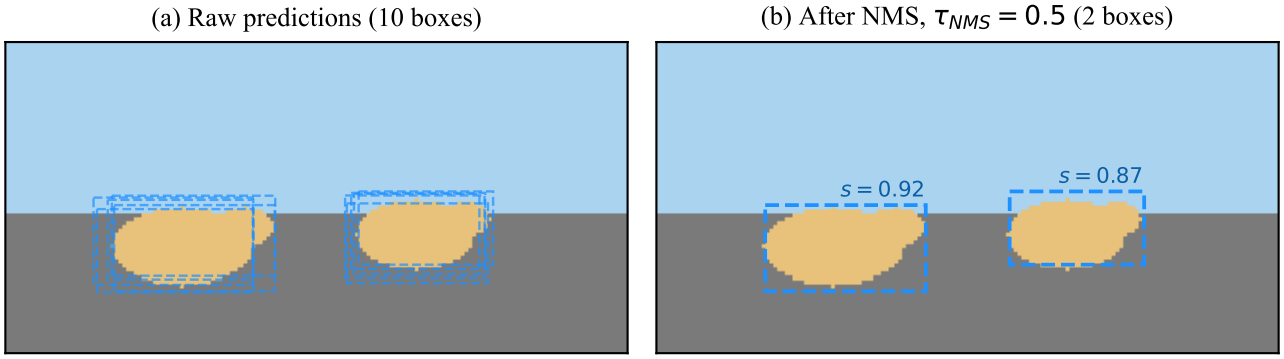


Figure 18.13.: Non-maximum suppression on a synthetic scene. (a) Dense raw predictions: each object fires from several nearby grid cells. (b) After NMS at $\tau_{\text{NMS}} = 0.5$ only the highest-scoring box per object survives.

Precision–recall curve

Goal: Turn the confidence scores into a precision–recall curve by sweeping the admission cut-off.

With every prediction labeled TP or FP and the FN count fixed (Stage 1), the only remaining freedom is how many predictions to admit. Each predicted box has a confidence s. Sweeping a cut-off s_0 from 1 down to 0 and admitting only predictions with $s \geq s_0$ produces a sequence of (precision, recall) pairs:

$$\text{Pr}(s_0) = \frac{\text{TP}(s_0)}{\text{TP}(s_0) + \text{FP}(s_0)}, \qquad \text{Re}(s_0) = \frac{\text{TP}(s_0)}{\text{TP}(s_0) + \text{FN}}. \tag{18.21}$$

Plotting Pr vs. Re gives the precision–recall curve at IoU threshold τ.

Average precision

Goal: Collapse the precision–recall curve into one number, the area under it, at a fixed IoU threshold.

A whole curve is awkward to compare across models, so Stage 3 reduces it to a single scalar. *Average Precision* (AP) at IoU threshold τ is the area under the (interpolated) precision–recall curve,

$$\mathrm{AP}_\tau \;=\; \int_0^1 \mathrm{Pr}_{\mathrm{int}}(r)\,\mathrm{d}r, \qquad \mathrm{Pr}_{\mathrm{int}}(r) \;=\; \max_{r' \geq r} \mathrm{Pr}(r'), \tag{18.22}$$

where the interpolation $\mathrm{Pr}_{\mathrm{int}}$ replaces precision at recall r by the highest precision attained at any recall $r' \geq r$, removing the zig-zag of the raw curve.

Two integration conventions coexist:

- *11-point interpolation* (Pascal VOC, Sec. 18.1, pre-2010): average $\mathrm{Pr}_{\mathrm{int}}$ over $r \in \{0, 0.1, \ldots, 1.0\}$.
- *All-point interpolation* (VOC post-2010, COCO, Sec. 18.1): integrate over all distinct recall values.

The two yield slightly different numbers and are not directly comparable.

Example 18.7 (AP from a small detection table): Reusing the four predictions from Example 18.6 (three ground-truth boxes; TP, FP, TP, TP at descending scores $0.95, 0.80, 0.65, 0.40$ at IoU threshold $\tau = 0.5$):

rank	score	TP/FP	cumulative TP	cumulative FP
1	0.95	TP	1	0
2	0.80	FP	1	1
3	0.65	TP	2	1
4	0.40	TP	3	1

Compute AP using all-point interpolation.

Solution: 1. Build the (recall, precision) points

With three ground-truth boxes (TP + FN = 3) the recall axis advances by 1/3 per TP. For each rank k, using the cumulative TP/FP counts from the table,

$$r_k = \frac{\text{cum TP}_k}{\text{TP} + \text{FN}}, \qquad p_k = \frac{\text{cum TP}_k}{\text{cum TP}_k + \text{cum FP}_k}.$$

For instance at rank $k = 3$ (score $s = 0.65$) the table gives cum $\mathrm{TP}_3 = 2$ and cum $\mathrm{FP}_3 = 1$, so

$$r_3 = \tfrac{2}{3}, \qquad p_3 = \tfrac{2}{2+1} = \tfrac{2}{3}.$$

Applying the same formulas to all four ranks gives the (recall, precision) points

$$(1/3,\ 1.00),\ (1/3,\ 1/2),\ (2/3,\ 2/3),\ (1,\ 3/4),$$

shown as red dots in Fig. 18.14, each labelled by its admission score s.

2. Interpolate

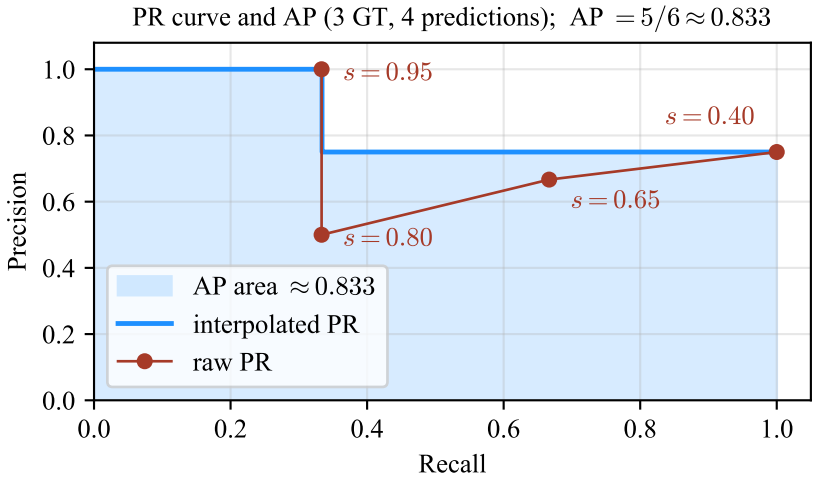


Figure 18.14.: All-point interpolated precision–recall curve for Example 18.7. Red dots: the four raw operating points $(1/3, 1), (1/3, 1/2), (2/3, 2/3), (1, 3/4)$, each annotated with the admission score s. Blue step: the interpolated precision $\mathrm{Pr}_{\mathrm{int}}(r)$ (running maximum of precision over $r' \geq r$). Shaded area: the average precision $\mathrm{AP} = 5/6 \approx 0.833$.

Interpolated precision is the running maximum of precision over recalls $r' \geq r$: the largest precision attained is 1 at $r = 1/3$ and the maximum at recall $\geq 2/3$ is $3/4$ (at $r = 1$, since $3/4 > 2/3$), so

$$\mathrm{Pr}_{\mathrm{int}}(r) = \begin{cases} 1 & 0 \leq r \leq 1/3, \\ 3/4 & 1/3 < r \leq 1, \end{cases}$$

plotted as the blue step in Fig. 18.14.

3. Integrate

   The integral, i.e. the shaded area in Fig. 18.14, evaluates to

$$\mathrm{AP} \;=\; \tfrac{1}{3} \cdot 1 + \tfrac{2}{3} \cdot \tfrac{3}{4} \;=\; \tfrac{1}{3} + \tfrac{1}{2} \;=\; \tfrac{5}{6} \approx 0.83.$$

mAP and COCO-style mAP

Goal: Average AP over classes (mAP) and over IoU thresholds (mAP@[.5:.95]) for the headline benchmark number.

Stage 3 produces one AP per class at one IoU threshold; the final stage averages those into the single number reported on a benchmark. *Mean Average Precision* (mAP) at IoU threshold τ is the arithmetic mean of per-class $\mathrm{AP}_{\tau,c}$ over the C classes:

$$\mathrm{mAP}_{\tau} \;=\; \frac{1}{C} \sum_{c=1}^{C} \mathrm{AP}_{\tau,c}, \tag{18.23}$$

where $\mathrm{AP}_{\tau,c}$ is computed exactly as in Eq. 18.22 but restricting predictions and ground-truth boxes to class c. Averaging over classes treats every class as equally important regardless of its frequency in the dataset: a model that nails the dominant class but misses every rare class will still score

poorly. (To compute one $\mathrm{AP}_{\tau,c}$, sweep the per-class predictions sorted by score, accumulate per-class TP/FP/FN, and integrate the resulting per-class PR curve.) COCO introduced a stricter metric that averages over IoU thresholds as well,

$$\mathrm{mAP}_{[.5:.95]} = \frac{1}{10} \sum_{\tau \in \{0.50, 0.55, \ldots, 0.95\}} \mathrm{mAP}_{\tau}, \tag{18.24}$$

which rewards tight localization: a model that overlaps ground-truth by 0.5 scores well at $\tau = 0.5$ but collapses at $\tau = 0.95$.

AP is per-class at one IoU threshold; *mAP* averages AP over classes; *mAP@[.5:.95]* also averages over 10 IoU thresholds and is the headline number on COCO.

18.4.2. Architecture

All detection networks share the same skeleton: a CNN backbone produces a dense *feature map*, i.e. a low-resolution $H' \times W' \times C$ tensor whose (h, w) cell carries a learned C-dimensional descriptor of the input-image patch that maps to that location; one or more prediction heads turn each feature-map cell into candidate boxes with an *objectness* (the class-agnostic probability that the box contains any object versus background) and a class distribution; NMS (Sec. 18.4.1) prunes duplicates. The per-(box, class) score s that the evaluator sees is built from these two: in one-stage detectors typically s = objectness × class probability, in two-stage detectors objectness is the score the first stage uses to keep proposals and s is the class probability output by the second-stage classifier. Families differ in how many passes it takes to get from the feature map to the final box list.

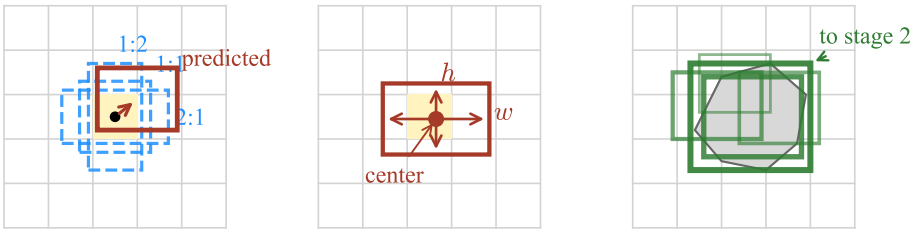
The two practically relevant families are then:

- *Two-stage detectors* (R-CNN family: Faster R-CNN, Mask R-CNN): stage 1 emits region proposals, stage 2 classifies and refines each.
- *One-stage detectors* (YOLO, SSD, RetinaNet): a single forward pass predicts boxes and classes densely over the feature grid.

A third family, *transformer-based detectors* (DETR and successors), predicts the whole set of boxes in one forward pass without anchors and without NMS, trading longer training for a simpler inference pipeline; it is increasingly competitive but not yet the practitioner default.

Three detection-specific terms used below (see Fig. 18.15):

- *Anchor box*: a fixed reference rectangle of preset size and aspect ratio, tiled at every feature-map location. The network predicts an offset and a score for each anchor instead of regressing a box from scratch.
- *Anchor-free head*: predicts box center and size (w, h) directly per feature-map cell, with no reference rectangles.
- *Region proposal*: a candidate box with an objectness score but no class yet, emitted by a first stage and fed to a second stage that classifies and refines it.



(a) (b) (c)

Figure 18.15.: Three building blocks used by detection architectures. (a) Anchor-based head: several fixed reference rectangles (blue, dashed) at the same feature-map cell, refined by a predicted offset into the final box (red). (b) Anchor-free head: the cell predicts a center point and a width/height directly. (c) Two-stage region proposals: a first stage emits several objectness-scored candidates (green, line width $\propto$ objectness) around the object; only the top ones are passed to the second-stage classifier.

R-CNN family The modern reference is *Faster R-CNN*: the backbone produces a feature map once per image; a small sub-network slides over that feature map and emits a fixed pool of anchor-based region proposals with objectness scores; the top-scoring proposals are passed to a second-stage head that outputs a class label and a refined box per proposal. *Mask R-CNN* adds a parallel head that produces a binary mask per proposal, making the same network usable for instance segmentation (Sec. 18.3). Practical profile: highest accuracy on COCO-style benchmarks, slowest per image, the default when latency is not the bottleneck or when masks or keypoints are also required.

YOLO YOLO ("You Only Look Once") treats detection as a single dense regression problem. A CNN backbone reduces the input to a coarse feature grid (e.g. 13×13); each grid cell predicts a small fixed number of candidate boxes together with an objectness score and a class distribution, all in one forward pass. Low-confidence boxes are discarded and overlapping survivors are pruned per class by NMS (Sec. 18.4.1). Modern variants predict at multiple feature-map resolutions so that small, medium and large objects share the same backbone, and recent open-source releases (Ultralytics YOLOv8/v11) ship anchor-free heads and one-line fine-tuning from COCO weights. Practical profile: real-time on commodity GPUs and edge devices, the practitioner default for deployment, accuracy now close to two-stage on modern variants.

Summary

Architecture choice in one line:

- Real-time or edge deployment → one-stage (YOLO).
- Maximum accuracy, or masks/keypoints also needed → two-stage (Mask R-CNN).
- Simpler pipeline, no NMS tuning, willing to train longer → transformer (DETR).

18.4.3. Notes

Hyperparameters

Architecture-specific hyperparameters are to be tuned (out of scope in this chapter), e.g.: for anchor-based detectors, the number, scales and aspect ratios of the anchor presets; for two-stage detectors, the number of proposals kept after stage 1; for one-stage detectors, the input resolution and the number of feature-map scales.

Reporting performance

- Reporting a single IoU threshold (mAP@.5) when comparing to COCO-era methods. Always include mAP@[.5:.95] alongside mAP@.5.
- Mixing 11-point and all-point AP across papers and quoting the numbers as comparable. They are not.

Some of hyperparameters are already introduced: the IoU threshold τ used to label predictions TP/FP (Sec. 18.4.1); the NMS IoU threshold τ_{NMS} and the confidence cut-off applied before NMS (Sec. 18.4.1).

Object size influence

Small objects are punished by IoU: a 5×5 ground-truth box with a one-pixel boundary error already loses $\sim 36\%$ of IoU. Reporting AP_{small}, $\text{AP}_{\text{medium}}$, AP_{large} separately (COCO convention) is informative whenever object size varies across the dataset.

Further Reading

Convolutional Neural Networks (Course 4 of the Deep Learning Specialization), especially C4W3 (object detection)

- Intersection Over Union (IoU): From Theory to Practice
- Intersection over Union (IoU) in Object Detection & Segmentation

Examples and Demos:

- Segment Anything and Segment Anything Model by Meta AI Demos (accessed 25-May-2026)
- Segment Anything by Hugging Face (accessed 25-May-2026)
- WPI Demo
- Web AI Demo of the library that allows you to run modern deep learning models directly in your web browser (accessed 25-May-2026)
- Conversational image segmentation with Gemini 2.5 by Google (accessed 25-May-2026)

19. Visual Representation Learning and Diagnostics

Goal: Learn image representations that transfer to tasks and classes not seen during training, and check that a trained vision model relies on real image structure.

Contents

19.1. Siamese Networks **266**
19.1.1. Definitions 266
19.1.2. When to use it 267
19.1.3. Architecture 268
19.1.4. Loss 268
19.1.5. Training 271
19.1.6. Classifier 271
19.1.7. Evaluation pitfall: leakage through pretraining 272
19.1.8. Performance metrics 273
19.1.9. Summary 274
19.2. Self-Supervised Pretraining with Masked Autoencoders **274**
19.2.1. The pretext task 274
19.2.2. Loss 275
19.2.3. Why the mask ratio is so high 276
19.2.4. Asymmetric encoder and decoder 276
19.2.5. Using the pretrained encoder 276
19.2.6. Summary 277
19.3. Comparison of CV tasks **277**
19.4. Image-Specific Sanity Checks (*) **278**
19.4.1. Visual Inspection of Worst-Performing Examples 278
19.4.2. Pixel Shuffle Test 279
19.4.3. Black-Patch Test 280
19.5. Interpreting Multimodal Models (MultiViz) (*) **280**

The previous chapter (Ch. 18) covered three tasks whose output is tied to a fixed class list: classification, segmentation and detection. This chapter covers what to do when that list is not fixed, or not available at all, and how to audit the result.

Notation As in Ch. 18, an input image is $\mathbf{X} \in \mathbb{R}^{H \times W \times C_{\text{in}}}$ with height H, width W and $C_{\text{in}} \in \{1, 3\}$ channels, and C denotes the number of target classes. The new object of this chapter is the embedding $\mathbf{z} = f_\theta(\mathbf{X}) \in \mathbb{R}^d$, a fixed-length vector that stands in for the image in every downstream computation.

The chapter covers:

- Section 19.1 *(Siamese Networks)*: learn a numeric "fingerprint" for each image so that similar images get nearby fingerprints; useful when new classes appear at test time or only a handful of examples per class are available.
- Section 19.2 *(Masked Autoencoders)*: learn the same kind of backbone from unlabeled images alone, by hiding most of each image and reconstructing it.
- Section 19.3 *(Comparison)*: the tasks of both chapters side by side, with their losses and headline metrics.
- Sections 19.4 and 19.5 *(Diagnostics)*: image-specific sanity checks, and the interpretation of models that fuse an image with another modality.

19.1. Siamese Networks

Goal: Learn a similarity-preserving embedding $f_\theta : \mathcal{X} \to \mathbb{R}^d$ such that semantically similar inputs map to nearby points and dissimilar ones to distant points, using two (or more) weight-sharing sub-networks.

A Siamese network is not a new layer type but a *training configuration*: the same backbone (CNN for images, RNN or Transformer for sequences, fully-connected for tabular data) is applied to each input in a pair (or triplet), and the loss is defined on *distances* in the resulting embedding space rather than on a classification output.

19.1.1. Definitions

Input space

The *input space* $\mathcal{X}$ is the set of all admissible raw inputs to the model. Each $\mathbf{x} \in \mathcal{X}$ is a single example (e.g. an image, a sentence, a signal segment, or a tabular row). The dimensionality of $\mathcal{X}$ is typically large and structured: a 224×224 RGB image lives in $\mathbb{R}^{224 \times 224 \times 3}$, a T-sample audio clip in $\mathbb{R}^T$.

Learned Features

In modern representation learning, *learned features* refer to any internal data representations generated by a machine learning model. Instead of relying on manual, hand-crafted feature engineering, models optimize parameters θ to automatically extract useful characteristics from a raw input space $\mathcal{X}$. These representations can take a variety of structural forms, including high-dimensional spatial grids in convolutional neural networks, categorical logit scores, or attention weight matrices in transformers.

Embedding

An *embedding* is a specific, highly structured subset of learned features. It is defined as a learned mapping $f_\theta : \mathcal{X} \to \mathbb{R}^d$ from an input space $\mathcal{X}$ (such as images, text, audio, or signals) to a low-dimensional Euclidean space $\mathbb{R}^d$, called the *embedding space* or *latent space*. The continuous vector $f_\theta(\mathbf{x}) \in \mathbb{R}^d$ is the embedding of input $\mathbf{x}$. An effective embedding satisfies two core properties:

- *Geometric*: Euclidean (or cosine) distance in $\mathbb{R}^d$ reflects task-relevant semantic similarity.
- *Compact*: $d \ll \dim(\mathcal{X})$, so downstream tasks (classification, retrieval, clustering) operate on a small fixed-size vector rather than on the raw input.

Embeddings are frequently termed *learned features*. However, learned features are not necessary embeddings. An embedding is a very specific, highly structured subset of learned features.

One-shot and few-shot learning

A classification setting in which the classes encountered at inference time are *not* those seen during training, and only a very small number of labelled examples per class is available for the new classes:

- *One-shot*: exactly one labelled example per class.
- *Few-shot* (K-shot, N-way): K labelled examples (typically $K \leq 5$) for each of N classes; a query is classified among those N.

The small set of labelled examples for the new classes is called the *support set*; the unlabelled inputs to be classified form the *query set*.

A standard solution is to learn a similarity-preserving embedding f_θ on a separate large dataset and, at inference, classify each query by some classifier.

19.1.2. When to use it

Choose a Siamese embedding when any of the following hold:

- The class set is *open*: new identities or categories appear at test time without retraining (as opposed to a *closed* set, fixed at training time).
- Only a handful of labelled examples per class is available (one-shot or few-shot, Sec. 19.1.1).
- The deployment task is verification or retrieval rather than closed-set classification.

Typical settings:

- Face verification and recognition (FaceNet, DeepFace): one model handles open-set identification, where the set of identities at test time is not known at training time.
- Signature and handwriting verification.
- One-shot and few-shot learning (Sec. 19.1.1): a handful of labelled examples per class are enough at inference, since classification reduces to nearest-neighbor search in embedding space.
- Signal similarity, e.g. matching ECG segments or audio clips by a learned distance rather than hand-crafted features.

Counter-indications. If the class set is closed and labelled data is plentiful per class, a plain softmax classifier (Sec. 18.2) is simpler, faster to train, and usually as accurate. If pixel-precise localization or per-pixel labels are needed, use segmentation (Sec. 18.3) or detection (Sec. 18.4) instead; an embedding alone does not localize.

19.1.3. Architecture

Two identical sub-networks share the same parameters θ (tied weights, Fig. 19.1). Each branch maps an input $\mathbf{x}$ to an embedding $f_\theta(\mathbf{x}) \in \mathbb{R}^d$, and a pair is compared by the embeddings distance $d(\mathbf{x}_1, \mathbf{x}_2)$ (Sec. 10.2). The most common embedding distances are L_2,

$$d(\mathbf{x}_1, \mathbf{x}_2) \;=\; \Big\| f_\theta(\mathbf{x}_1) - f_\theta(\mathbf{x}_2) \Big\|_2 . \tag{19.1}$$

and cosine distance.

Properties of the architecture:

- Weight sharing guarantees the comparison is *symmetric*: $d(\mathbf{x}_1, \mathbf{x}_2) = d(\mathbf{x}_2, \mathbf{x}_1)$.
- Halves the number of trainable parameters relative to two independent encoders.
- Both inputs are guaranteed to be processed by the *same* feature extractor, so distances are meaningful.
- The embedding is normalised (typically to the unit sphere, $\|f_\theta(\mathbf{x})\|_2 = 1$) to bound the loss and decouple it from feature magnitude.

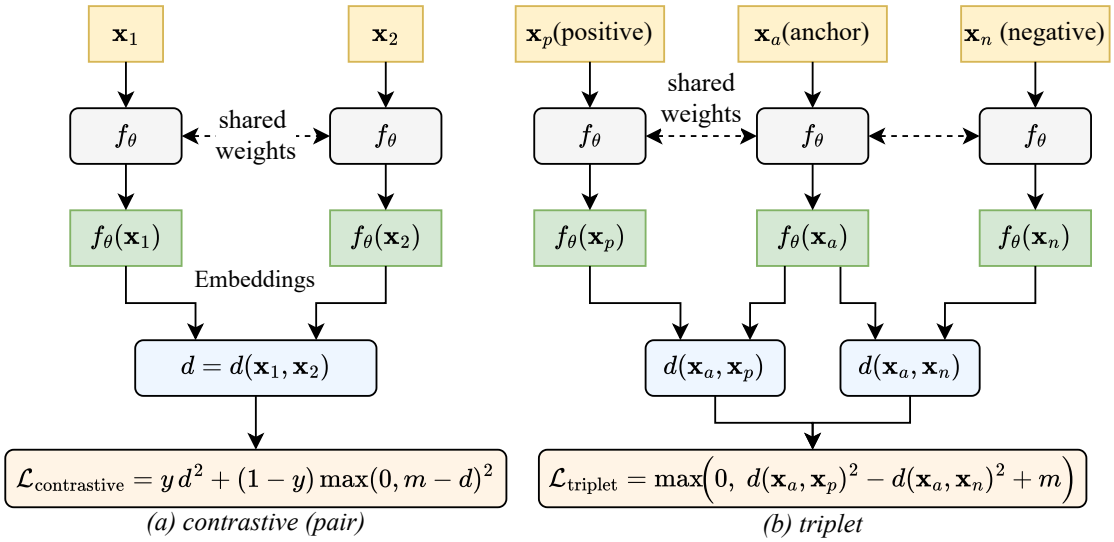


Figure 19.1.: Siamese architectures. (a) Pair / contrastive setup: two inputs share the same encoder f_θ and are compared by a distance d. (b) Triplet setup: anchor, positive and negative inputs share the same encoder, and the loss enforces $d(\mathbf{x}_a, \mathbf{x}_p) < d(\mathbf{x}_a, \mathbf{x}_n) - m$.

19.1.4. Loss

The Siamese architecture (Fig. 19.1) produces embeddings, not class probabilities, so cross-entropy is not directly applicable. The supervision is structural: rather than a class label per input, the

training set provides pairs $(\mathbf{x}_1, \mathbf{x}_2, y)$ with a binary similarity label $y \in \{0, 1\}$, or triplets $(\mathbf{x}_a, \mathbf{x}_p, \mathbf{x}_n)$ where $\mathbf{x}_p$ is known to share the anchor's class and $\mathbf{x}_n$ does not. The loss is therefore a function of the embedding distance $d(\mathbf{x}_1, \mathbf{x}_2) = \|f_\theta(\mathbf{x}_1) - f_\theta(\mathbf{x}_2)\|_2$ and of the similarity labels.

Two design choices recur in all Siamese losses:

- *Distance pulls similar pairs together and pushes dissimilar pairs apart.* Without an asymmetry between the two, the trivial solution $f_\theta \equiv$ const minimises every distance and the network learns nothing.
- *A margin $m > 0$ caps how hard a dissimilar pair is pushed.* Once the pair is far enough apart, the loss contributes no gradient; training effort is spent only on pairs that still violate the geometric goal.

The next two paragraphs introduce the two standard instantiations: contrastive loss on pairs, and triplet loss on (anchor, positive, negative) triplets.

Contrastive loss Pair-based supervision with a binary similarity label $y \in \{0, 1\}$ ($y = 1$ similar, $y = 0$ dissimilar). The idea is to enforce one geometric goal per pair:

$$d(\mathbf{x}_1, \mathbf{x}_2) \;\to\; 0 \text{ when } y = 1, \qquad d(\mathbf{x}_1, \mathbf{x}_2) \;\geq\; m \text{ when } y = 0, \tag{19.2}$$

i.e. similar pairs should collapse to the same embedding, while dissimilar pairs should be at least m apart. The two cases are turned into a single loss by penalising the squared distance for similar pairs and the hinged margin violation $\max(0,\, m - d)$ for dissimilar ones, then selecting between them with the label y:

$$\mathcal{L}_{\text{contrastive}} \;=\; y\, d^2 \;+\; (1 - y)\, \max(0,\, m - d)^2, \tag{19.3}$$

where $m > 0$ is a *margin*. Similar pairs are pulled together (the first term shrinks d), while dissimilar pairs are pushed apart only until their distance reaches m; beyond that, no gradient flows. The margin prevents the trivial collapse $f_\theta \equiv$ const.

Example 19.1: Take margin $m = 1$ and four illustrative pairs:

- Similar ($y = 1$), $d = 0.3$: $\mathcal{L} = 1 \cdot 0.3^2 + 0 = 0.09$. Small loss, weak pull-together gradient.
- Similar ($y = 1$), $d = 1.5$: $\mathcal{L} = 1 \cdot 1.5^2 + 0 = 2.25$. Strong gradient pulls the two embeddings together.
- Dissimilar ($y = 0$), $d = 0.3$ (inside the margin): $\mathcal{L} = 0 + \max(0,\, 1 - 0.3)^2 = 0.49$. The pair is pushed apart.
- Dissimilar ($y = 0$), $d = 1.5$ (beyond the margin): $\mathcal{L} = 0 + \max(0,\, 1 - 1.5)^2 = 0$. *Zero gradient*: once two dissimilar embeddings are at least m apart, the loss stops caring how much further they go.

The last row is what the margin buys: it caps the dissimilar-pair penalty so that training effort is spent on pairs that are still too close, not on already-well-separated negatives.

Triplet loss Triplet-based supervision is usually preferred in practice. Each example is a triplet $(\mathbf{x}_a, \mathbf{x}_p, \mathbf{x}_n)$ with:

- *anchor* $\mathbf{x}_a$,
- *positive* $\mathbf{x}_p$ (same class as the anchor) and

- *negative* $\mathbf{x}_n$ (different class).

The idea is to enforce one geometric inequality per triplet:

$$d(\mathbf{x}_a, \mathbf{x}_p) + m \;\leq\; d(\mathbf{x}_a, \mathbf{x}_n), \tag{19.4}$$

i.e. the anchor must be at least m closer to the positive than to the negative. Rearranging gives a non-negative *violation* $v = d(\mathbf{x}_a, \mathbf{x}_p)^2 - d(\mathbf{x}_a, \mathbf{x}_n)^2 + m$, positive exactly when the constraint is broken. Hinging it at 0 (so satisfied triplets contribute no loss) and using squared distances for a smoother gradient yields

$$\mathcal{L}_{\text{triplet}} \;=\; \max\Big(0,\; \|f(\mathbf{x}_a) - f(\mathbf{x}_p)\|_2^2 \;-\; \|f(\mathbf{x}_a) - f(\mathbf{x}_n)\|_2^2 \;+\; m\Big) \tag{19.5}$$

forces the anchor-to-positive distance to be smaller than the anchor-to-negative distance by at least the margin m. Compared with contrastive loss, the triplet form encodes a *relative* ranking: it does not require an absolute distance threshold for similar pairs, only that positives are closer than negatives.

Supervised contrastive (SupCon) loss Triplet loss uses *one* positive and *one* negative per anchor, so each gradient step depends critically on the quality of the mined triplet. *Supervised contrastive* (SupCon) loss generalises this: every same-class example in the mini-batch is treated as a positive, and every other example as a negative, so each anchor contributes a softmax over many pairs at once. SupCon pairs naturally with the $P \times K$ sampling described in *Batch construction* below and removes the need for explicit triplet mining.

Table 19.1.: Comparison of contrastive, triplet and SupCon losses for Siamese training.

	Contrastive loss	*Triplet loss*	*SupCon loss*
Supervision	Pair $(\mathbf{x}_1, \mathbf{x}_2, y)$, $y \in \{0, 1\}$	Triplet $(\mathbf{x}_a, \mathbf{x}_p, \mathbf{x}_n)$	All same-class batch items as positives, all others as negatives ($P \times K$ batch)
Geometric goal	Absolute: similar pairs $d \to 0$, dissimilar $d \geq m$	Relative: $d(\mathbf{x}_a, \mathbf{x}_p) + m \leq d(\mathbf{x}_a, \mathbf{x}_n)$	Softmax over similarities; pull all same-class together, push all others apart
Margin / temperature	Margin m caps push on dissimilar pairs beyond m	Margin m sets minimum gap between positive and negative distances	Temperature τ sharpens the softmax (no explicit margin)
Strengths	Simple pair labels; direct distance threshold for verification	Encodes relative ranking; no absolute distance scale required	Many positives and negatives per anchor; no triplet mining; strong empirical results
Weaknesses	Requires tuning absolute scale; sensitive to choice of m	Cubic triplet space; needs mining (semi-hard / batch-hard)	Needs L_2-normalized embeddings and tuned τ; benefits from large batches

19.1.5. Training

Hard negative mining The number of possible triplets grows cubically with the dataset, but most are uninformative.

- *Easy* triplets already satisfy the margin and contribute zero gradient; including them slows training without improving the model.
- *Hard* triplets violate the margin by a large amount and dominate the loss; using only the hardest ones can destabilise training (especially with label noise).
- *Semi-hard* triplets: negatives that are farther than the positive but still within the margin. This is the standard compromise.
- *Batch-hard* mining: for each anchor in a mini-batch, form a triplet from the *hardest positive* (the same-class example that is currently *farthest* in embedding space, i.e. the one the model gets most wrong) and the *hardest negative* (the different-class example that is currently *closest*). The two extrema are taken *within the batch* only, so the cost reduces to one $O(B^2)$ pairwise-distance matrix on the B batch embeddings instead of a nearest-neighbour search across the whole training set, and the gradient at every step is concentrated on the most informative triplet per anchor. It pairs naturally with $P \times K$ batch construction, which guarantees that each batch contains enough same-class and different-class candidates.

Batch construction Triplet and contrastive losses are only as good as the pairs the mini-batch actually contains: a batch of mostly singletons cannot supply positives, and a batch of mostly one class cannot supply informative negatives. The standard recipe is $P \times K$ *sampling*: each mini-batch is built from P classes drawn uniformly, with K examples per class.

- Guarantees $K - 1$ in-batch positives and $(P - 1)\,K$ in-batch negatives for every anchor, so batch-hard mining always has candidates.
- Typical values: $P \in [8, 32]$, $K \in [4, 8]$, giving batches of 64–256 samples; small K wastes positives, large K wastes negatives.
- Classes are usually sampled *uniformly* rather than proportionally to frequency, so head classes do not dominate the loss.
- Combined naturally with batch-hard mining: for each anchor, pick the farthest same-class example and the closest different-class example *within the batch*.

19.1.6. Classifier

The trained Siamese network produces embeddings, not class probabilities; classification is built on top of them in one of four ways.

- *Verification* (open-set, "are these two inputs from the same class?"; e.g. face unlock, signature check): map a *pair* to a binary accept/reject decision $\widehat{y} = \mathbf{1}[\, d(\mathbf{x}_1, \mathbf{x}_2) \le \tau \,] \in \{0, 1\}$, accepting the pair as same-class when their embedding distance falls below a threshold τ tuned on a held-out validation set of labelled same/different pairs. Because the decision depends only on the distance, identities unseen during training can be verified at test time without retraining.
- *Identification / retrieval* (open-set, "which class is this?"): *single input* $\to$ class label. Precompute a *gallery* of embeddings $\{f_\theta(\mathbf{x}_i)\}$ with known labels $y_i \in \{1, \dots, C\}$ and classify a query $\mathbf{x}$ by k-NN in embedding space, $\widehat{y}(\mathbf{x}) \in \{1, \dots, C\}$. Adding a new class only requires inserting its embeddings into the gallery.

- *Embeddings as features for an ML classifier* (closed-set): freeze f_θ and treat $\mathbf{z} = f_\theta(\mathbf{x}) \in \mathbb{R}^d$ as a fixed-length feature vector. Fit a standard supervised classifier, e.g. multinomial logistic regression or a (kernel) SVM, on the labelled set $\{(\mathbf{z}_i, y_i)\}$ for the C target classes. Because the embedding is low-dimensional and (typically) L_2-normalised, even a linear classifier is usually sufficient and trains in seconds.
- *Closed-set classification* (Siamese as pre-training): once f_θ has learned a good representation, freeze it and fit a small classifier head $g_\phi : \mathbb{R}^d \to \Delta^{C-1}$ (typically a single linear layer with softmax) on labelled data for the C target classes. This decouples representation learning from classification and is the standard recipe for using a Siamese / contrastive backbone in supervised downstream tasks.

Table 19.2.: Deployment modes for a trained Siamese embedding f_θ.

	Verification	*Identification (k-NN)*	*ML classifier on embeddings*	*Linear softmax head*
Question	Same class? (pair)	Which class? (single)	Which class? (single)	Which class? (single)
Setting	Open-set	Open-set	Closed-set	Closed-set
Decision rule	$d(\mathbf{x}_1, \mathbf{x}_2) \leq \tau$	k-NN in gallery	ML classifier on $\mathbf{z}$	softmax($\mathbf{Wz} + \mathbf{b}$)
Parametric?	No (threshold only)	No (lazy, gallery)	Yes	Yes
Training cost	Tune τ	None (store gallery)	Fast (convex fit)	Fast (one layer)
Add new class	Free	Insert into gallery	Retrain classifier	Retrain head
Inference cost	$O(1)$ per pair	Grows with gallery	$O(Cd)$	$O(Cd)$
Typical use	Face verification	Few-shot retrieval	Frozen-embedding ML pipeline	Supervised downstream task

19.1.7. Evaluation pitfall: leakage through pretraining

Every deployment mode above is built in two phases: pretrain f_θ, then freeze it and fit something on the embeddings. The train/test split is normally applied in the second phase, when the classifier is fitted. Phase 1 is left to run over whatever data is at hand, which is usually the whole dataset. When that happens the test examples are inside the embedding's own training set, and the score reported at the end is optimistic (Fig. 19.2).

Leakage through pretraining

- *The split is applied too late.* Fitting the probe on a training split says nothing about phase 1. If f_θ was pretrained on the full dataset, every test example helped shape the embedding space it is later scored in.
- *Contrastive pretraining is supervised.* Contrastive, triplet and SupCon losses all read labels, because they have to know which pairs are positive. What leaks is therefore label information and not merely the input distribution, which is what separates this case from ordinary unsupervised pretraining on unlabelled data.
- *The rule: split first, then pretrain.* Everything downstream of the split, phase 1 included,

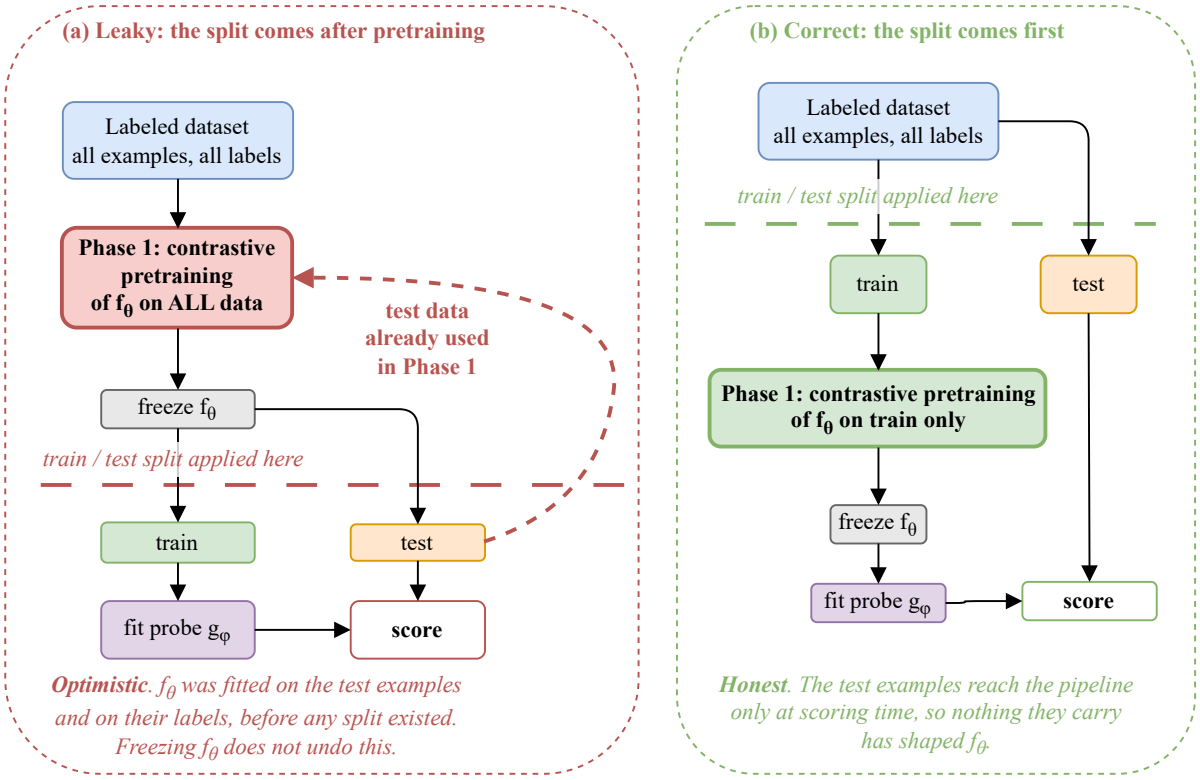


Figure 19.2.: Where the train/test split has to go. (a) The split is applied when the probe is fitted, so the test examples have already been seen by phase 1, together with their labels. (b) The split is applied first, so phase 1 sees the training portion only and the test examples enter at scoring time. The only thing that moves between the two panels is the dashed line.

sees the training portion only.

19.1.8. Performance metrics

For the verification deployment mode, a binary same/different decision is driven by a threshold τ on the embedding distance $d(\mathbf{x}_1, \mathbf{x}_2)$. Writing

$$\mathrm{TAR}(\tau) = \Pr\Big(d(\mathbf{x}_1,\mathbf{x}_2) \le \tau \,\Big|\, y = 1\Big), \qquad \mathrm{FAR}(\tau) = \Pr\Big(d(\mathbf{x}_1,\mathbf{x}_2) \le \tau \,\Big|\, y = 0\Big), \tag{19.6}$$

the true accept rate (TAR = TPR) and false accept rate (FAR = FPR) trace a ROC curve as τ varies, summarised by AUC (Sec. 12.5 of Chapter 12). Two operating-point summaries dominate biometric reporting:

- *Equal Error Rate* (EER): the point where FAR = FRR (false reject rate = 1 − TAR), the intersection of the ROC with the anti-diagonal TPR = 1 − FPR. Single-number summary.
- *TAR@FAR*= 10^{-k}: true-accept rate at a fixed, very small FAR (typically 10^{-3}, 10^{-4} or 10^{-6}). The standard reporting on face-verification benchmarks (LFW, MegaFace, IJB-C) because deployments target a low false-accept budget.

Retrieval and frozen-embedding deployment modes are evaluated by Rank-k / CMC and Recall@k for retrieval, and by a linear probe or k-NN accuracy for downstream classification on frozen embeddings.

19.1.9. Summary

- *Identity leakage*: in open-set verification, no identity in the test pair set may appear in the training set, otherwise the embedding has merely memorised it.
- *Leakage through pretraining* (Sec. 19.1.7): applying the train/test split only when the probe is fitted leaves phase 1 running on the whole dataset, so the frozen embedding has already seen the test examples and their labels. Split first, then pretrain.
- *Threshold tuned on the test set*: τ must be tuned on a held-out validation set of same/different pairs, never on the test pairs.
- *Reporting only AUC* on biometrics. AUC is dominated by the easy region of the ROC; deployments care about the very-low-FAR regime, which AUC barely sees. Always pair AUC with TAR@FAR$= 10^{-k}$.

Train the embedding f_θ with triplet loss and semi-hard or batch-hard mining; choose the deployment mode (verification, k-NN, ML classifier on embeddings, or linear head) afterwards based on whether the class set is open or closed.

19.2. Self-Supervised Pretraining with Masked Autoencoders

Goal: Learn a reusable image backbone from *unlabeled* images, by hiding most of each image and training the network to reconstruct what was hidden.

Every recipe so far in this chapter reads labels. The contrastive, triplet and SupCon losses of Sec. 19.1 all have to know which pairs share a class, which is exactly why the leakage box of Sec. 19.1.7 classifies contrastive pretraining as supervised. In practice that requirement, and not the network, is the bottleneck: unlabeled images are abundant, annotated ones are not.

Self-supervised learning (SSL) removes the requirement by manufacturing the supervision from the input itself. The training signal comes from a *pretext task*: an artificial prediction problem whose target is produced by a deterministic corruption of the input, so that every unlabeled image supplies its own target for free. The pretext task is never the goal; it is a device for forcing the encoder to learn something transferable, after which it is thrown away and the encoder is reused. The *masked autoencoder* (MAE) is the pretext task that dominates image pretraining, and its rule is one line: hide most of the image, reconstruct it.

19.2.1. The pretext task

Masked autoencoder (MAE): Cut the input image $\mathbf{X} \in \mathbb{R}^{H\times W\times C_{\text{in}}}$ into $N = (H/\ell)(W/\ell)$ non-overlapping square *patches* $\mathbf{x}_1, \dots, \mathbf{x}_N$ of side ℓ pixels, each flattened to $\mathbf{x}_n \in \mathbb{R}^D$ with $D = \ell^2 C_{\text{in}}$.

Draw a random subset $\mathcal{M} \subset \{1, \dots, N\}$ of size $|\mathcal{M}| = \rho N$, where $\rho \in (0, 1)$ is the *mask ratio*, and hide those patches. Then

- the *encoder* f_θ processes the $(1-\rho)N$ *visible* patches only, and never sees a masked one;
- the *decoder* g_ϕ receives the encoded visible patches together with a shared learned *mask token* placed at each masked position, and outputs a reconstruction $\hat{\mathbf{x}}_n \in \mathbb{R}^D$ for every $n \in \mathcal{M}$.

Figure 19.3 shows the pipeline. The patch decomposition is not an extra assumption: it is how a vision transformer (the ViT rows of Table 18.1) already reads an image, one token per patch, so masking a patch simply means dropping a token.

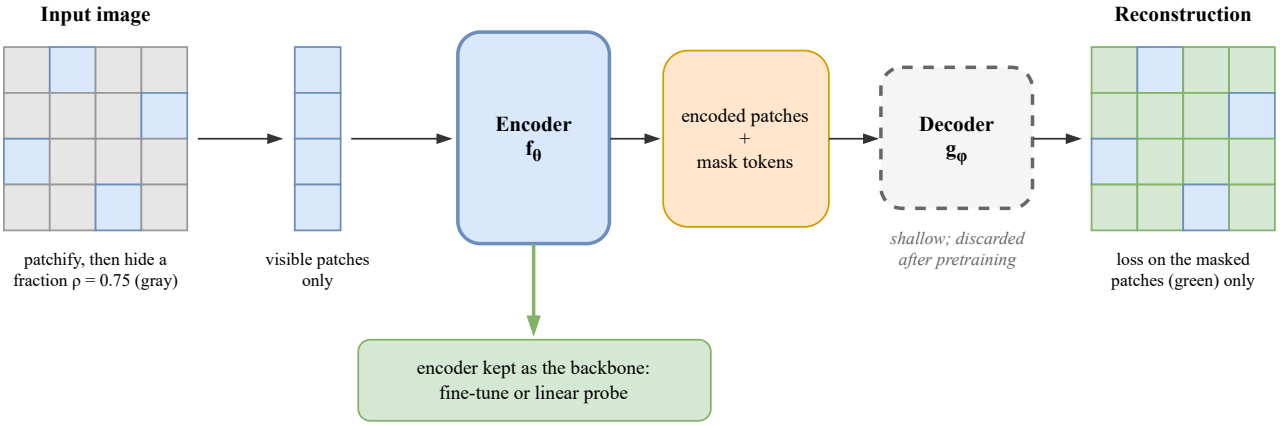


Figure 19.3.: Masked-autoencoder pretraining. The image is cut into patches and a fraction ρ of them (gray) is hidden. The encoder f_θ sees the visible patches only; the decoder g_ϕ receives the encoded patches plus a mask token at each hidden position and reconstructs the missing pixels, which is where the loss (19.7) is evaluated. After pretraining the decoder is discarded (dashed) and the encoder is reused as the backbone of a downstream task.

19.2.2. Loss

The loss is the mean squared error over the *masked* patches only,

$$\mathcal{L}_{\text{MAE}} = \frac{1}{|\mathcal{M}|\, D} \sum_{n \in \mathcal{M}} \left\| \hat{\mathbf{x}}_n - \mathbf{x}_n \right\|_2^2, \tag{19.7}$$

that is, the average squared pixel error per masked patch, normalized per pixel by D so the number is comparable across patch sizes. Two conventions inside (19.7) carry most of its behavior:

- *The visible patches are excluded from the sum.* Were they included, the network could drive the loss down by copying the patches it can already see, and that term, being trivially solvable, would dominate the gradient while teaching nothing about the hidden content.
- *The target patch is normalized before the comparison.* Each $\mathbf{x}_n$ has its own mean subtracted and is divided by its own standard deviation (biased, over the D values of the patch). This removes patch brightness and contrast, which are predictable from the surrounding patches and would otherwise consume most of the loss budget, and leaves the local structure as the thing being scored.

19.2.3. Why the mask ratio is so high

The standard setting is $\rho = 0.75$: three quarters of the patches are removed. This is far more aggressive than the equivalent recipe for text, where masking roughly 15 % of the tokens is enough, and the reason is redundancy. Neighboring pixels are strongly correlated, so at a low mask ratio a hidden patch is recoverable by interpolating from the patches that touch it, a local operation that requires no notion of what the image contains. Raising ρ removes that shortcut: at $\rho = 0.75$ the surviving patches are spatially sparse, the nearest visible neighbor of a hole is typically several patches away, and filling the hole requires the global structure of the object.

The mask ratio ρ and the patch side ℓ are the two hyperparameters of the pretext task, and they interact: a small ℓ makes each patch more predictable from its neighbors and therefore calls for a larger ρ, while a large ℓ makes the reconstruction harder and the token sequence shorter, so pretraining is cheaper but coarser.

19.2.4. Asymmetric encoder and decoder

The design has one more property worth naming, because it is what makes the recipe affordable. The encoder runs on the visible patches only, so at $\rho = 0.75$ it processes a quarter of the tokens; the mask tokens enter at the decoder, not at the encoder. The decoder is deliberately shallow, a few blocks against the encoder's dozens, because it is discarded after pretraining and any capacity spent there is capacity not spent on the part that is kept.

> The asymmetry is a compute argument, not an accuracy argument: for a fixed budget, an encoder that sees 25 % of the tokens can be trained for several times as many epochs as one that sees all of them. Pretraining epochs are the currency SSL spends, so this is what makes a large mask ratio attractive rather than merely tolerable.

19.2.5. Using the pretrained encoder

When pretraining ends, g_ϕ and the mask token are discarded and f_θ is the backbone. From there the options are the ones already tabulated for the Siamese embedding (Table 19.2, Sec. 19.1.6), since both phases produce the same object, a frozen map from image to vector:

- *Fine-tuning*: unfreeze f_θ and train it together with a task head on the labelled set. This is the default for MAE and where its accuracy advantage appears.
- *Linear probe*: freeze f_θ and fit a single linear softmax layer on $\mathbf{z} = f_\theta(\mathbf{X})$.
- *k-NN on frozen embeddings*: no fitting at all, used as a cheap diagnostic of representation quality.

Judge the backbone by the mode you will deploy

> MAE and contrastive pretraining do not rank the same way under the two evaluation modes: a masked autoencoder typically fine-tunes better and linear-probes worse than a contrastive backbone of comparable size. Nothing in (19.7) asks for linearly separable features, whereas a contrastive loss optimizes distances directly and therefore hands the linear probe a representation already shaped for it. A linear probe accordingly understates an MAE backbone.

Compare pretraining recipes under the deployment mode you actually intend to use.

Table 19.3 places the two pretraining families side by side.

Table 19.3.: Masked-autoencoder pretraining against the contrastive / Siamese pretraining of Sec. 19.1.

	Masked autoencoder	*Contrastive / Siamese*
Supervision	Pixels of the masked patches, produced by the masking itself	Pair or triplet labels, which require class information
Labels needed	None	Yes, to decide which pairs are positive
What leaks into pretraining	Input distribution only	Input distribution *and* label information (Sec. 19.1.7)
Augmentations	Minimal; the masking is the corruption	Central; positives are augmented views, and the augmentation set defines the invariances
Batch size	Ordinary	Large batches (or mining) needed to supply informative negatives
Strong under	Fine-tuning	Linear probe and k-NN retrieval
Backbone	Patch-token (ViT-style)	Any encoder, CNN included

19.2.6. Summary

- *Split first, then pretrain* (Sec. 19.1.7). The rule survives the move to SSL. What leaks here is weaker, the input distribution rather than the labels, but a backbone pretrained on the full dataset has still seen every test image before it is scored on it.
- *The reconstruction is not the objective.* A low $\mathcal{L}_{\text{MAE}}$, or a reconstruction that looks convincing, is not evidence of a useful backbone; the pretext task is a means. Compare pretraining runs by the downstream metric, never by the pretraining loss.
- *The recipe needs scale.* MAE assumes a patch-token backbone and a large unlabeled corpus. On a few thousand images, pretraining on that same small set buys little over a supervised baseline or an off-the-shelf pretrained backbone.

19.3. Comparison of CV tasks

Table 19.4 summarizes the computer-vision tasks covered in this chapter and the previous one: image classification (Sec. 18.2), image segmentation (Sec. 18.3), object detection (Sec. 18.4), Siamese embeddings (Sec. 19.1), and masked-autoencoder pretraining (Sec. 19.2). The right column captures the practical question each one is designed to answer.

Table 19.5 consolidates the evaluation metrics of both chapters, grouped by the task that produces them. Classification primitives (accuracy, F_1, AUC) are deferred to Chapter 12 and reused by several modes here.

Table 19.4.: Computer-vision tasks side by side.

Task	Output	Loss family	Headline metric	When to use
Image classification	one class label per image	cross-entropy (optionally focal)	Top-1 / Top-k, macro-F_1	one dominant object, closed set
Image segmentation	per-pixel class map $\in \{0, \dots, C\}^{H \times W}$	CE + Dice	mIoU, Dice	masks needed, boundaries matter
Object detection	list of (b, y, s) tuples	box regression + objectness/class CE	mAP@[.5:.95]	locate and count objects, real-time
Siamese embedding	embedding $\mathbf{z} \in \mathbb{R}^d$	contrastive / triplet / SupCon	verification AUC, top-k retrieval	open set, few-shot, similarity search
Masked autoencoder	reusable backbone f_θ	masked-patch MSE	fine-tuned / linear-probe accuracy	unlabeled images plentiful, labels scarce

19.4. Image-Specific Sanity Checks (*)

Goal: Validate that image-based classifiers rely on meaningful spatial or global structure rather than localized artifacts or per-pixel statistics.

The sanity checks in this section (beyond these in Ch. 13) are specific to classifiers operating on image data (raw pixels or spatial feature maps). They test whether the model exploits spatial structure—edges, textures, shapes—or relies on superficial statistics that happen to correlate with the labels.

19.4.1. Visual Inspection of Worst-Performing Examples

Worst-case visual inspection: Rank the held-out examples by a per-example performance score (loss, or $1 - p_y$ for classification, $1 - \text{IoU}$ for segmentation, $1 - \text{IoU}$ between best-matched boxes for detection), select the worst k (typically $k = 20$ to 50), and display them in a grid annotated with the true label, the predicted label, and the model's confidence.

Aggregate metrics (Sec. 18.2.1 for classification, Sec. 18.3.1 for segmentation, Sec. 18.4.1 for detection) summarize performance into a single number and hide *which* images the model gets wrong. Manually inspecting a few dozen worst cases regularly surfaces failure modes that no single-number metric can reveal, and it is among the simplest diagnostics available: it requires no retraining and no additional labels.

Patterns to look for in the worst-case grid:

- *Label noise*: the ground-truth annotation is wrong or ambiguous, and the model's prediction is in fact reasonable. Common on crowd-sourced or hastily relabeled datasets.
- *Out-of-distribution inputs*: corrupt files, extreme rotation or cropping, drastically different lighting or resolution from the training set.
- *Confounding artifacts*: timestamps, watermarks, scanner markers, ruler overlays, or hospital identifiers that correlate with the label in training but not at deployment.

Table 19.5.: Headline performance metrics of the two computer-vision chapters, grouped by task.

Task	Metric	What it measures	Reference
Image classification	Top-1, Top-k, macro-F_1, AUC	Per-image label accuracy and ranking quality.	Sec. 18.2.1
Semantic segmentation	mIoU (Eq. 18.13), per-class IoU, fwIoU	Pixel overlap, class-averaged or frequency-weighted.	Sec. 18.3.2
Medical segmentation	Dice (Eq. 18.16), boundary IoU/F	Same as IoU on a different scale; boundary metrics for thin contours.	Sec. 18.3.3
Instance / panoptic seg.	mask-AP, Panoptic Quality (Eq. 18.18)	Detection-style mask matching; PQ combines segmentation and recognition quality.	Sec. 18.3.5
Object detection	AP@τ (Eq. 18.22), mAP@.5, mAP@[.5:.95] (Eq. 18.24)	Area under PR curve at one or many IoU thresholds.	Sec. 18.4.1
Detection size bands	$\text{AP}_{\text{small/medium/large}}$	AP restricted to GT boxes in a size band; diagnoses small-object failure.	Sec. 18.4.1
Siamese verification	AUC, EER, TAR@FAR= 10^{-k}	Same/different decision on pairs at a distance threshold; biometrics low-FAR operating point.	Sec. 19.1.8
Siamese retrieval	Rank-k / CMC, Recall@k, mAP	Ranked-gallery quality for an open-set query.	Sec. 19.1.8
Siamese as features	Linear probe acc., k-NN acc.	Downstream usefulness of frozen embeddings.	Sec. 19.1.8
Self-supervised pretraining	Fine-tuned acc., linear probe acc.	Downstream usefulness of the backbone; the pretraining loss is not a quality measure.	Sec. 19.2

- *Systematic class confusion*: one class is consistently mistaken for another (e.g. visually similar dog breeds, two adjacent organ classes), pointing to a missing distinguishing feature or insufficient training examples for the confused pair.

Model vs annotation

The worst-case set is usually a mix of:

- model failure and
- annotation failure.

Separate the two before reacting: relabeling or removing mislabeled examples often improves more than another round of hyperparameter tuning.

19.4.2. Pixel Shuffle Test

Pixel shuffle test: Given a dataset of N images, randomly permute all pixel locations within each

image independently, producing a shuffled dataset $\mathbf{X}_{\text{shuf}}$. Re-extract features from $\mathbf{X}_{\text{shuf}}$ and repeat the cross-validation procedure.

Shuffling destroys all spatial structure (edges, textures, object shapes) while preserving the per-image pixel histogram (mean intensity, variance, and higher-order marginal statistics remain identical). A classifier that genuinely relies on spatial patterns will see its accuracy drop to chance level on $\mathbf{X}_{\text{shuf}}$. Conversely, if the accuracy remains high after shuffling, the classifier exploits per-pixel statistics rather than spatial structure, and the original cross-validation result is unreliable.

The pixel shuffle test is specific to classifiers that operate on raw pixel features or spatial feature maps (e.g., convolutional networks). It does not apply when hand-crafted, non-spatial features are extracted before classification.

19.4.3. Black-Patch Test

Black-patch test: Occlude a randomly positioned square region of each image with a black (zero or other constant-valued) patch of side length s pixels. Re-extract features and repeat cross-validation. Repeat for progressively larger patch sizes (e.g., $s \in \{16, 32, 56, 112\}$) to obtain an accuracy-versus-occlusion curve.

The black-patch test probes whether the classifier uses distributed spatial information or relies on a localized region. Three representative outcomes:

- Gradual, monotonic decline in accuracy as s increases—the classifier uses information spread across the image, which is the expected behavior for a well-trained model.
- Sharp drop at a specific patch size—the classifier depends on a localized region. This may indicate a genuine region of interest, but could also signal reliance on a confounding artifact (e.g., a timestamp, label overlay, or acquisition marker embedded in a fixed image location).
- Stable accuracy across all patch sizes—the classifier is largely insensitive to spatial content, raising concern that it exploits non-image-based confounders (e.g., differences in file encoding or image dimensions between classes).

When interpreting the black-patch test, consider the patch area relative to the total image area. A 112×112 patch occludes roughly 44% of a 168×168 image but only 5% of a 512×512 image. Report patch sizes as fractions of the image dimensions to allow meaningful comparison across datasets.

19.5. Interpreting Multimodal Models (MultiViz) (*)

Goal: Explain a prediction from a model that fuses several modalities (e.g., an image and a text question) by attributing it to the contributing inputs, their interactions, and the internal representations.

The diagnostics above probe a classifier that sees a single image. Many modern vision models are instead *multimodal*: they fuse an image with another modality, such as the text of a question in visual question answering (VQA), or accompanying tabular and clinical data. For such a model the

interpretation question is richer: not only *where* in one image the model looks, but *which* modality drives the answer and *how* the modalities interact. MultiViz is a framework that organizes this analysis into four stages.

It is worth separating this task from the Classifier Comparison chapter (Ch. 14). Those methods *compare two* models from their outputs alone (hard labels or predicted probabilities on a shared test set) to decide which is better or whether to fuse them. MultiViz instead *interprets a single* fused model, one prediction at a time, by opening its inputs and internal features. The two live on orthogonal axes; the closest relatives of MultiViz are the single-model, input-level diagnostics of this chapter, such as the worst-case inspection and the black-patch/occlusion test (Sec. 19.4.3).

The four MultiViz stages are:

- *Unimodal importance:* attribute the prediction to individual inputs within each modality, using per-modality saliency (gradient-based or LIME). The single-image analogue is the black-patch/occlusion test (Sec. 19.4.3), which localizes the image region a classifier depends on.
- *Cross-modal interactions:* identify which pairs of inputs *across* modalities jointly drive the prediction, for example a specific question word tied to a specific image region. This stage has no single-modality counterpart, and it is often where a multimodal model succeeds or fails.
- *Multimodal representations:* inspect the fused internal features, in the same spirit as the embedding analysis used for Siamese networks (Sec. 19.1) and the low-dimensional projections of Ch. 11 (t-SNE, UMAP), to see what the representation has encoded.
- *Prediction analysis:* fit a sparse, local surrogate that relates a few representation units to the final output, closing the loop from inputs, through interactions and representations, to the prediction.

The visualizations produced at each stage are hypotheses about the model's behavior, not proofs. A highlighted region or a clean-looking embedding cluster can still be a spurious artifact, so the same cautions as the sanity checks above apply: confirm an interpretation with a controlled intervention (e.g., occluding the highlighted region, or perturbing the implicated modality) before trusting it.

Further Reading

- Convolutional Neural Networks (Course 4 of the Deep Learning Specialization), especially C4W4 (Siamese Network)
- Kaiming He et al., Masked Autoencoders Are Scalable Vision Learners, CVPR 2022 (Sec. 19.2).
- Paul Pu Liang et al., MultiViz: Towards Visualizing and Understanding Multimodal Models, ICLR 2023 (Sec. 19.5).

Part II.

Linear Least Squares in Signals and Signal Modeling

Preface

In recent years, the convergence of machine learning (ML) and signal processing (SP) has gathered growing attention in engineering education. Students are often introduced to ML principles at an early stage, yet many advanced SP topics, ranging from linear systems and time-frequency analysis to probabilistic modeling, traditionally require multiple specialized courses [8]. Although these SP methods yield comprehensive performance insights and rigorous conclusions, teaching them can be both time-consuming and demanding.

A key bridge between basic ML concepts and advanced SP techniques is the *least squares* (LS) method. LS is grounded in a simple and intuitive idea: minimizing the sum of squared errors. While direct LS computations may be $\mathcal{O}(N^3)$, and thus less efficient than typical SP methods ($\mathcal{O}(N \log N)$ to $\mathcal{O}(N^2)$), the LS perspective fosters a simpler, data-driven understanding of fundamental SP tasks. For example, the estimation of sinusoidal signal parameters in noise can be introduced by viewing it purely as a regression problem, bypassing the need for more involved probabilistic analyses. Likewise, the discrete Fourier transform (DFT) can be reframed as an extension of sinusoidal parameter estimation, illustrating SP principles with real arithmetic alone.

An LS-centric viewpoint aligns well with the foundational prerequisites of many ML courses and can be integrated at an early stage of engineering or data science programs. It offers an accessible path for teaching core SP ideas to engineering students who might lack extensive mathematical or probabilistic training. Although the underlying techniques are not new, this data-driven, regression-based interpretation may be more intuitive for those already familiar with basic ML concepts, enabling them to explore SP topics with minimal additional theoretical overhead.

20. Sinusoidal Signal Analysis

Goal: This chapter introduces the fundamental concepts and methods for analyzing and estimating (learning) parameters of a discrete-time sinusoidal signal observed in additive noise.

Contents

20.1. Signal Preliminaries **285**
20.1.1. Cosine Signal 285
20.1.2. Energy & Power 286
20.1.3. Noise 287
20.1.4. Basic Signal Analysis 287
20.2. Amplitude estimation **288**
20.2.1. Model 288
20.2.2. Signal-to-Noise-Ratio 289
20.2.3. Theoretical performance bounds (*) 290
20.2.4. Monte-Carlo Simulation 291
20.3. Amplitude and phase estimation **291**
20.3.1. Model 292
20.3.2. Theoretical performance bounds (*) 293
20.4. Frequency estimation **293**
20.4.1. Model 294
20.4.2. Periodogram 295
20.4.3. Theoretical performance bounds (*) 296
20.5. Harmonic Signal Analysis **296**
20.5.1. Model 296
20.5.2. Theoretical performance bounds (*) 298
20.6. Discrete Fourier Transform (DFT) **300**
20.6.1. Definition 300
20.6.2. Properties 300
20.6.3. Power Spectral Density 301
20.6.4. Advanced notes (*) 301
20.6.5. Spectral Spreading and Leakage (*) 305
20.7. Short-Time Fourier Transform (STFT) **306**
20.7.1. Definition 307
20.7.2. Spectrogram 307

20.7.3. Time-frequency trade-off 307
20.7.4. Welch's Periodogram 308
20.8. Quasi-periodic signals 310
20.9. Summary 311
20.10Exercise 315

20.1. Signal Preliminaries

20.1.1. Cosine Signal

A general continuous-time cosine signal can be written as

$$\begin{aligned} x(t) &= A\cos(2\pi F_0 t + \theta), \\ &= A\cos(\Omega_0 t + \theta), \end{aligned} \tag{20.1}$$

where

- $A > 0$ is the amplitude,
- $-\pi < \theta \leq \pi$ or $0 \leq \theta < 2\pi$ is the phase,
- F_0 is the frequency in Hz,
- $\Omega_0 = 2\pi F_0$ is the radial frequency in rad/sec,

For the further analysis, we use the sampled version $x[n]$ of the continuous-time signal $x(t)$, sampled with frequency $F_s = 1/T$,

$$\begin{aligned} x[n] &= x(nT) \\ &= A\cos(\omega_0 n + \theta) \quad n = 0, \ldots, L-1, \end{aligned} \tag{20.2}$$

where

$$\omega_0 = 2\pi F_0 T = 2\pi \frac{F_0}{F_s} \tag{20.3}$$

is the angular frequency (measured in radians per sample) derived from the analog frequency F_0 and L is the resulting number of samples.

Nyquist criterion

In order to accurately reproduce a cosine signal, the Nyquist criterion demands $F_0 < F_s/2$, which implies $0 \leq \omega_0 < \pi$. This requirement can be easily illustrated by the following example. Consider two signals:

$$x_1(t) = \cos(0.6\pi t), \quad x_2(t) = \cos(2.6\pi t)$$

Sampling with $F_s = 1$ Hz results in two identical signals,

$$\begin{aligned} x_1[n] &= \cos(0.6\pi n), \\ x_2[n] &= \cos(2.6\pi n) = \cos(0.6\pi n + 2\pi n) = x_1[n]. \end{aligned}$$

This phenomenon is called aliasing. Therefore, the sampling frequency requirement is $0 \leqslant \omega < \pi$. Note, when $\omega_0 = 0$ the signal is the DC level, $y(t) = y[n] = A\cos(\theta)$. For $\omega_0 = \pi$ the signal is "alternating sign" signal $y[n] = A(-1)^n$.

This relation holds for all the following discussions and derivations.

20.1.2. Energy & Power

The energy of (any) signal $x[n]$ is defined as

$$E_{\mathbf{x}} = \|\mathbf{x}\|^2 = \sum_{n=0}^{L-1} x^2[n], \tag{20.4}$$

where $\mathbf{x}$ is the vector of samples of the signal $x[n]$. The corresponding power is time-averaged energy,

$$P_{\mathbf{x}} = \frac{1}{L} E_{\mathbf{x}} = \frac{1}{L}\|\mathbf{x}\|^2. \tag{20.5}$$

The square root of the power,

$$x_{rms} = \sqrt{P_{\mathbf{x}}} = \frac{\|\mathbf{x}\|}{\sqrt{L}}, \tag{20.6}$$

is termed the root mean square (RMS) of the signal. Unlike the power, it carries the units of the signal itself and is therefore directly comparable with the sample values.

Crest Factor

The RMS reports the typical size of a sample and the maximum reports the largest one. Their ratio,

$$\mathrm{CF}_{\mathbf{x}} = \frac{\max_n \left|x[n]\right|}{x_{rms}}, \tag{20.7}$$

is termed the *crest factor*. It quantifies how *peaky* a signal is, that is how far its largest sample stands above its typical one.

Two properties follow directly from Eq. (20.7):

- It is dimensionless and never below 1, because the largest magnitude in a record cannot be smaller than the quadratic average of all of them. The value 1 is reached only by a signal of constant magnitude, such as a square wave or a DC level.
- It is unchanged by a gain: scaling $x[n]$ by c multiplies both $\max_n \left|x[n]\right|$ and x_{rms} by $|c|$. A crest factor is therefore comparable across sensors of different calibration, which is what makes it usable as a feature rather than as a measurement.

Example 20.1 (Crest factor of a sinusoid): Find the crest factor of $x[n] = A\cos(\omega_0 n + \theta)$.

Solution: No sample exceeds A for a densely sampled sinusoid, $\omega_0 \ll \pi$.

The denominator follows from Eq. (20.5) with $\cos^2(\phi) = \frac{1}{2}\left[1 + \cos(2\phi)\right]$,

$$P_{\mathbf{x}} = \frac{A^2}{L}\sum_{n=0}^{L-1}\cos^2(\omega_0 n + \theta) = \frac{A^2}{2} + \frac{A^2}{2L}\sum_{n=0}^{L-1}\cos(2\omega_0 n + 2\theta) \approx \frac{A^2}{2}. \tag{20.8}$$

Hence $x_{rms} = A/\sqrt{2}$ and

$$\mathrm{CF}_{\mathbf{x}} = \frac{A}{A/\sqrt{2}} = \sqrt{2} \approx 1.414, \tag{20.9}$$

independent of the amplitude, the frequency or the phase.

20.1.3. Noise

We define noisy signal

$$y[n] = x[n] + \epsilon[n], \tag{20.10}$$

where $\epsilon(t)$ is additive noise.

The only assumption for the additive noise is that it is zero-mean,

$$\bar{\epsilon} = \frac{1}{L}\sum_{n=0}^{L-1} \epsilon[n] = 0. \tag{20.11}$$

No additional assumptions, such as Gaussianity, are applied; however, the special case of additive white Gaussian noise (AWGN) is further refined as tips for selected topics.

The interpretation of the power (20.5) of the zero-mean noise signal is the (biased) noise variance,

$$P_\epsilon = \sigma_\epsilon^2. \tag{20.12}$$

20.1.4. Basic Signal Analysis

Given the noise signal $y[n]$, the goal is to find the parameters of $x[n]$, as well as noise power, as illustrated in Fig. 20.1.

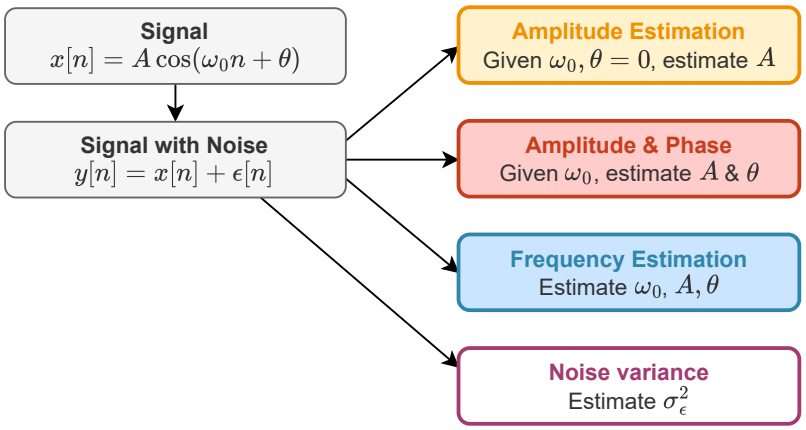


Figure 20.1.: Basic signal analysis goals.

Linear regression model (Chapter 3)

Recall the linear model (Sec. 3.1), with $\mathbf{X}$ built as in (3.7),

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}, \tag{20.13}$$

where $\mathbf{w}$ are learned weights. SSE loss (2.4), in vector form (3.4), is

$$\mathcal{L}(\mathbf{w}) = \|\mathbf{y} - \hat{\mathbf{y}}\|^2 = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 \tag{20.14}$$

with the loss minimum at the normal equation solution (3.5),

$$\hat{\mathbf{w}} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y} \tag{20.15}$$

Due to the error/prediction orthogonality (3.15),

$$\|\mathbf{y}\|^2 = \|\hat{\mathbf{y}}\|^2 + \|\mathbf{e}\|^2 \tag{20.16}$$

In the following, for each method, matrices/vectors $\mathbf{X}, \mathbf{y}, \mathbf{w}$ and there relation to a sinusoidal signal are to be defined.

20.2. Amplitude estimation

Goal: Find the *amplitude* of a sinusoidal signal in noise.

Method: Define matrix/vector $\mathbf{X}$ and vector $\mathbf{y}$. Use $\mathbf{w}$ to find the frequency.

20.2.1. Model

Signal model Given a signal model with $\theta = 0$ and a known frequency ω_0,

$$x[n] = A\cos(\omega_0 n) \tag{20.17}$$

the goal is to estimate the amplitude A from

$$y[n] = x[n] + \epsilon[n] \quad n = 0, \ldots, L-1 \tag{20.18}$$

that best fits a provided model,

$$\hat{y}[n] = \hat{A}\cos(\omega_0 n) \tag{20.19}$$

LS Formulation Technically, we are looking for the value of A that minimizes the squared error,

$$\mathcal{L} = \sum_n (y[n] - \hat{y}[n])^2. \tag{20.20}$$

In the vector form, $\mathbf{X}$ is formed by samples of the signal $\left\{\cos(\omega_0 n)\right\}_{n=0}^{L-1}$ and $\mathbf{y}$ is formed by samples of the signal $\left\{y[n]\right\}_{n=0}^{L-1}$,

$$\mathbf{y} = \begin{bmatrix} y[0] \\ y[1] \\ y[2] \\ \vdots \\ y[L-1] \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} 1 \\ \cos(\omega_0) \\ \cos(2\omega_0) \\ \vdots \\ \cos\big((L-1)\omega_0\big) \end{bmatrix}, \quad \mathbf{w} = A \tag{20.21}$$

such that corresponding vector form of the SSE loss ((20.14)) is

$$\mathcal{L}(A) = \|\mathbf{y} - \mathbf{X}A\|^2 \tag{20.22}$$

The resulting minimization is straightforward,

$$\begin{aligned} \hat{A} &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \\ &= \frac{\sum_n y[n]\cos(\omega_0 n)}{\sum_n \cos^2(\omega_0 n)} \end{aligned} \tag{20.23}$$

Interpretation If we substitute the signal model into the resulting solution,

$$\begin{aligned} \hat{A} &= \frac{\sum_n (A\cos(\omega_0 n) + \epsilon[n])\cos(\omega_0 n)}{\sum_n \cos^2(\omega_0 n)} \\ &= A + \frac{\sum_n \epsilon[n]\cos(\omega_0)}{\sum_n \cos^2(\omega_0 n)} \end{aligned} \tag{20.24}$$

it produces a true value of A with some additive noise.

20.2.2. Signal-to-Noise-Ratio

Using the interpretation of $\|\mathbf{y}\|^2$ as the signal energy and $\|\hat{\mathbf{y}}\|^2$ as an energy of the estimated sinusoidal signal (20.19), the residual error $\|\mathbf{e}\|^2$ (20.16) is the estimation of the noise,

$$\|\mathbf{y}\|^2 = \|\hat{\mathbf{y}}\|^2 + \|\mathbf{e}\|^2 \;\Rightarrow\; \begin{cases} E_{\mathbf{y}} = E_{\hat{\mathbf{y}}} + E_{\mathbf{e}} \\ P_{\mathbf{y}} = P_{\hat{\mathbf{y}}} + P_{\mathbf{e}} \end{cases} \tag{20.25}$$

Moreover, due to zero-mean property of the noise, the estimated power and/or variance of the noise (20.5) is

$$\hat{P}_\epsilon = \hat{\sigma}_\epsilon^2 = \frac{1}{L}\|\mathbf{e}\|^2. \tag{20.26}$$

An interesting interpretation of this result is estimated signal to noise ratio (SNR), defined as

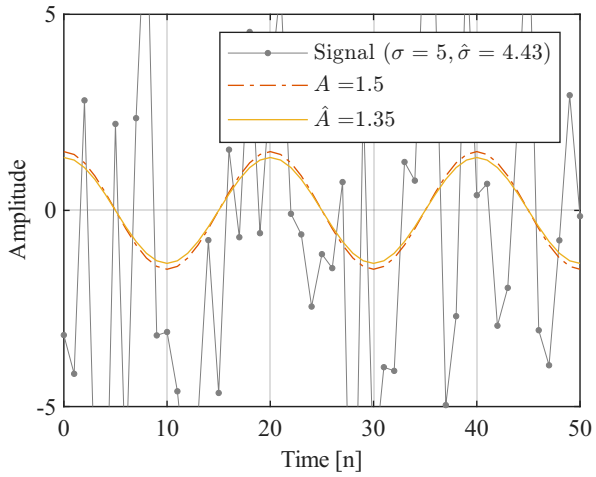
$$\widehat{SNR} = \frac{P_{\hat{\mathbf{y}}}}{P_{\mathbf{e}}} = \frac{\|\hat{\mathbf{y}}\|^2}{\|\mathbf{e}\|^2} \tag{20.27}$$

Since the SNR is a ratio of two powers, it is commonly reported on a logarithmic scale in decibels (dB),

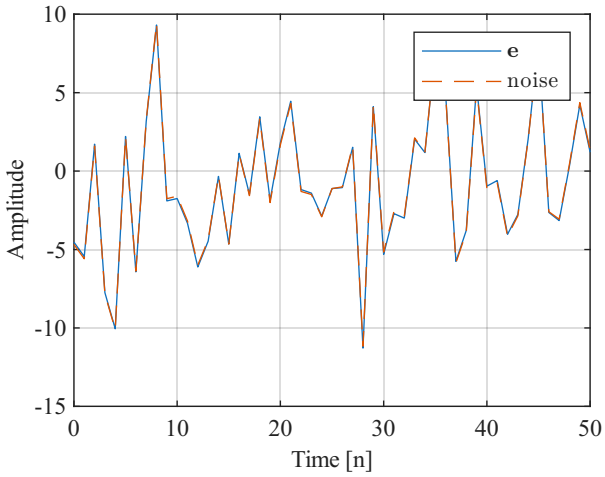
$$SNR_{dB} = 10\log_{10}(SNR), \tag{20.28}$$

so that $SNR = 1$ (equal signal and noise power) corresponds to 0 dB and every negative value indicates a signal weaker than the noise.

Example 20.2: The following example (Fig. 20.2) uses a synthetic cosine signal of length $L = 51$ samples, angular frequency $\omega_0 = 0.1\pi$ and amplitude $A = 1.5$. Gaussian noise with standard deviation $\sigma = 5$ is then added to create a noisy observation. A least-squares regression is applied to estimate the amplitude, yielding $\hat{A} = 1.35$ and $\hat{\sigma}_\epsilon = 4.43$. The theoretical SNR of such a signal is $A^2/(2\sigma^2) = 0.045$ (-13.5 dB), and the value estimated from the residuals (20.27) is $\widehat{SNR} = 0.047$ (-13.2 dB). The signal is therefore more than twenty times weaker than the noise, which is why it is barely visible in Fig. 20.2, yet the amplitude is still recovered within 10%.



(a) Reconstructed signal, $\hat{\mathbf{y}}$. Note the significant noise amount.



(b) Residual error. Ideally, if the model was perfect, the residual would be equal to the added noise.

Figure 20.2.: Example of the cosine signal amplitude estimation.

20.2.3. Theoretical performance bounds (*)

- For AWGN and a known frequency ω_0, the LS solution (20.23) is also the maximum likelihood (ML) estimator of the amplitude, and it is unbiased, as already shown by the interpretation above [12].
- Its variance is available in a closed form and attains the Cramér-Rao lower bound (CRLB) with equality,

$$\mathrm{Var}\left[\hat{A}\right] = \frac{\sigma^2}{\sum_n \cos^2(\omega_0 n)} \approx \frac{2\sigma^2}{L} \quad [V^2], \tag{20.29}$$

  where the approximation uses $\sum_n \cos^2(\omega_0 n) \approx L/2$, valid for any $\omega_0 \neq 0, \pi$.
- The two excluded frequencies are the DC level $x[n] = A$ ($\omega_0 = 0$) and the alternating-sign signal $x[n] = A(-1)^n$ ($\omega_0 = \pi$). For both, the regressor energy is exactly $\sum_n \cos^2(\omega_0 n) = L$ rather than $L/2$, so

$$\mathrm{Var}\left[\hat{A}\right] = \frac{\sigma^2}{L} \qquad \omega_0 = 0, \pi. \tag{20.30}$$

  These are the most accurate cases: the variance is half of the value at any other frequency, i.e. the standard deviation is $\sqrt{2}$ times smaller. For $\omega_0 = 0$ the estimator (20.23) reduces to the sample mean, $\hat{A} = \frac{1}{L}\sum_n y[n]$.
- The practical reading of (20.29) is a relative accuracy of

$$\frac{\sqrt{\mathrm{Var}\left[\hat{A}\right]}}{A} \approx \frac{1}{\sqrt{L \cdot SNR}}, \tag{20.31}$$

  with $SNR = A^2/(2\sigma^2)$. Halving the amplitude error therefore requires four times more samples, and a low SNR can always be compensated by a longer record.
- As above, for other noise distributions or very short records, the estimation variance is readily approximated by Monte-Carlo simulations (Sec. 20.2.4).

20.2.4. Monte-Carlo Simulation

The accuracy of every estimator of this chapter is characterized by repeating the noisy experiment many times, which is the Monte-Carlo simulation of Sec. 1.6. Only its signal-specific form is recalled here.

A single realization is a noise record $\epsilon_j[n]$ of length L, added to the deterministic signal $x[n]$,

$$y_j[n] = x[n] + \epsilon_j[n], \qquad n = 0, \ldots, L-1, \quad j = 1, \ldots, K, \tag{20.32}$$

where K is the number of realizations. Each realization yields one estimate of the parameter of interest, say the amplitude $\hat{A}_j$, and the K collected estimates characterize the estimator by

$$\text{bias}\left[\hat{A}\right] = \frac{1}{K}\sum_{j=1}^{K} \hat{A}_j - A, \tag{20.33}$$

$$\text{Var}\left[\hat{A}\right] \approx \frac{1}{K}\sum_{j=1}^{K} \left(\hat{A}_j - \frac{1}{K}\sum_{i=1}^{K} \hat{A}_i\right)^2, \tag{20.34}$$

which are then compared with the theoretical values, whenever such values exist. The identical loop applies to the phase, the frequency, or any other estimated quantity.

Simulation loop

1. Set the true parameters A, ω_0, θ, the noise power σ_ϵ^2, the record length L, the number of realizations K, and the seed.
2. For $j = 1, \ldots, K$: draw $\epsilon_j[n]$, form $y_j[n]$ by (20.32), and estimate $\hat{A}_j$.
3. Summarize the K estimates by (20.33) and compare against the theoretical bias and variance.

The closed-form expressions of the following sections are derived for AWGN and for long records. The loop above makes neither assumption: the noise in (20.32) may follow any zero-mean distribution and L may be arbitrarily short. This is why the simulation serves as the default validation tool throughout the chapter.

The number of realizations K sets the accuracy of the simulated result itself, as $1/\sqrt{2K}$ for a std (1.28) and $\sqrt{2/K}$ for a variance (1.29). Typically, values of K below a few hundreds are not sufficient to confirm or to reject a theoretical bound. Note that k is reserved for the frequency index in this chapter, hence the capital K is used.

20.3. Amplitude and phase estimation

Goal: Find both the *amplitude* and the *phase* of a sinusoidal signal in noise.

Method: Define matrix $\mathbf{X}$ and vector $\mathbf{y}$. Use $\mathbf{w}$ to recover the amplitude and phase.

20.3.1. Model

Signal model The analysis is provided for the more general model,

$$x[n] = A\cos\left(\omega_0 n + \theta\right) \quad n = 0, \ldots, L-1, \tag{20.35}$$

The goal is to estimate amplitude A and the phase θ that best fits a provided model,

$$\hat{y}[n] = \hat{A}\cos\left(\omega_0 n + \hat{\theta}\right) \tag{20.36}$$

LS Formulation The first step involves the use of trigonometric identities to express the cosine with a phase shift as a linear combination of sine and cosine signals,

$$A\cos\left(\omega_0 n + \theta\right) = w_c\cos(\omega_0 n) + w_s\sin(\omega_0 n), \tag{20.37}$$

where

$$\begin{aligned} w_c &= A\cos(\theta) \\ w_s &= -A\sin(\theta) \end{aligned} \tag{20.38}$$

and

$$\begin{aligned} A &= \sqrt{w_c^2 + w_s^2} \\ \theta &= -\arctan\left(\frac{w_c}{w_s}\right) \end{aligned} \tag{20.39}$$

This transforms the problem into a two-parameter linear LS problem in terms of w_c and w_s [1]. In the vector form, $\mathbf{y}$ is formed by samples of the signal $\{y[n]\}_{n=0}^{L-1}$, the matrix $\mathbf{X}$ has dimensions $L \times 2$, and $\mathbf{w}$ collects the two linear coefficients,

$$\mathbf{y} = \begin{bmatrix} y[0] \\ y[1] \\ y[2] \\ \vdots \\ y[L-1] \end{bmatrix}, \quad \mathbf{X} = \begin{bmatrix} 1 & 0 \\ \cos(\omega_0) & \sin(\omega_0) \\ \cos(2\omega_0) & \sin(2\omega_0) \\ \vdots & \vdots \\ \cos\big((L-1)\omega_0\big) & \sin\big((L-1)\omega_0\big) \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} w_c \\ w_s \end{bmatrix}. \tag{20.40}$$

The SSE loss (20.14) is $\mathcal{L}(\mathbf{w}) = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2$, with the LS solution given by (20.15).

Interpretation Once $\hat{\mathbf{w}}$ has been found, the amplitude and phase can be recovered from (20.39). The SNR and noise variance interpretations follow the orthogonality decomposition (20.16), similar to the previous model in Eqs. (20.27) and (20.26).

Example 20.3: The numerical example is presented in Fig. 20.3. The configuration is similar to the previous figure, except the lower noise variance, $\sigma = 1.5$, and the phase $\theta = -\pi/4$. The theoretical SNR is now $A^2/(2\sigma^2) = 0.5$ (-3 dB), about eleven times higher than in the previous example, and the estimated values are $\widehat{SNR} = 0.51$ (-2.9 dB) and $\hat{\sigma}_\epsilon = 1.32$. Nevertheless, there is a decrease in performance, since two parameters are estimated simultaneously.

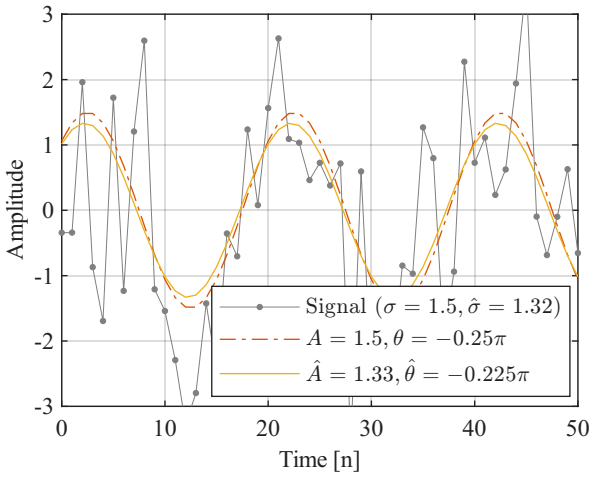


Figure 20.3.: Example of the cosine signal amplitude and phase estimation. Note the lower estimation accuracy compared to Fig. 20.2, since two parameters are estimated simultaneously.

20.3.2. Theoretical performance bounds (*)

- This estimation procedure is optimal in the maximum likelihood (ML) sense under additive white Gaussian noise (AWGN) and achieves the Cramér-Rao lower bound (CRLB) [1, 12].
- The theoretical lower bound (also termed Cramer-Rao lower bound(CRLB)) on the variance of estimation accuracy of w_c, w_s is given by [1, Eqs. (5.47-48)]

$$\mathrm{Var}\left[\hat{w}_{c,s}\right] \gtrsim \frac{2\sigma^2}{L} \tag{20.41}$$

  This bound is the tightest for the AWGN case and is less accurate for other noise distributions.
- Propagating (20.41) through the parameter transformation (20.39) gives the corresponding bounds on the physical parameters [12],

$$\mathrm{Var}\left[\hat{A}\right] \gtrsim \frac{2\sigma^2}{L} \quad [V^2] \tag{20.42}$$

$$\mathrm{Var}\left[\hat{\theta}\right] \gtrsim \frac{2\sigma^2}{LA^2} = \frac{1}{L \cdot SNR} \quad [rad^2] \tag{20.43}$$

  with $SNR = A^2/(2\sigma^2)$. The amplitude bound (20.42) is the same as in the known-phase case (20.29): estimating the phase in addition costs nothing in amplitude accuracy. The phase bound, in contrast, depends on the amplitude itself, since a weak signal is harder to align in time. Both improve as $1/L$.
- While the derivation of CRLB is non-trivial, the approximated numerical estimation variance can be easily evaluated by Monte-Carlo simulations (Sec. 20.2.4) for any set of parameters and any distribution of interest. An example of numerical validation is shown in Fig. 20.4.

20.4. Frequency estimation

Goal: Find the *frequency* ω_0 of a sinusoidal signal in noise, together with its amplitude and phase.

Method: Define a frequency-dependent matrix $\mathbf{X}(\omega)$. Search for the ω that maximizes the reconstructed energy, then use $\mathbf{w}(\hat{\omega}_0)$ to recover the amplitude and phase.

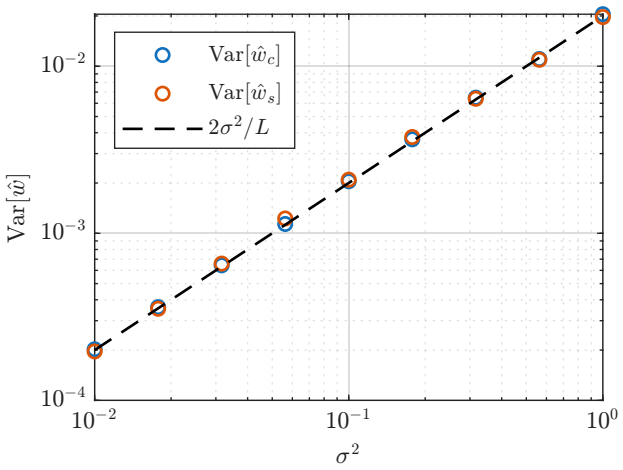


Figure 20.4.: Numerical validation of the bound (20.41) for $A = 1.5$, $\theta = -\pi/4$, $\omega_0 = 0.1\pi$ and $L = 100$, following the Monte-Carlo loop of Sec. 20.2.4 with $K = 2000$ realizations. Each point is the empirical variance (20.34) of one coefficient, itself accurate to $\sqrt{2/K} \approx 3\%$ (1.28). Both coefficients follow the theoretical $2\sigma^2/L$ line to within this accuracy over the whole range of the noise variance.

20.4.1. Model

Signal model Here all of A, θ, ω_0 are unknown. The frequency ω_0 is estimated by searching for the $\hat{\omega}_0$ that best fits a sinusoidal model for the observed data, i.e., that minimizes the residual error norm or maximizes the reconstructed signal energy. The corresponding matrix $\mathbf{X}$ is parameterized as a frequency-dependent one, $\mathbf{X}(\omega)$, so the estimated signal is frequency-dependent

$$\hat{\mathbf{y}}(\omega) = \mathbf{X}(\omega)\mathbf{w}(\omega), \tag{20.44}$$

where $\mathbf{w}(\omega)$ are the estimated parameters w_c and w_s at that frequency, as defined in (20.40).

LS Formulation The corresponding frequency-dependent residual error is given by (see also (20.25))

$$\mathbf{e}(\omega) = \mathbf{y} - \hat{\mathbf{y}}(\omega). \tag{20.45}$$

Since the error $\mathbf{e}(\omega)$ is orthogonal to $\hat{\mathbf{y}}$ (20.16),

$$\|\mathbf{y}\|^2 = \left\|\hat{\mathbf{y}}(\omega)\right\|^2 + \left\|\mathbf{e}(\omega)\right\|^2. \tag{20.46}$$

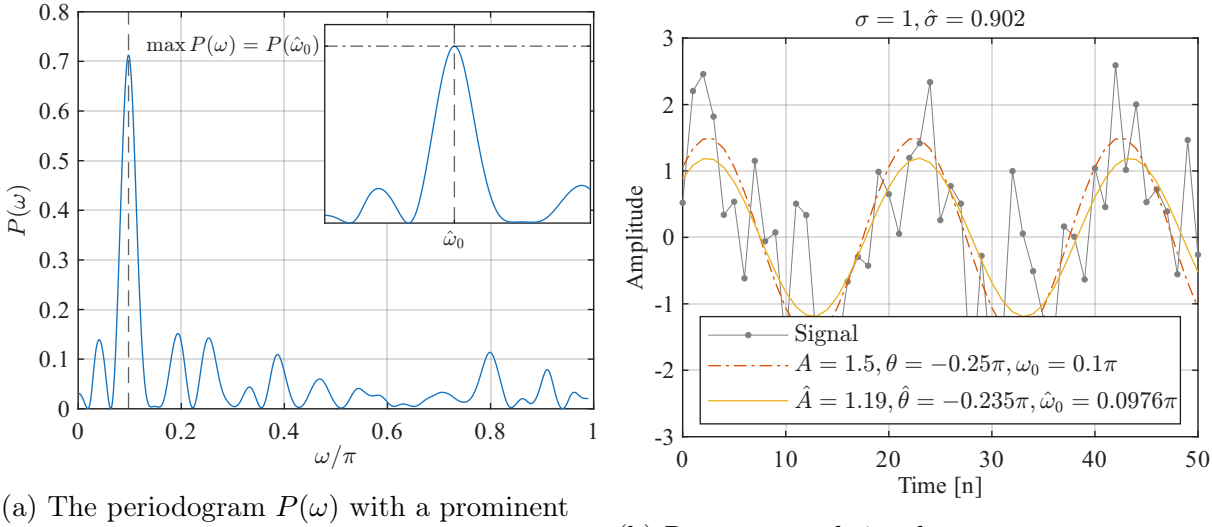
To find the frequency that best represents the data, we seek the one that maximizes the energy of the reconstructed signal (or equivalently minimizes the residual error),

$$\hat{\omega}_0 = \arg\min_{\omega} \left\|\mathbf{e}(\omega)\right\|^2 = \arg\max_{\omega} \left\|\hat{\mathbf{y}}(\omega)\right\|^2. \tag{20.47}$$

Note, this optimization problem can be challenging because the objective function may exhibit multiple local maxima/minima. Therefore, an appropriate numerical global optimization method is required.

Interpretation Once $\hat{\omega}_0$ has been found, the amplitude and phase are estimated using the corresponding linear LS solution $\mathbf{w}(\omega_0)$ (Sec. 20.3). This solution also results in SNR and noise variance estimations, as in Eqs. (20.27) and (20.26).

Example 20.4: A numerical example of the signal with additive white Gaussian noise (AWGN), and with the parameters $A = 1.5, \omega_0 = 0.1\pi, \theta = -\pi/4$ and $\sigma_\epsilon^2 = 1$, is presented in Fig. 20.5. First, periodogram peak is found (Fig. 20.5a). Than, the subsequent amplitude/phase estimation result is presented (Fig. 20.5b).



(a) The periodogram $P(\omega)$ with a prominent peak at $\omega_0 \approx 0.1\pi$.

(b) Reconstracted signal.

Figure 20.5.: The reconstruction in (b) uses the estimated amplitude, phase, and angular frequency $(\hat{A}, \hat{\theta}, \hat{\omega}_0)$ found by maximizing the periodogram in (a). Note the lower estimation accuracy than in Fig. 20.3, since three parameters are estimated.

20.4.2. Periodogram

The function

$$P(\omega) = \frac{1}{L}\left\|\hat{\mathbf{y}}(\omega)\right\|^2 \tag{20.48}$$

as a function of ω is termed a periodogram that is a frequency-dependent quantification of signal power that approximates the power spectral density (PSD) of the signal. By scanning over frequencies, the maximum periodogram value is taken as the frequency estimate,

$$\hat{\omega}_0 = \arg\min_{\omega} P(\omega) \tag{20.49}$$

The complexity of frequency estimation (without amplitude/phase step) can significantly reduced by noting that

$$\arg\max_{\omega} \left\|\hat{\mathbf{y}}(\omega)\right\|^2 \propto \arg\max_{\omega} \left|\sum_{n=0}^{L-1} [n] \exp(-j\omega n)\right|^2 \tag{20.50}$$

However, this definition involves frequencies in the range $0 \leq \omega < 2\pi$; these has challenging physical interpretation.

20.4.3. Theoretical performance bounds (*)

Under AWGN assumption, theoretical SNR is given by

$$SNR = \frac{A^2}{2\sigma^2} \tag{20.51}$$

and the corresponding CRLB on the estimation variances are [12]

$$\mathrm{Var}\left[\hat{A}\right] \geqslant \frac{2\sigma^2}{L} \quad [V^2] \tag{20.52}$$

$$\mathrm{Var}[\hat{\omega}_0] \geqslant \frac{12}{SNR \times L(L^2-1)} \approx \frac{12}{SNR \times L^3} \quad \left[\left(\frac{rad}{sample}\right)^2\right] \tag{20.53}$$

$$\mathrm{Var}\left[\hat{\theta}\right] \geqslant \frac{2(2L-1)}{SNR \times L(L+1)} \approx \frac{4}{SNR \times L} \quad [rad^2] \tag{20.54}$$

The amplitude bound is unchanged with respect to the known-frequency case (20.42), while the phase bound is about four times looser than (20.43): the price of estimating ω_0 is paid by the phase, which is strongly coupled to it.

For analog frequency $F_0 = \frac{\omega_0}{2\pi} F_s$,

$$\mathrm{Var}[F_0] = \mathrm{Var}[\omega_0] \left(\frac{F_s}{2\pi}\right)^2 \quad [Hz^2] \tag{20.55}$$

20.5. Harmonic Signal Analysis

Goal: Decompose a periodic signal into a sum of harmonics (Fourier series) and estimate each amplitude and phase.

Method: Build $\mathbf{X}$ from cosine/sine columns at multiples of ω_0. Use $\mathbf{w}$ to recover the harmonic amplitudes and phases (and ω_0 itself, if unknown).

20.5.1. Model

Signal model A particularly important class of signals encountered in many practical applications is the *harmonic* or *periodic* signal (also termed Fourier series). Such a signal can be expressed as a sum of cosine terms whose frequencies are integer multiples (harmonics) of a fundamental frequency ω_0.

$$y[n] = A_0 + \sum_{m=1}^{M} A_m \cos(m\omega_0 n + \theta_m) = \sum_{m=\mathbf{0}}^{M} A_m \cos(m\omega_0 n + \theta_m), \tag{20.56}$$

where:

- A_0 is the constant (DC) component,
- A_m and θ_m represent the amplitude and phase of the m-th harmonic,
- ω_0 is the fundamental angular frequency,
- $m\omega_0$ corresponds to the frequency of the m-th harmonic,

- and M is the number of harmonics in the model.

The model order M (number of harmonics) is a hyper-parameter that should be chosen carefully. Too few harmonics can fail to capture essential signal structure, while too many may overfit noise. The maximum value of M is bounded by the Nyquist criterion, $M\omega_0 < \pi$.

LS Formulation Given ω_0, the model is linear in terms of the unknown parameters $\{A_m, \theta_m\}$ for each harmonic $m = 1, \ldots, M$. Similar to the single-frequency case, the LS matrix $\mathbf{X}$ is constructed with columns corresponding to $\cos(m\omega_0 n)$ and $\sin(m\omega_0 n)$ for $m = 1, \ldots, M$, plus a column of ones for the DC component,

$$\mathbf{X} = \begin{bmatrix} 1 & \cdots & 1 & 0 & \cdots \\ 1 & \cdots & \cos(\omega_0 m) & \sin(\omega_0 m) & \cdots \\ 1 & \cdots & \cos(2\omega_0 m) & \sin(2\omega_0 m) & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \\ 1 & \cdots & \cos\big((L-1)\omega_0 m\big) & \sin\big((L-1)\omega_0 m\big) & \cdots \end{bmatrix}, \quad \mathbf{w} = \begin{bmatrix} A_0 & \cdots & w_{c,m} & w_{s,m} & \cdots \end{bmatrix}^T. \tag{20.57}$$

Each pair (A_m, θ_m) can be recovered from the LS estimated cosine and sine coefficients in the manner described for single-frequency amplitude-phase estimation from per-harmonic cosine/sine coefficient pairs $w_{c,m}, w_{s,m}$. The resulting SNR and noise variance estimates are similar to those described in the previous sections.

Fundamental frequency estimation If ω_0 is not known, the approach that is described in the frequency estimation section can also be applied here. Once $\hat{\omega}_0$ has been determined from a maximum of the harmonic periodogram,

$$P_h(\omega) = \frac{1}{L} \sum_{m=1}^{M} \left\| \hat{\mathbf{y}}(m\omega) \right\|^2, \tag{20.58}$$

the harmonic amplitudes and phases can be estimated via LS at this frequency [3].

Total harmonic distortion (THD)

Total harmonic distortion (THD) is a measure commonly used in electrical engineering, audio processing, and other fields to quantify how much the harmonic components of a signal differ from a pure sinusoid at the fundamental frequency. It is defined as the ratio of the root-sum-square of the harmonic amplitudes and the amplitude of the fundamental frequency,

$$THD = \frac{\sqrt{\sum_{m=2}^{M} A_m^2}}{A_1}. \tag{20.59}$$

A lower THD value indicates that the signal is closer to a pure sinusoidal shape, whereas a higher THD signifies a stronger presence of higher-order harmonics.

Example 20.5: The example is the sampled current of a switch-mode power supply in a 50Hz network sampled at a 50kHz frequency [2]. Figure 20.6a shows a reconstruction of the signal with $M = 250$ harmonics. The estimated amplitudes $\hat{A}_m$ are shown (Fig. 20.6b) as a function of the harmonic index m, including the DC term at $m = 0$. A larger magnitude indicates a more

prominent harmonic component. The first non-DC harmonic amplitude $m = 1$ corresponds to the fundamental frequency, ω_0, while higher indices capture additional harmonics in the signal. The estimated fundamental frequency is 50.104Hz with the corresponding THD of about 1.6 ratio ($\approx 160\%$). Figure 20.6c shows estimated SNR (top) and the noise standard deviation (bottom) vary as the number of harmonics M in the model increases.

20.5.2. Theoretical performance bounds (*)

- The presented estimator, for both known and unknown ω_0, is an efficient ML estimator in the case of AWGN. It is a minimum variance unbiased estimator (MVUE) and attains the CRLB asymptotically, i.e. for a large number of samples L [3].
- For a *known* ω_0 the model is linear, and the harmonic columns of $\mathbf{X}$ are (nearly) orthogonal to each other. Every harmonic therefore behaves as the isolated two-parameter problem of Sec. 20.3, with the bounds (20.42) and (20.43) applied per harmonic,

$$\mathrm{Var}\left[\hat{A}_m\right] \gtrsim \frac{2\sigma^2}{L} \quad [V^2] \tag{20.60}$$

$$\mathrm{Var}\left[\hat{\theta}_m\right] \gtrsim \frac{2\sigma^2}{LA_m^2} = \frac{1}{L \cdot SNR_m} \quad [rad^2] \tag{20.61}$$

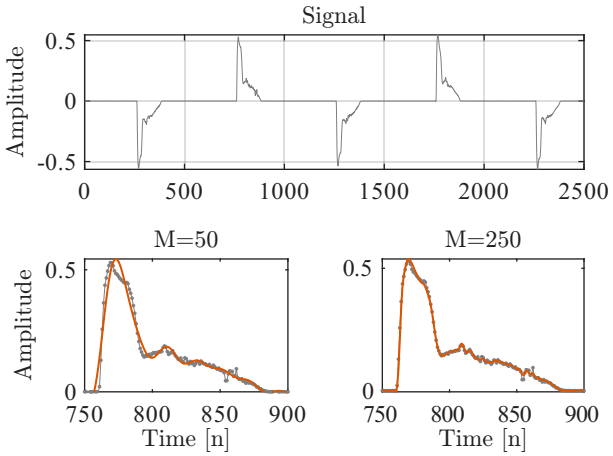
  with the per-harmonic $SNR_m = A_m^2/(2\sigma^2)$. Weak harmonics have accurate amplitudes but poorly defined phases. The DC term is the single exception and is twice as accurate, $\mathrm{Var}\left[\hat{A}_0\right] \gtrsim \sigma^2/L$, because its column of ones has squared norm L instead of $L/2$.
- For an *unknown* ω_0, the whole set of harmonics contributes to the fundamental frequency estimate [3],

$$\mathrm{Var}[\hat{\omega}_0] \gtrsim \frac{24\sigma^2}{L^3 \sum_{m=1}^{M} m^2 A_m^2} = \frac{12}{SNR_h \cdot L^3} \quad \left[\left(\frac{rad}{sample}\right)^2\right], \tag{20.62}$$

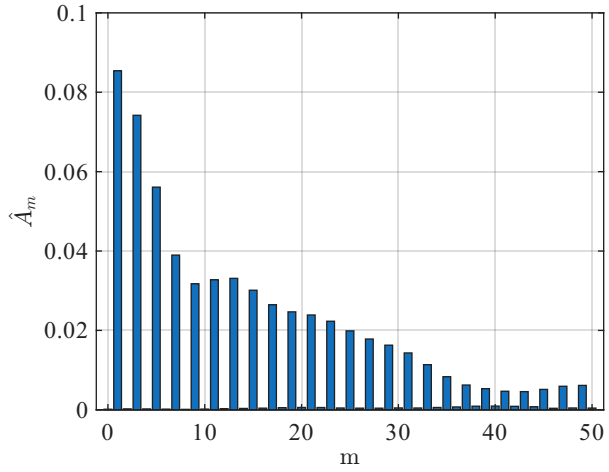
  where

$$SNR_h = \frac{1}{2\sigma^2} \sum_{m=1}^{M} m^2 A_m^2 \tag{20.63}$$

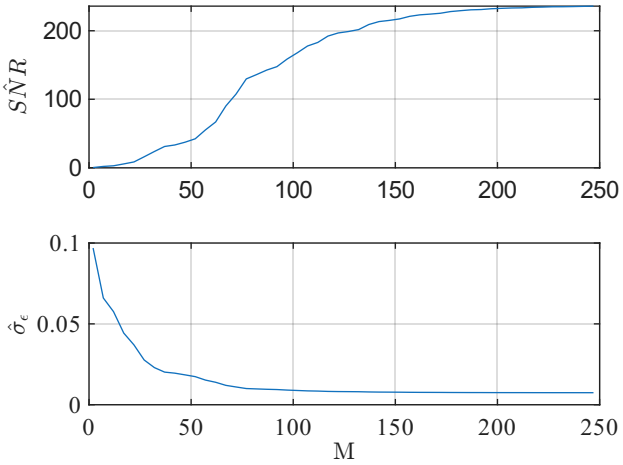
  is the harmonic SNR, in which each harmonic is weighted by m^2. This is exactly the single-tone bound of Sec. 20.4 with SNR replaced by SNR_h, and it reduces to it for $M = 1$. The analog frequency bound follows as before, $\mathrm{Var}\left[\hat{F}_0\right] = \mathrm{Var}[\hat{\omega}_0]\,(F_s/2\pi)^2$.
- The practical reading of (20.62) is that a harmonic model estimates the fundamental frequency far better than a single tone of the same power does. Every additional harmonic contributes $m^2A_m^2$, so the high harmonics are the valuable ones even when their amplitudes are moderate. For M harmonics of comparable amplitude A, $\sum_m m^2A_m^2 \approx A^2M^3/3$, and the frequency variance drops roughly as M^3. Fitting the harmonics jointly is therefore much cheaper than lengthening the record, which only gives L^3.
- As above, for other noise distributions or short records, the estimation variance is readily approximated by Monte-Carlo simulations (Sec. 20.2.4).



(a) The upper plot shows the signal overlaid with the least-squares harmonic reconstruction using the estimated frequency, amplitude, and phase parameters. The lower plots zoom in on a smaller portion of the time axis for different values of M, and demonstrate the challenging shape of the signal.



(b) Estimated harmonic amplitudes, A_m.



(c) Estimated SNR and noise std, $\hat{\sigma}_\epsilon$.

Figure 20.6.: Example of a harmonic signal analysis.

20.6. Discrete Fourier Transform (DFT)

Goal: Represent any finite-length signal as a sum of N harmonically related sinusoids, evenly spaced in frequency.

Method: Apply the harmonic model with $N \geq L$ harmonics at fundamental frequency $\omega_0 = \frac{2\pi}{N}$. The orthogonal LS solution yields the spectral coefficients $\{A_k, \theta_k\}$.

20.6.1. Definition

The discrete Fourier transform (DFT) is a special case of the harmonic model with:

- $N \geq L$ harmonics,
- fundamental angular frequency $\omega_0 = \dfrac{2\pi}{N}$.

Formally speaking, DFT is harmonic decomposition of a finite-length signal into a sum of N harmonically related sinusoids that are evenly spaced in frequency.

DFT representation states that:

- any arbitrary, finite-time signal $y[n]$ may be represented as a sum of sinusoidal signals,

$$y[n] = \sum_{k=0}^{N-1} A_k \cos\left(k\frac{2\pi}{N}n + \theta_k\right), \quad n = 0, \ldots, L-1 \tag{20.64}$$

- For $N \geq L$, the DFT allows for *perfect* reconstruction of the signal using its harmonic representation parameters, $\{A_k, \theta_k\}_{k=0}^{N-1}$.

The corresponding LS formulation follows the harmonic model above for known ω_0.

20.6.2. Properties

Zero-padding

The basic requirement is $N = L$ harmonic signals. Sometimes, denser spectral representation with $N > L$ is required. In this case, the corresponding time-domain signal can be interpreted as the original signal $y[n]$ with $N - L$ zeros padded at the end of this signal. This case it termed *zero padding*.

Symmetry

It is also worth noting the symmetry in the DFT,

$$\begin{aligned} \cos\big((N-k)\Delta\omega\big) &= \cos(k\Delta\omega) \\ \sin\big((N-k)\Delta\omega\big) &= -\sin(k\Delta\omega), \end{aligned} \tag{20.65}$$

resulting in redundant information for frequencies $k\omega_0$ above and below π,

$$\begin{aligned} A_k &= A_{N-k} \\ \theta_k &= -\theta_{N-k} \end{aligned}. \tag{20.66}$$

As a result, only frequencies $k\omega_0 \leq \pi$ need to be considered uniquely.

20.6.3. Power Spectral Density

The power of a signal of the form

$$x_k[n] = A_k \cos\left(k\frac{2\pi}{N}n + \theta_k\right) \tag{20.67}$$

is

$$S_{\mathbf{y}}[k] = P_{\mathbf{y}_k} = \frac{1}{L}\|\mathbf{y}_k\|^2 = \frac{A_k^2}{2}. \tag{20.68}$$

This value is known as the power spectral density (PSD) at the frequency $\omega = k\frac{2\pi}{N}$. The corresponding squared magnitude values $A_k^2/2$ are known as the discrete-frequency periodogram (Eq. (20.48)), and this is the basic method for the PSD estimation of a signal. Plotting such a periodogram gives a frequency-domain representation of the signal's power distribution, highlighting which frequencies carry the most power.

Parseval's Theorem DFT is energy conservation transform with

$$\sum_{k=0}^{N-1} A_k^2 = \frac{1}{L}\|\mathbf{y}\|^2. \tag{20.69}$$

20.6.4. Advanced notes (*)

Fast Fourier transform (FFT) & Goertzel algorithm

Goal: Effective algorithms for Fourier transform.

The fast Fourier transform (FFT) algorithm efficiently computes $Y[k] = \text{DFT}\left\{y[n]\right\}$, providing $A_k = |Y[k]|/N$ and $\theta_k = \angle(Y[k])$ with significantly lower memory requirements and complexity than direct calculation in Eq. (20.80).

When only a single frequency value is of interest, Goertzel algorithm is more efficient method for the task. Moreover, it can be used for computationally effective peaking of the maximum in Eq. (20.47).

Parseval's Theorem (Energy Preservation) Using FFT notation,

$$A_k^2 = \left|Y[k]\right|^2/L^2 \Rightarrow \sum_{k=0}^{L-1}\left|Y[k]\right|^2 = L\|\mathbf{y}\|^2. \tag{20.70}$$

Sinusoidal signal power

Goal: Understand how the power of a finite-length sinusoid depends on its frequency.

For a more general case of an arbitrary ω value, the signal of the form

$$y[n] = A\cos(\omega_0 n) \tag{20.71}$$

has an ω_0-dependent power. It is convenient to collect this dependence in a single quantity, the *regressor energy*

$$\mathcal{E}(\omega_0, L) \triangleq \sum_{n=0}^{L-1} \cos^2(\omega_0 n) = \frac{L}{2} + \frac{\sin(L\omega_0)\cos\big((L-1)\omega_0\big)}{2\sin(\omega_0)}, \tag{20.72}$$

which counts how much energy the cosine $\cos(\omega_0 n)$ accumulates over the L available samples. The same quantity is the denominator of the amplitude estimate (20.23), so it also determines the estimation accuracy: the more energy the regressor carries, the more accurate the estimate. With this notation the power is

$$\begin{aligned} P_{\mathbf{y}} &= \frac{A^2}{L}\mathcal{E}(\omega_0, L) \\ &= \frac{A^2}{4L}\left(1 + 2L - \frac{\sin(\omega_0 - 2L\omega_0)}{\sin(\omega_0)}\right), \end{aligned} \tag{20.73}$$

where the ω_0-dependence results from the time-limited origin of the signal $y[n]$. For almost every frequency the second term of (20.72) is negligible against $L/2$, so

$$\mathcal{E}(\omega_0, L) \approx \frac{L}{2} \qquad \omega_0 \neq 0, \pi, \tag{20.74}$$

and for the infinite number of samples the power converges to the continuous-time power expression,

$$\lim_{L\to\infty} P_{\mathbf{y}} \to \frac{A^2}{2}. \tag{20.75}$$

The two excluded frequencies are the DC level $y[n] = A$ ($\omega_0 = 0$) and the alternating-sign signal $y[n] = A(-1)^n$ ($\omega_0 = \pi$). There every sample of $y^2[n]$ equals A^2 instead of oscillating between 0 and A^2, so the energy is exactly $\mathcal{E} = L$ and the power is A^2, twice the value of any other frequency. The illustration of (20.73) over the whole band is presented in Fig. 20.7a, and the origin of this factor of two is shown in Fig. 20.7b.

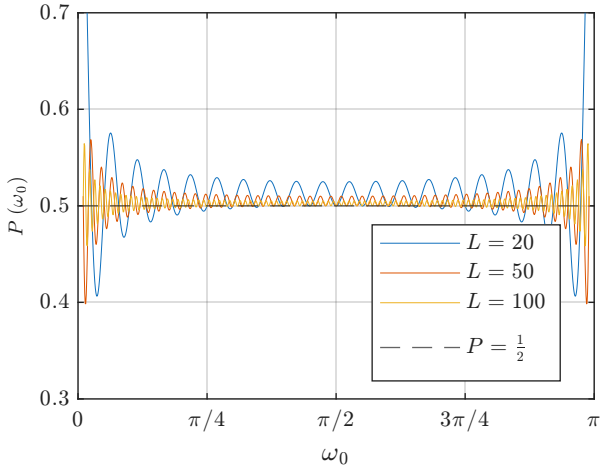
For almost any frequency, the regressor energy is $\mathcal{E}(\omega_0, L) \approx L/2$ and the power of a finite sinusoid is $P_{\mathbf{y}} \approx A^2/2$. The deviation appears only near $\omega_0 = 0$ and $\omega_0 = \pi$, where $\mathcal{E}$ grows towards L, and it is a consequence of the finite number of samples L rather than of the signal itself. As $L \to \infty$ the power converges exactly to $A^2/2$ for every frequency. This same finite-length effect is what later produces spectral spreading and leakage (Sec. 20.6.5).

Single frequency analysis

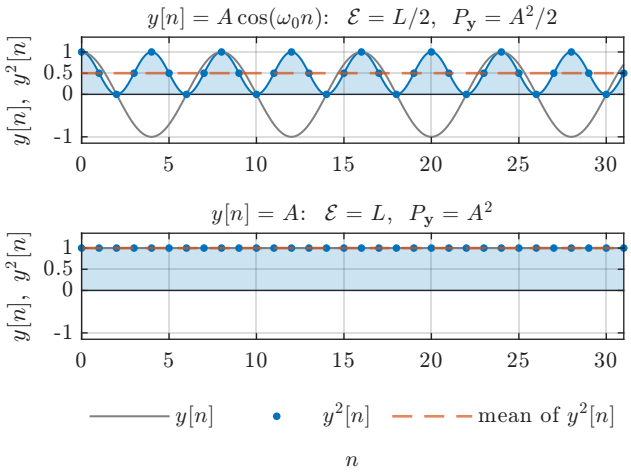
Goal: Estimate the amplitude and phase at one exact DFT frequency, where the cosine and sine columns are orthogonal.

Consider a signal $y_k[n]$ with a discrete frequency $\omega_k = \dfrac{2\pi}{N}k$ is given,

$$y_n[k] = A\cos(\omega_k n + \theta) = A\cos\left(\frac{2\pi}{N}kn + \theta\right) \tag{20.76}$$



(a) Power of the sinusoidal signal as a function of ω_0 for different values of L. The deviation from $A^2/2$ is visible only near the band edges, and it shrinks as L grows.



(b) Where the factor of two comes from, for $A = 1$ and $L = 32$. The squared samples of a cosine (top) swing between 0 and A^2 and average to $A^2/2$, so the shaded energy is $\mathcal{E} = L/2$. For a constant signal of the same amplitude (bottom) every squared sample equals A^2, the shaded energy doubles to $\mathcal{E} = L$, and the power is A^2.

Figure 20.7.: Power of a finite-length sinusoid and its dependence on the frequency.

To estimate amplitude and phase at this predefined frequency, we can form a matrix $\mathbf{X}$,

$$\mathbf{X}_k = \begin{bmatrix} \mathbf{x}_c & \mathbf{x}_s \end{bmatrix},$$

where

$$\mathbf{x}_c = \begin{bmatrix} \cos\left(2\pi \frac{k}{N} \cdot 0\right) \\ \cos\left(2\pi \frac{k}{N} \cdot 1\right) \\ \vdots \\ \cos\left(2\pi \frac{k}{N}(L-1)\right) \end{bmatrix} \text{ and } \mathbf{x}_s = \begin{bmatrix} \sin\left(2\pi \frac{k}{N} \cdot 0\right) \\ \sin\left(2\pi \frac{k}{N} \cdot 1\right) \\ \vdots \\ \sin\left(2\pi \frac{k}{N}(L-1)\right) \end{bmatrix}.$$

By evaluating

$$\mathbf{X}_k^T \mathbf{X}_k = \begin{bmatrix} \mathbf{x}_c^T \mathbf{x}_c & \mathbf{x}_s^T \mathbf{x}_c \\ \mathbf{x}_s^T \mathbf{x}_c & \mathbf{x}_s^T \mathbf{x}_s \end{bmatrix} \tag{20.77}$$

we find that the sine and cosine columns form an orthogonal basis for this single frequency, with for $0 < k < \frac{N}{2}$:

$$\begin{aligned} \mathbf{x}_c^T \mathbf{x}_c &= \sum_{n=0}^{N-1} \cos^2\left(2\pi \frac{k}{N} n\right) = \frac{N}{2} \\ \mathbf{x}_c^T \mathbf{x}_s &= \sum_{n=0}^{N-1} \cos\left(2\pi \frac{k}{N} n\right) \sin\left(2\pi \frac{k}{N} n\right) = 0 \\ &= \mathbf{x}_s^T \mathbf{x}_c \\ \mathbf{x}_s^T \mathbf{x}_s &= \sum_{n=0}^{N-1} \sin^2\left(2\pi \frac{k}{N} n\right) = \frac{N}{2} \end{aligned} \tag{20.78}$$

For the special cases $k = 0$ (DC) and, when N is even, $k = N/2$ (Nyquist), the cosine column has squared norm N rather than $N/2$ (it is all ones at DC and alternating $(-1)^n$ at Nyquist). At those same indices the sine column is identically zero.

Stacking these results for all $k = 0, \ldots, N-1$ yields the complete DFT matrix forms a complete orthogonal basis for the L-sample signal space. The further discussion of $\left(\mathbf{X}^T \mathbf{X}\right)^{-1}$ matrix properties can be found in Examples 4.2 and 8.5 in [12].

Moreover, since $\left(\mathbf{X}_k^T \mathbf{X}_k\right)^{-1}$ takes a particularly simple diagonal form,

$$\left(\mathbf{X}_k^T \mathbf{X}_k\right)^{-1} = \begin{bmatrix} \frac{2}{N} & 0 \\ 0 & \frac{2}{N} \end{bmatrix} \tag{20.79}$$

and the least squares solution $\hat{\mathbf{w}}$ for the parameters $w_{c,k}$ and $w_{s,k}$ (corresponding to amplitude and phase components at ω_k) is

$$w_{c,k} = \frac{2}{N} \sum_{n=0}^{L-1} y[n] \cos\left(2\pi \frac{k}{N} n\right), \tag{20.80a}$$

$$w_{s,k} = \frac{2}{N} \sum_{n=0}^{L-1} y[n] \sin\left(2\pi \frac{k}{N} n\right). \tag{20.80b}$$

For the special cases $k = 0$ (DC) and, when N is even, $k = N/2$ (Nyquist), the cosine column has squared norm N rather than $N/2$, since it is all ones at DC and alternates as $(-1)^n$ at Nyquist,

$$w_{c,0} = \frac{1}{N}\sum_{n=0}^{N-1} y[n], \qquad w_{c,N/2} = \frac{1}{N}\sum_{n=0}^{N-1} y[n](-1)^n, \qquad w_{s,0} = w_{s,N/2} = 0 \tag{20.81}$$

At an exact DFT frequency the matrix $\mathbf{X}_k^T\mathbf{X}_k$ is diagonal with both entries equal to $N/2$. The two-parameter LS problem therefore decouples: the cosine and sine coefficients are obtained independently, and each reduces to a simple correlation of the signal with the corresponding cosine or sine. No matrix inversion is needed, and these correlation sums in (20.80) are exactly the DFT coefficients.

General orthogonality

Goal: Show that the orthogonality extends across all DFT frequencies, so the entire matrix $\mathbf{X}^T\mathbf{X}$ is diagonal.

The single-frequency result of Eq. (20.78) extends to *different* harmonic indices $j \neq k$, which is what makes the full DFT matrix $\mathbf{X}^T\mathbf{X}$ orthogonal and its inverse trivial,

$$\sum_{n=0}^{N-1} \cos\left(2\pi\frac{j}{N}n\right)\cos\left(2\pi\frac{k}{N}n\right) = \frac{N}{2}\delta\left[j-k\right] \tag{20.82}$$

$$\sum_{n=0}^{N-1} \cos\left(2\pi\frac{j}{N}n\right)\sin\left(2\pi\frac{k}{N}n\right) = 0 \quad \forall j,k \tag{20.83}$$

$$\sum_{n=0}^{N-1} \sin\left(2\pi\frac{j}{N}n\right)\sin\left(2\pi\frac{k}{N}n\right) = \frac{N}{2}\delta\left[j-k\right]. \tag{20.84}$$

The Kronecker delta $\delta[j-k]$ makes every cross-frequency inner product vanish, so $\mathbf{X}^T\mathbf{X} = \frac{N}{2}\mathbf{I}$ for the full DFT matrix. Each frequency is therefore independent of all the others: the DFT coefficients can be read off one frequency at a time (20.80), with no matrix inversion and no interaction between frequencies.

20.6.5. Spectral Spreading and Leakage (*)

Goal: Understand why a pure sinusoid does not always appear as a single spectral line.

In an idealized setting, a pure cosine signal has a perfectly defined frequency representation. For instance, consider the discrete-time signal,

$$x[n] = A\cos\left(k_0\frac{2\pi}{L}n\right), \; k_0 \in \{1,\cdots,L-1\} \tag{20.85}$$

where k_0 is the frequency index. The Fourier transform of this signal yields a single spectral component at frequency $w_0 = k_0 \frac{2\pi}{L}$, such that the spectral amplitude A_k at each value of k is given by

$$A_k = \begin{cases} \frac{A}{2} & k = k_0, N - k_0 \\ 0 & \text{otherwise} \end{cases} . \tag{20.86}$$

Under these conditions, the signal's spectral representation seems to be strictly localized at the specific frequency ω_k, with no energy distributed elsewhere in the spectrum. However, practical

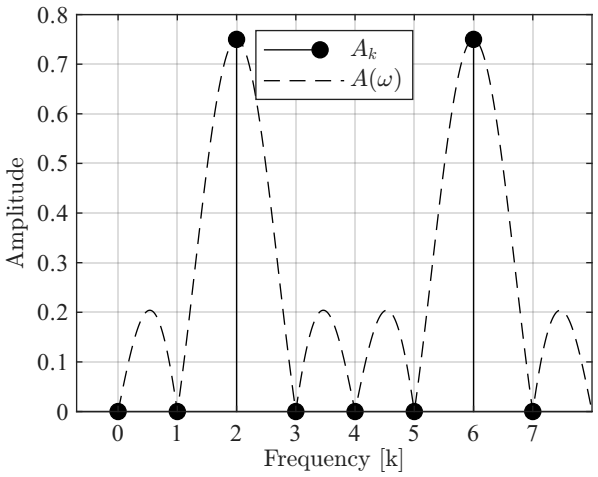


Figure 20.8.: Illustration of a single-frequency cosine signal's spectrum under ideal assumptions (discrete, integer-multiple frequencies) versus practical conditions (denser frequency grid or non-integer frequencies). Note how the ideal single peak broadens and additional low-level components appear, highlighting the effects of spectral spreading and leakage.

scenarios deviate from this ideal case. In particular, if a denser frequency grid is employed (i.e. $N > L$) or the frequency varies continuously (as in Eq. (20.44)), the resulting spectral distribution can differ substantially from the discrete, single-peak ideal (Fig. 20.8). This difference arises because, in general, $\mathbf{X}(\omega)^T\mathbf{X}(\omega)$ is not orthogonal as in Eq. (20.78). As a result, two effects are introduced:

- The main frequency peak broadens, resulting in "spectral spreading".
- Additional frequency components emerge beyond the broadened main peak, termed "spectral leakage."

20.7. Short-Time Fourier Transform (STFT)

All the analysis presented above, e.g. harmonic analysis and DFT, assume that the signal has time-constant harmonic components (termed *stationary* signal) over the entire observation interval of L samples. In many practical applications, however, the frequency content of a signal changes over time.

Goal: Capture time-varying spectral behavior.

20.7.1. Definition

The short-time Fourier transform (STFT) addresses analysis of time-varying spectral behavior by applying the DFT to short, overlapping segments of the signal. A window function $w[n]$ of length N_w is used to extract a local segment centered around a given time instant. The ℓ-th windowed segment is defined as

$$y_\ell[n] = w[n] \cdot y[\ell H + n], \quad n = 0, \ldots, N_w - 1, \tag{20.87}$$

where H is the hop size (the number of samples between consecutive windows) and $\ell = 0, 1, 2, \ldots$ is the segment (frame) index.

Each windowed segment $y_\ell[n]$ is then analyzed using the DFT framework described above, i.e., the LS matrix $\mathbf{X}$ of dimension $N_w \times 2N_w$ with cosine and sine columns at the DFT frequencies $\omega_k = \frac{2\pi k}{N_w}$ is applied to the segment. The resulting DFT coefficients become time-dependent,

$$A_k[\ell], \quad \theta_k[\ell], \quad k = 0, \ldots, N_w - 1, \tag{20.88}$$

providing the amplitude and phase of each frequency component at each time frame ℓ.

20.7.2. Spectrogram

Goal: Visualize how a signal's power is distributed jointly across time and frequency.

The power spectral density of each segment defines a time-frequency representation known as the spectrogram,

$$S[\ell, k] = \frac{A_k^2[\ell]}{2}, \tag{20.89}$$

which displays the signal's power distribution as a function of both time (frame index ℓ) and frequency (index k). The spectrogram is one of the most widely used tools for visualizing non-stationary signals and serves as a basis for feature extraction in signal classification tasks.

The construction of the spectrogram from the windowed frames is illustrated in Fig. 20.9: each frame contributes a single column of the time-frequency image.

The following example uses a chirp signal whose instantaneous frequency sweeps linearly from 50 Hz to 400 Hz over 1 s, sampled at $F_s = 1000$ Hz, with additive Gaussian noise ($\sigma = 0.5$). The time-domain signal is shown in Fig. 20.10a and its spectrogram ($N_w = 128$, $H = 16$, Hann window) in Fig. 20.10b.

20.7.3. Time-frequency trade-off

The choice of the window length N_w involves a fundamental trade-off:

- A longer window (large N_w) provides better frequency resolution (smaller $\Delta\omega = \frac{2\pi}{N_w}$) but poorer time localization, since each segment spans a longer time interval.
- A shorter window (small N_w) provides better time localization but coarser frequency resolution.

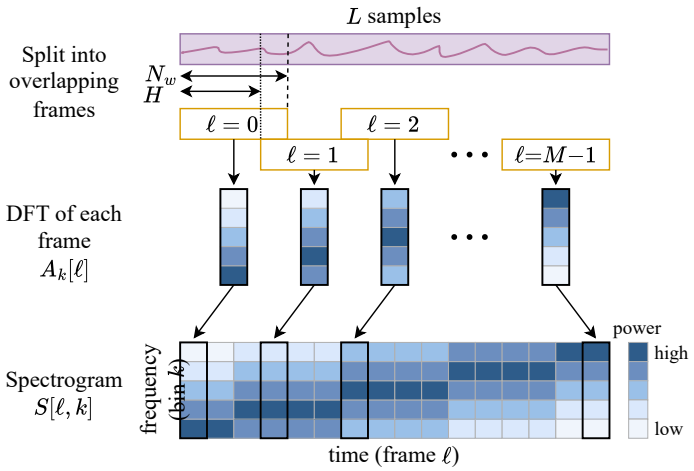


Figure 20.9.: Construction of the spectrogram. The signal is split into overlapping frames of length N_w with hop H; the DFT of each frame yields the amplitudes $A_k[\ell]$, and the resulting column $S[\ell, k] = A_k^2[\ell]/2$ is placed at time index ℓ of the time-frequency image.

This time-frequency trade-off is inherent to STFT and cannot be resolved by the choice of window function alone. Figure 20.10c illustrates this trade-off: a short window ($N_w = 32$) provides sharp time localization but blurred frequency content, while a long window ($N_w = 256$) yields precise frequency resolution at the cost of temporal smearing.

Software reference

Common implementations of STFT and spectrogram:

- **Python** — `scipy.signal.stft`, `scipy.signal.spectrogram`; `librosa.stft`.
- **MATLAB** — `stft`, `spectrogram`, `pspectrum`.

Summary

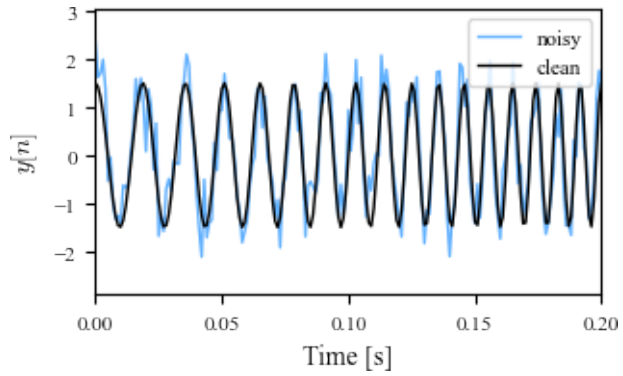
The STFT extends the DFT to non-stationary signals by applying windowed DFT analysis to overlapping segments of the signal. The key parameters are:

- Window length N_w — controls the time-frequency resolution trade-off.
- Hop size H — determines the overlap between consecutive frames.
- Window function $w[n]$ — shapes the spectral characteristics of each segment (e.g., Hann, Hamming).

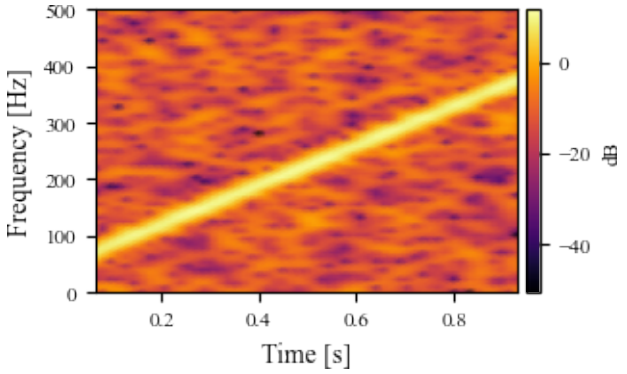
The resulting spectrogram $S[\ell, k]$ provides a real-valued two-dimensional representation of the signal's power as a function of time and frequency.

20.7.4. Welch's Periodogram

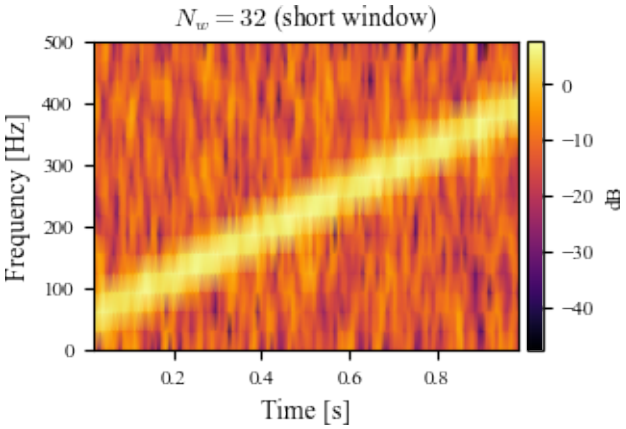
Goal: Reduce the variance of the raw periodogram estimate.

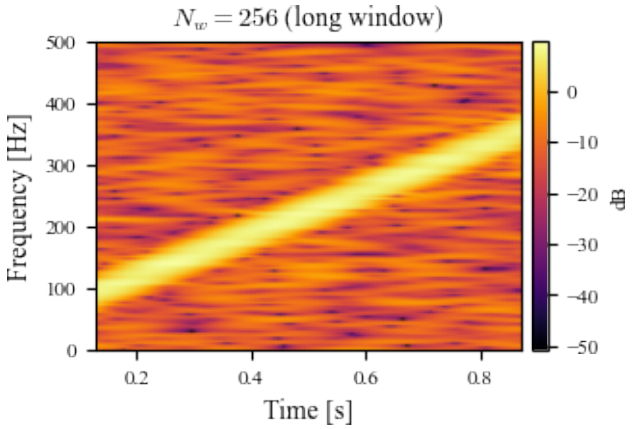


(a) Time-domain chirp signal with linearly increasing frequency and additive noise.



(b) Spectrogram ($N_w = 128$, $H = 16$). The linearly increasing frequency is clearly visible.





(c) Time-frequency trade-off. Left: short window ($N_w = 32$) — good time resolution, poor frequency resolution. Right: long window ($N_w = 256$) — good frequency resolution, poor time resolution.

Figure 20.10.: Example of STFT analysis of a chirp signal.

The raw periodogram, $\hat{S}_{\mathbf{xx}}[k] = \left|\mathrm{DFT}\left\{x[n]\right\}\right|^2 / L$, is an asymptotically unbiased but inconsistent estimator of the PSD: its variance does not vanish as the signal length L grows. Welch's method addresses this by trading frequency resolution for variance reduction through segment averaging.

The procedure is:

1. Split $x[n]$ of length L into M segments of length N_{seg}, with overlap N_{ov} (typically 50%).
2. Apply a window $w[n]$ (e.g., Hann) to each segment to reduce spectral leakage.
3. Compute the periodogram of each windowed segment,

$$\hat{S}_{\mathbf{xx}}^{(m)}[k] = \frac{1}{U N_{seg}} \left|\mathrm{DFT}\left\{w[n]\, x_m[n]\right\}\right|^2, \tag{20.90}$$

   where $U = \frac{1}{N_{seg}} \sum_n w^2[n]$ compensates for the window energy.

4. Average across segments,

$$\hat{S}_{\mathbf{xx},welch}[k] = \frac{1}{M} \sum_{m=1}^{M} \hat{S}_{\mathbf{xx}}^{(m)}[k]. \tag{20.91}$$

Properties

- Variance decreases approximately as $1/M$, so longer segments yield finer frequency resolution while more segments yield smoother estimates.
- Frequency resolution is dictated by the segment length, $\Delta f \approx f_s/N_{seg}$, not the total signal length.

Bartlett's method The special case with no overlap ($N_{ov} = 0$) and a rectangular window is known as Bartlett's method. Welch's modification (overlap and tapering) typically yields lower variance for the same L.

20.8. Quasi-periodic signals

Goal: Introduce the regime between strictly periodic and fully non-stationary signals, and clarify how the tools developed in this chapter behave in that case.

A *quasi-periodic* signal has a clear repeating structure with an identifiable period, but no integer T satisfies $x[n+T] = x[n]$ exactly. Successive cycles drift slowly in amplitude, phase, or local period.

Three regimes

This chapter has implicitly addressed three regimes; the quasi-periodic regime sits between them.

- *Stationary / strictly periodic* (Sec. 20.5 and the DFT section): the same sinusoidal parameters describe the entire record of length L.
- *Quasi-periodic*: a dominant period exists but its individual cycles are not identical.
- *Non-stationary* (Sec. 20.7): the spectral content itself moves over time.

Canonical forms

- Slow amplitude/phase modulation,

$$x[n] = A[n]\cos\left(\omega_0 n + \theta[n]\right), \tag{20.92}$$

  where $A[n]$ and $\theta[n]$ vary on a much slower timescale than ω_0.
- Sums of incommensurate sinusoids, e.g. $\cos(n) + \cos\left(\sqrt{2}\,n\right)$, which look periodic locally but never exactly repeat.
- Natural sources: voiced speech (wandering pitch), ECG (heart-rate variability), mechanical vibration under varying load.

Behavior of the chapter's tools

- *Harmonic LS* (Sec. 20.5) assumes constant A_m, θ_m over the whole record. With slow modulation, LS returns time-averaged values and the residual systematically exceeds the noise floor.
- *DFT and periodogram* (Sec. 20.4.2) show a broadened spectral line instead of a sharp delta. This spreading is of physical origin and is distinct from the windowing-induced leakage discussed in Sections. 20.6.5 and 20.7.

Table 20.1.: Comparison and summary of different signal estimation methods.

Task	Parameters	Matrix $\mathbf{X}$	SNR
Amplitude only (Sec. 20.2)	A given $\omega_0, \theta = 0$	A single column of $\cos(\omega_0 n)$	$\frac{\|\hat{\mathbf{y}}\|^2}{\|\mathbf{e}\|^2}$
Amplitude & phase	A, θ given ω_0	Two columns of $\cos(\omega_0 n)$ and $\sin(\omega_0 n)$	$\frac{\|\hat{\mathbf{y}}\|^2}{\|\mathbf{e}\|^2}$
Frequency estimation	ω_0, A, θ	Frequency-dependent $\cos(\omega n)$ and $\sin(\omega n)$ columns	Maximum of $\frac{\|\hat{\mathbf{y}}(\omega)\|^2}{\|\mathbf{e}(\omega)\|^2}$
Fourier series (harmonic decomposition)	$A_0, \{A_m, \theta_m\}_{m=1}^{M}$, possibly ω_0	Harmonic cos/sin columns at multiples of ω_0, $\cos(m\omega_0 n)$, $\sin(m\omega_0 n)$	$\frac{\|\hat{\mathbf{y}}\|^2}{\|\mathbf{e}\|^2}$, can include frequency dependence if ω_0 unknown
DFT	$\{A_k, \theta_k\}_{k=0}^{N-1}$	Multiple pairs of columns $\cos\left(\frac{2\pi k}{N} n\right)$, $\sin\left(\frac{2\pi k}{N} n\right)$ for $k = 0, \ldots, N-1$	Not used directly. *Perfect* reconstruction for $N \geq L$

- *STFT* (Sec. 20.7) recovers the slow variation as time-dependent coefficients $A_k[\ell], \theta_k[\ell]$, trading frequency resolution for time localization.

20.9. Summary

The summary of the presented approach is shown in Table 20.1. The presented approach involves a design of matrix $\mathbf{X}$ and using LS to estimate unknown parameters. The key addressed task are as follows.

Amplitude Estimation With a known frequency ω_0, the amplitude A is found via LS. The resulting residuals provide noise variance and SNR estimates.

Amplitude and Phase Estimation For known ω_0, rewriting

$$A\cos(\omega_0 n + \theta) = w_c \cos(\omega_0 n) + w_s \sin(\omega_0 n)$$

transforms the problem into a two-parameter LS regression.

Frequency Estimation: If ω_0 is unknown, it is found by searching for the frequency that maximizes the fitted signal energy.

Harmonic Signal Analysis: Signals can be expressed as sums of multiple harmonics. Extending the LS approach to multiple harmonics allows estimation of each amplitude and phase. THD quantifies deviations from a pure tone.

Discrete Fourier Transform (DFT): The DFT is a special case of harmonic modeling, decomposing a signal into equally spaced frequency components. Efficiently computed by the FFT, the DFT is central to signal spectral analysis.

STFT: For non-stationary signals, the STFT applies the DFT to short, overlapping windowed segments, producing time-dependent spectral coefficients. The resulting spectrogram provides a two-dimensional time-frequency representation of the signal's power distribution. The window length N_w controls the trade-off between time and frequency resolution.

Performance bounds For AWGN, all the estimators above are ML estimators and attain the Cramer-Rao lower bound (CRLB), so the achievable accuracy is known in closed form [12, 10]. The bounds collected along the chapter are summarized in Table 20.2. Three rules follow from it.

- Amplitude accuracy is $2\sigma^2/L$ in *every* case. Estimating the phase, the frequency, or additional harmonics on top of it costs nothing in amplitude accuracy. The only exception is a constant (DC) or an alternating-sign signal, $\omega_0 = 0, \pi$, whose amplitude is estimated twice as accurately, σ^2/L. The same holds for the DC term A_0 of a harmonic model.
- Phase accuracy scales as $1/(L \cdot SNR)$ and frequency accuracy as $1/(L^3 \cdot SNR)$. Doubling the record length halves the phase error, but reduces the frequency error by a factor of about eight. Record length is thus far more valuable for frequency than for amplitude or phase.
- For a harmonic signal only the frequency bound improves, through the m^2-weighted harmonic SNR (20.63): high harmonics sharpen the fundamental frequency estimate, roughly as M^3.

These expressions hold for AWGN. For other noise distributions or short records, the bias and the variance of the estimators are readily approximated via Monte-Carlo simulations (Sec. 20.2.4) under any parameter setting of interest.

Table 20.2.: Summary of the AWGN performance bounds of the chapter. $SNR = A^2/(2\sigma^2)$ is the single-tone SNR, $SNR_m = A_m^2/(2\sigma^2)$ is the per-harmonic SNR, and SNR_h is the harmonic SNR (20.63). The phase entry of the harmonic row assumes a known ω_0.

Task	$\mathrm{Var}\left[\hat{A}\right]$	$\mathrm{Var}\left[\hat{\theta}\right]$	$\mathrm{Var}[\hat{\omega}_0]$
Amplitude only (20.29)	$\dfrac{2\sigma^2}{L}$	n/a	n/a
DC and Nyquist, $\omega_0 = 0, \pi$ (20.30)	$\dfrac{\sigma^2}{L}$	n/a	n/a
Amplitude & phase (20.42), (20.43)	$\dfrac{2\sigma^2}{L}$	$\dfrac{1}{L \cdot SNR}$	n/a
Frequency estimation (Sec. 20.4)	$\dfrac{2\sigma^2}{L}$	$\approx \dfrac{4}{L \cdot SNR}$	$\approx \dfrac{12}{L^3 \cdot SNR}$
Harmonic, M tones (20.62)	$\dfrac{2\sigma^2}{L}$	$\dfrac{1}{L \cdot SNR_m}$	$\approx \dfrac{12}{L^3 \cdot SNR_h}$

Table 20.3.: Summary of $\mathbf{X}$ matrices.

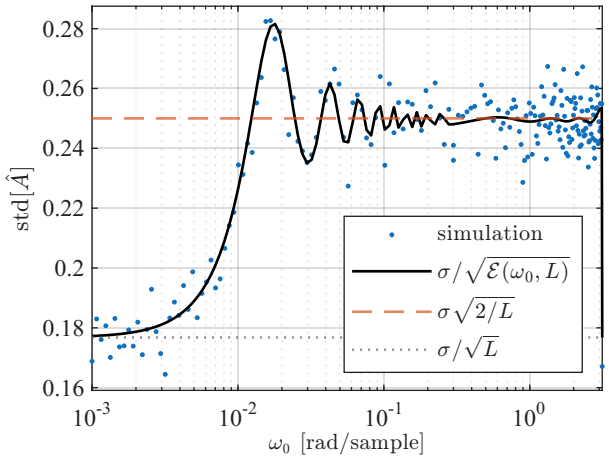
Task	$\mathbf{X}$
Amplitude	$\mathbf{X} = \begin{bmatrix} 1 \\ \cos(\omega_0) \\ \cos(2\omega_0) \\ \cdots \\ \cos\big((L-1)\omega_0\big) \end{bmatrix}$
Amplitude & phase	$\mathbf{X} = \begin{bmatrix} 1 & \\ \cos(\omega_0) & \sin(\omega_0) \\ \cos(2\omega_0) & \sin(2\omega_0) \\ \vdots & \vdots \\ \cos\big((L-1)\omega_0\big) & \sin\big((L-1)\omega_0\big) \end{bmatrix}$
Frequency	$\mathbf{X}(\omega) = \begin{bmatrix} 1 & \\ \cos(\omega) & \sin(\omega) \\ \cos(2\omega) & \sin(2\omega) \\ \vdots & \vdots \\ \cos\big((L-1)\omega\big) & \sin\big((L-1)\omega\big) \end{bmatrix}$
Fourier series, ω_0 known	$\mathbf{X} = \begin{bmatrix} 1 & \cdots & 1 & 0 & \cdots \\ 1 & \cdots & \cos(\omega_0 m) & \sin(\omega_0 m) & \cdots \\ 1 & \cdots & \cos(2\omega_0 m) & \sin(2\omega_0 m) & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \\ 1 & \cdots & \cos\big((L-1)\omega_0 m\big) & \sin\big((L-1)\omega_0 m\big) & \cdots \end{bmatrix}$
Fourier series, ω_0 unknown	$\mathbf{X}(\omega)$
DFT	$\mathbf{X} = \begin{bmatrix} 1 & \cdots & 1 & 0 & \cdots \\ 1 & \cdots & \cos\left(\frac{2\pi}{N}k\right) & \sin\left(\frac{2\pi}{N}k\right) & \cdots \\ 1 & \cdots & \cos\left(2\frac{2\pi}{N}k\right) & \sin\left(2\frac{2\pi}{N}k\right) & \cdots \\ \vdots & & \vdots & \vdots & \ddots \\ 1 & \cdots & \sin\left((L-1)\frac{2\pi}{N}k\right) & \sin\left((L-1)\frac{2\pi}{N}k\right) & \cdots \end{bmatrix}$

20.10. Exercise

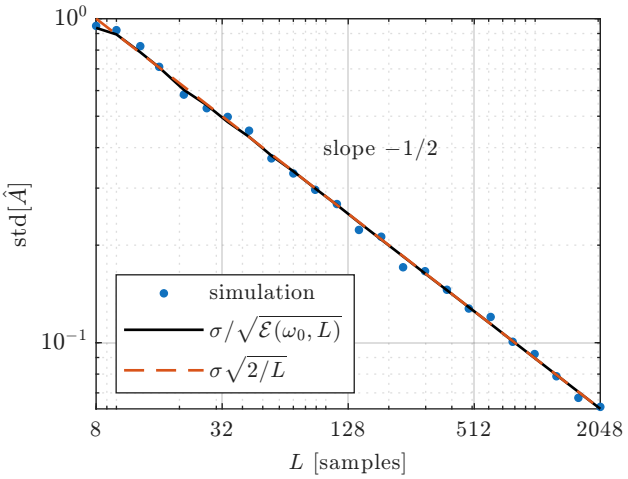
1 (Amplitude estimation accuracy) Validate the amplitude estimation bound (20.29) by Monte-Carlo simulation, following the loop of Sec. 20.2.4. Use $y[n] = A\cos(\omega_0 n) + \epsilon[n]$ with AWGN, e.g. $A = 2$ and $\sigma = 2$ ($SNR = -3$ dB), and estimate $\hat{A}$ by (20.23). For every parameter point, average over at least 100 independent noise realizations.

1. Verify that the estimator is unbiased.
2. Fix $L = 128$ and sweep ω_0 over the whole band $(0, \pi]$. Compare the sample standard deviation of $\hat{A}$ with $\sigma/\sqrt{\mathcal{E}(\omega_0, L)}$ and with the approximation $\sigma\sqrt{2/L}$, where the regressor energy $\mathcal{E}(\omega_0, L)$ is given by (20.72). An example of the expected result is shown in Fig. 20.11a.
3. Fix ω_0 and sweep L from 8 to 2048. Show that the result is a straight line of slope $-1/2$ on log-log axes, as in Fig. 20.11b.
4. Explain what happens near $\omega_0 = 0$ and $\omega_0 = \pi$?

Hint: use the closed form of $\mathcal{E}(\omega_0, L)$ given in (20.72), and keep in mind that it is $0/0$ at $\omega_0 = 0, \pi$, where the sum has to be evaluated directly.



(a) Frequency sweep of item 2, $L = 128$. Over almost the whole band the accuracy does not depend on ω_0 and stays at $\sigma\sqrt{2/L}$. It improves by $\sqrt{2}$ only within about one period of the record length from the band edges, where the regressor energy grows from $L/2$ to L.



(b) Record-length sweep of item 3, $\omega_0 = 0.3\pi$. On log-log axes the accuracy is a straight line of slope $-1/2$ (the measured slope is -0.4995), so halving the estimation error costs four times the record length.

Figure 20.11.: Accuracy of the LS amplitude estimate, for $A = 2$, $\sigma = 2$ and 500 repetitions per point.

21. ARMA Model

Goal: Linear prediction coefficients for systems-inspired models.

This chapter extends least-squares (LS)-based modeling to systems and time-series contexts, focusing on linear prediction coefficients derived from signals. Instead of restricting to a predefined parametric form like a sinusoid, here we consider arbitrary zero-mean finite-time signals $x[n]$ and $y[n], n = 0, \ldots, L-1$, in the presence of zero-mean noise, $\epsilon[n]$.

Contents

21.1. Auto-Correlation Function . . . **317**
21.1.1. AR(1) . . . 317
21.1.2. Auto-correlation function (ACF) . . . 320
21.1.3. ACF Properties . . . 323
21.1.4. Auto-covariance . . . 325
21.2. Power Spectral Density (PSD) . . . **325**
21.2.1. Relation between ACF and PSD . . . 325
21.2.2. Frequency estimation via ACF peak . . . 326
21.3. AR(p) Model . . . **328**
21.3.1. Yule-Walker Form . . . 330
21.3.2. Moving Average Filter . . . 332
21.3.3. Nearest Neighbor (Naïve) . . . 332
21.4. Linear Prediction of Sinusoidal Signal . . . **333**
21.5. Partial auto-correlation function . . . **335**
21.5.1. Relation between PACF and AR(p) . . . 336
21.6. MA model . . . **336**
21.6.1. MA and AR relations . . . 337
21.6.2. The relation between MA(q) and ACF . . . 338
21.7. ARMA . . . **339**
21.7.1. Order Selection . . . 342
21.7.2. Prediction Horizon . . . 343
21.7.3. Rational PSD (*) . . . 345

21.1. Auto-Correlation Function

Goal: Evaluate correlation between a signal and its time-shifted replicas to derive meaningful insights.

21.1.1. AR(1)

In system modeling and time-series analysis, a common approach is to express the current sample of a signal in terms of its past values. A simple first-order autoregressive (AR(1)) model predicts the current sample $x[n]$ from a single past sample $x[n-1]$, using the system model

$$x[n] = h_1 x[n-1] + \epsilon[n]. \tag{21.1}$$

and

$$\hat{x}[n] = h_1 x[n-1] \tag{21.2}$$

Here, h_1 is the linear prediction coefficient that indicates how strongly the previous sample $x[n-1]$ influences the current sample $x[n]$.

To determine h_1, minimization of SE loss function

$$\mathcal{L}(h_1) = \sum_n \big(x[n] - h_1 x[n-1])\big)^2 \tag{21.3}$$

is used. Setting the derivative of $\mathcal{L}(h_1)$ with respect to h_1 to zero leads to

$$\frac{d\mathcal{L}(a)}{da} = 2\sum_n (x[n] - h_1 x[n-1])(-x[n-1]) = 0 \tag{21.4}$$

Finally, the LS solution for h_1 is

$$h_1 = \frac{\sum_n x[n]x[n-1]}{\sum_n x^2[n-1]}. \tag{21.5}$$

The value h_1 is termed as the (single) prediction coefficient of AR(1) model.

Matrix formulation

The coefficient may be formulated as a relation between two vectors

$$\hat{\mathbf{y}} = \begin{bmatrix} x[1] \\ x[2] \\ \vdots \\ x[L-2] \\ x[L-1] \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x[0] \\ x[1] \\ \vdots \\ x[L-3] \\ x[L-2] \end{bmatrix}, \tag{21.6}$$

such that

$$\hat{\mathbf{y}} = h_1 \mathbf{x}. \tag{21.7}$$

The correspoding MMSE loss is

$$\mathcal{L}(h_1) = \|\mathbf{y} - h_1 \mathbf{x}\|^2 \tag{21.8}$$

and its minimum is given by normal equation, (Eq. (3.5)),

$$h_1 = \left(\mathbf{x}^T\mathbf{x}\right)^{-1}\mathbf{x}^T\mathbf{y}. \tag{21.9}$$

The substitution results in the solution identical to Eq. (21.5).

Prediction with k-samples difference

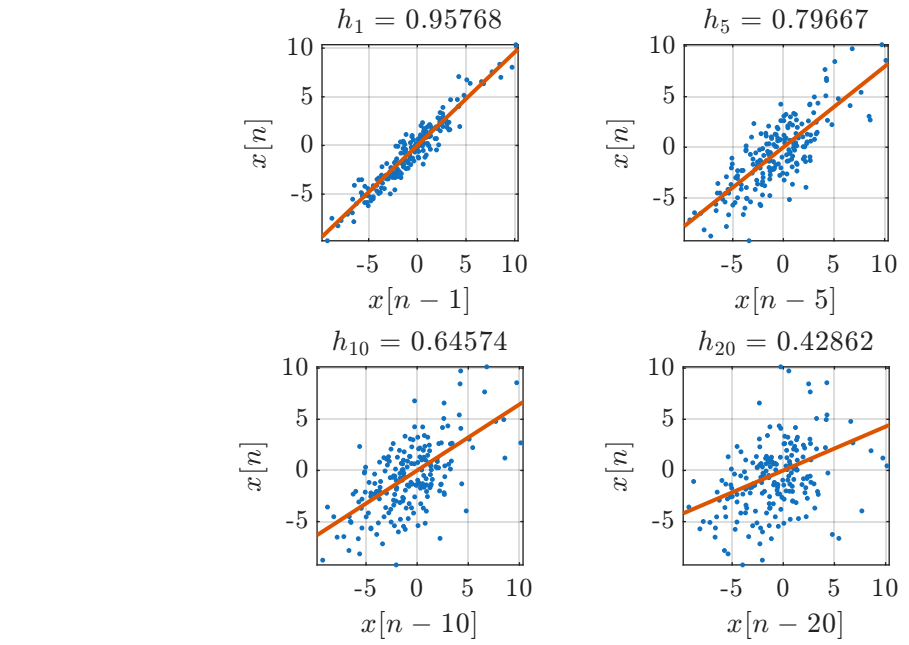
To generalize beyond a single lag, we use one-coefficient prediction from the current sample $x[n]$ and its time-shifted version $x[n-k]$, using the k-step predictor,

$$\hat{x}[n] = h_k x[n-k]. \tag{21.10}$$

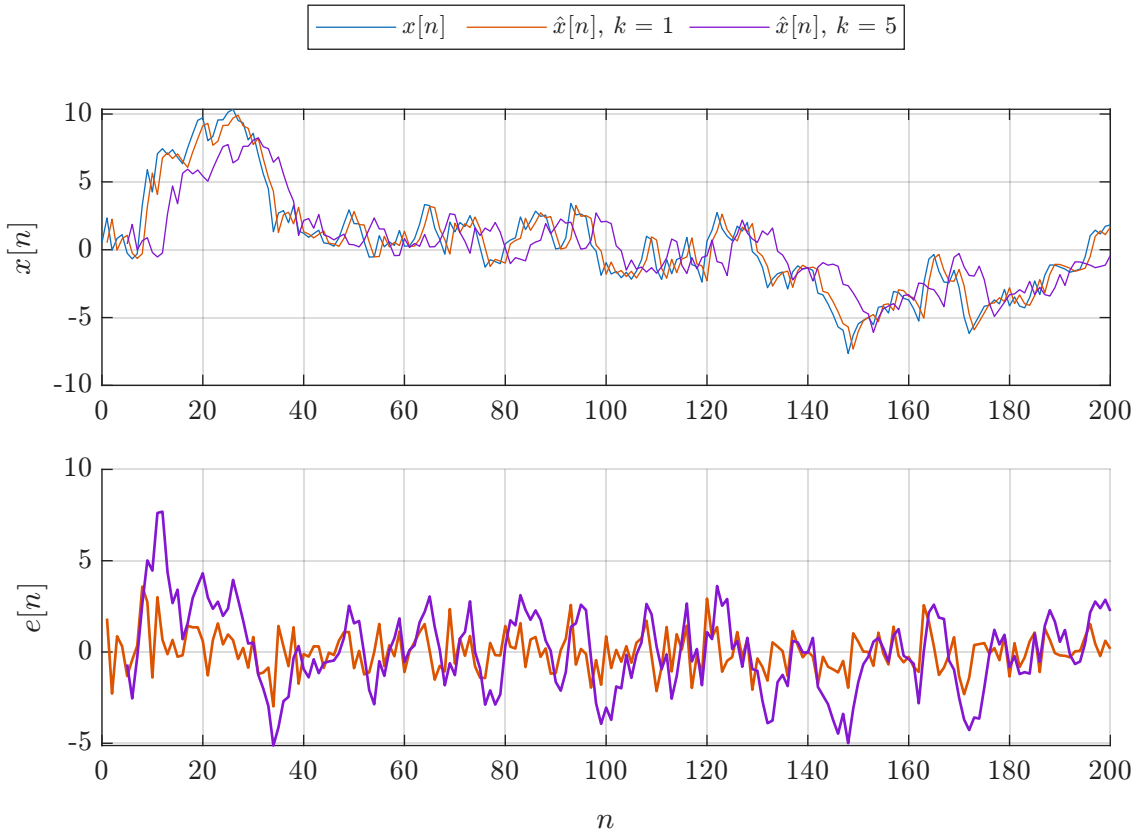
The corresponding solution is very similar to the solution for $k = 1$,

$$h_k = \frac{\sum_n x[n]x[n-k]}{\sum_n x^2[n-k]}. \tag{21.11}$$

The linear dependence between $x[n]$ and $x[n-k]$, and the prediction it enables, are illustrated in Fig. 21.1 for an AR(1) signal. The dependence weakens as the lag grows, and with it the accuracy of the prediction.



(a) $x[n]$ vs. $x[n-k]$



(b) Prediction and error

Figure 21.1.: One-coefficient prediction of an AR(1) signal. In (a) the line is the fitted predictor $\hat{x}[n] = h_k x[n-k]$, with $h_1 = 0.9577$, $h_5 = 0.7967$, $h_{10} = 0.6457$ and $h_{20} = 0.4286$; the cloud widens around it as the lag grows. In (b) the $k = 5$ prediction trails the signal and its error is correspondingly larger, RMSE $= 2.103$ against RMSE $= 0.999$ for $k = 1$.

21.1.2. Auto-correlation function (ACF)

The nominator of Eq. (21.11) is termed (raw, or unscaled) auto-correlation function (ACF) at lag k,

$$R_{\mathbf{xx}}[k] = \sum_n x[n]x[n-k], \quad k = 0, \ldots, L-1 \tag{21.12}$$

The ACF provides a measure of how linearly dependent the signal is on its shifted versions. Variants like biased, unbiased, and normalized ACF offer different normalization schemes to handle finite data length and scaling issues.

The signal energy is given by

$$E_{\mathbf{x}} = \sum_{n=0}^{L-1} x^2[n] = R_{\mathbf{xx}}[0] \tag{21.13}$$

Biased auto-correlation Another useful definition is averaged sum of $y[n]y[n-k]$,

$$\begin{aligned} R_{\mathbf{xx},biased}[k] &= \frac{1}{L}\sum_n x[n]x[n-k], \quad k = 0, \ldots, L-1 \\ &= \frac{1}{L} R_{\mathbf{xx}}[k] \end{aligned} \tag{21.14}$$

is termed biased (see Sec. 1.1.3 for bias definition) ACF. The corresponding average power is given by

$$P_{\mathbf{x}} = \frac{1}{L}\sum_n x^2[n] = R_{\mathbf{xx},biased}[0] \tag{21.15}$$

Normalized auto-correlation Another version is normalized ACF of the form

$$R_{\mathbf{xx},norm}[k] = \frac{R_{\mathbf{xx}}[k]}{R_{\mathbf{xx}}[0]} \lesssim h_k. \tag{21.16}$$

Note, the difference between denominator above and the one in Eq. (21.11) (where the lagged sum runs over $n = k, \ldots, L-1$) is

$$\sum_{n=0}^{L-1} x^2[n] - \sum_{n=k}^{L-1} x^2[n-k] = x^2[L-k] + \cdots + x^2[L-1], \tag{21.17}$$

i.e. the last k samples of the squared signal. Under stationarity and $k \ll L$, these boundary terms are negligible relative to $R_{\mathbf{xx}}[0]$, so h_k in Eq. (21.11) and $R_{\mathbf{xx},norm}[k]$ are approximately equal for a sufficiently high L. Moreover, here the denominator is k-independent.

Unbiased auto-correlation Note, that the ACF includes summation only of $n-k$ terms. Another useful normalization is

$$\begin{aligned} R_{\mathbf{xx},unbiased}[k] &= \frac{1}{L-k}\sum_n x[n]x[n-k] \\ &= \frac{1}{L-k} R_{\mathbf{xx}}[k] \end{aligned} \tag{21.18}$$

This time it is assumed that

$$\begin{aligned} \frac{1}{L}\sum_n x^2[n] &\approx \frac{1}{L-k}\sum_n x^2[n-k] \\ \frac{x^2[0]+x^2[1]+\cdots x^2[L-1]}{L} &\approx \frac{x^2[k]+\cdots x^2[L-1]}{L-k} \end{aligned} \tag{21.19}$$

and the resulting expression

$$\begin{aligned} \frac{L}{L-k}\frac{R_{\mathbf{xx}}[k]}{R_{\mathbf{xx}}[0]} &= \frac{L}{L-k}R_{\mathbf{xx},norm}[k] \\ &= \frac{R_{\mathbf{xx},unbiased}[k]}{R_{\mathbf{xx},unbiased}[0]} \approx h_k \end{aligned} \tag{21.20}$$

is assumed to be closer approximation to h_k than the normalized auto-correlation, for a relatively small values of k.

An AR(1) signal and its normalized ACF are shown in Fig. 21.2. The ACF starts at $R_{\mathbf{xx},norm}[0] = 1$ and decays with the lag, reproducing the loss of predictability seen in Fig. 21.1.

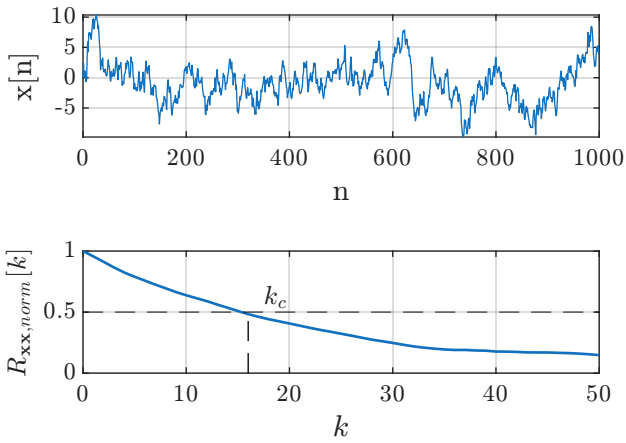


Figure 21.2.: An AR(1) signal and its normalized ACF, with the correlation time k_c marked at the 0.5 threshold.

Correlation time The correlation time, k_c is the lag where $R_{\mathbf{xx},norm}[k_c]$ falls below a threshold,

$$R_{\mathbf{xx},norm}[k_c] = 0.5 \text{ or } 0.1 \text{ or } \exp(-1) \tag{21.21}$$

The predictability is assumed negligible for $k > k_c$,

$$R_{\mathbf{xx},norm}[k > k_c] \approx 0 \tag{21.22}$$

The decision threshold depends on the field of application. For example, one of the most applicable ACF for natural processes is

$$R_{\mathbf{xx},norm}[k] = \exp\left(-\frac{k}{k_c}\right) \tag{21.23}$$

with $R_{\mathbf{xx},norm}[k_c] = \exp(-1)$.

Note, correlation time is mostly used in physics and engineering models and is less applicable in social sciences.

Correlation Coefficient Interpretation

All normalizations above still depend on the amplitude of the signal: replacing $x[n]$ by $ax[n]$ multiplies $R_{\mathbf{xx}}[k]$ by a^2. Comparing the predictability of two different signals, or of the same signal recorded at a different gain, therefore requires a quantity on a fixed scale. The *correlation coefficient* achieves this by dividing the raw ACF by the energies of both vectors being compared,

$$\rho_{\mathbf{xx}}[k] \triangleq \frac{\sum_n x[n]x[n-k]}{\sqrt{\sum_n x^2[n]\,\sum_n x^2[n-k]}} = \frac{R_{\mathbf{xx}}[k]}{\sqrt{R_{\mathbf{xx}}[0]\cdot\sum_n x^2[n-k]}}. \tag{21.24}$$

The denominator combines both energies involved: $R_{\mathbf{xx}}[0] = \sum_n x^2[n]$, the energy of the signal (Eq. (21.13)), and $\sum_n x^2[n-k]$, the energy of its lagged copy. Taking the square root of their product, instead of dividing by either energy alone, treats the two vectors symmetrically, so that exchanging them leaves $\rho_{\mathbf{xx}}[k]$ unchanged. The result is always bounded,

$$-1 \le \rho_{\mathbf{xx}}[k] \le 1, \tag{21.25}$$

where $\left|\rho_{\mathbf{xx}}[k]\right| = 1$ is reached only if $x[n]$ and $x[n-k]$ are exactly proportional.

Geometric mean and the Cauchy–Schwarz bound (*) The *geometric mean* of two non-negative numbers a and b is $\sqrt{ab}$, the multiplicative counterpart of the arithmetic mean $(a+b)/2$. The denominator of Eq. (21.24) is therefore the geometric mean of the two energies, and the bound Eq. (21.25) is the Cauchy–Schwarz inequality applied to $x[n]$ and $x[n-k]$.

To prove it, note that for any real λ a sum of squares cannot be negative,

$$q(\lambda) = \sum_n \big(\lambda\, x[n] + x[n-k]\big)^2 = \lambda^2 R_{\mathbf{xx}}[0] + 2\lambda\, R_{\mathbf{xx}}[k] + \sum_n x^2[n-k] \;\ge\; 0. \tag{21.26}$$

Assume $R_{\mathbf{xx}}[0] > 0$, so that $q(\lambda)$ is a parabola opening upwards; otherwise $x[n] \equiv 0$ and $\rho_{\mathbf{xx}}[k]$ is undefined. A parabola that is never negative crosses zero at most once, so its discriminant cannot be positive,

$$4R_{\mathbf{xx}}^2[k] - 4R_{\mathbf{xx}}[0]\sum_n x^2[n-k] \;\le\; 0 \quad\Longleftrightarrow\quad \left|R_{\mathbf{xx}}[k]\right| \;\le\; \sqrt{R_{\mathbf{xx}}[0]\sum_n x^2[n-k]}. \tag{21.27}$$

The ACF at any lag therefore never exceeds the geometric mean of the two energies, and dividing by it gives Eq. (21.25).

Reading the value Eq. (21.24) is the sample Pearson correlation coefficient of Sec. 2.2.2, evaluated for the pair of vectors $x[n]$ and $x[n-k]$. The mean subtraction appearing there is unnecessary here, because the signal is assumed zero-mean. Its interpretation carries over unchanged: the sign gives the direction of the linear association and the magnitude its strength, with $\rho_{\mathbf{xx}}[k] = 0$ indicating no linear dependence at that lag. Two points are specific to the lag setting:

- Because the amplitude cancels, $\rho_{\mathbf{xx}}[k]$ is dimensionless and unaffected by the gain of the recording, so values obtained at different lags and from different records are directly comparable.
- Its magnitude fixes the loss achievable by the one-coefficient predictor, as developed in the Model Loss part below.

Relation to the normalized ACF Under the same stationarity assumption used for the unbiased ACF (Eq. (21.19)), $\sum_n x^2[n-k] \approx \frac{L-k}{L} R_{\mathbf{xx}}[0]$, and

$$\rho_{\mathbf{xx}}[k] \approx \sqrt{\frac{L}{L-k}}\, R_{\mathbf{xx},norm}[k] \approx R_{\mathbf{xx},norm}[k] \approx h_k \qquad (k \ll L). \tag{21.28}$$

The second step drops the factor $\sqrt{L/(L-k)}$, which tends to 1 only for $k \ll L$. For a short record or a large lag this factor is not negligible, and Eq. (21.24) has to be evaluated directly.

Sample vs. population coefficient

Eq. (21.24) is a *sample* quantity: it is computed from one finite record of length L, so a different record of the same process yields a slightly different value. The *population* coefficient is the value it converges to,

$$\rho^{\text{pop}}_{\mathbf{xx}}[k] = \lim_{L\to\infty} \rho_{\mathbf{xx}}[k], \tag{21.29}$$

and it is the quantity that a modeling assumption such as whiteness of the noise actually refers to. In practice the sample $\rho_{\mathbf{xx}}[k]$ is used in its place, with $R_{\mathbf{xx},norm}[k]$ as the stationarity-based approximation.

The summary of the different ACF normalizations and their relation to the prediction coefficient h_k is given in Table 21.1. The biased and unbiased ACFs keep the scale of the signal, as their value at $k = 0$ shows, so neither is by itself an approximation of the dimensionless h_k; only their ratio to that lag-0 value is comparable with it.

Table 21.1.: Summary of ACF normalizations and the correlation coefficient $\rho_{\mathbf{xx}}[k]$.

Quantity	Eq.	Norm. factor	Value at $k=0$	Ratio to $k=0$ value	Relation to h_k
$R_{\mathbf{xx}}[k]$ (raw)	(21.11)	none	$E_{\mathbf{x}}$	$R_{\mathbf{xx},norm}[k]$	numerator of h_k
$R_{\mathbf{xx},biased}[k]$	(21.14)	L	$P_{\mathbf{x}}$	$R_{\mathbf{xx},norm}[k]$	only through the ratio
$R_{\mathbf{xx},norm}[k]$	(21.16)	$R_{\mathbf{xx}}[0]$	1	$R_{\mathbf{xx},norm}[k]$	$\lesssim h_k$, underestimates for large k
$R_{\mathbf{xx},unbiased}[k]$	(21.18)	$L-k$	$P_{\mathbf{x}}$	$\frac{L}{L-k} R_{\mathbf{xx},norm}[k]$	$\approx h_k$ through the ratio
$\rho_{\mathbf{xx}}[k]$	(21.24)	geometric mean of energies	1	$\rho_{\mathbf{xx}}[k]$	$\approx h_k$ under stationarity, $\lvert\rho_{\mathbf{xx}}[k]\rvert \le 1$

21.1.3. ACF Properties

The signal energy (21.13) is the highest value of ACF,

$$R_{\mathbf{xx}}[0] > R_{\mathbf{xx}}[k]. \tag{21.30}$$

Theoretically, $R_{\mathbf{xx}}[0] = R_{\mathbf{xx}}[k]$ may happen under certain conditions but unachievable for the practical time-limited signals.

The ACF has inherent time symmetry (backward prediction),

$$R_{\mathbf{xx}}[k] = R_{\mathbf{xx}}[-k] \tag{21.31}$$

that reduces the need to recalculation for $k < 0$.

Minimum Loss

In matrix form (by Eq. (3.21)), the loss is given by

$$\mathcal{L}_{min}(h_k) = \mathbf{e}^T\mathbf{e} = \mathbf{y}^T\mathbf{y} - h_k\mathbf{y}^T\mathbf{x} \tag{21.32}$$

The resulting loss is given by (following Eq. (3.21))

$$\begin{aligned} \mathcal{L}_{min}(h_k) &= \sum_{n=0}^{L-1} x^2[n] - h_k \sum_{n=0}^{L-1} x[n]x[n-k] \\ &= R_{\mathbf{xx}}[0] - h_k R_{\mathbf{xx}}[k]. \end{aligned} \tag{21.33}$$

Since $\sum_n x^2[n-k] \leq R_{\mathbf{xx}}[0]$ implies $h_k \geq R_{\mathbf{xx},norm}[k]$, the loss is upper-bounded by

$$\mathcal{L}_{min}(h_k) \leq R_{\mathbf{xx}}[0]\left(1 - R^2_{\mathbf{xx},norm}[k]\right), \tag{21.34}$$

and, under the stationarity approximation $R_{\mathbf{xx},norm}[k] \approx \rho_{\mathbf{xx}}[k]$ (Eq. (21.24)),

$$\mathcal{L}_{min}(h_k) \approx R_{\mathbf{xx}}[0]\left(1 - \rho^2_{\mathbf{xx}}[k]\right). \tag{21.35}$$

The correlation coefficient $\rho_{\mathbf{xx}}[k]$ is a measure of linear dependence between $x[n]$ and $x[n-k]$.

- For noiseless data and a linear relation between the samples, the prediction is perfect, $\left|\rho_{\mathbf{xx}}[k]\right| = 1$ and $\mathcal{L}_{min} = 0$.
- Without any linear dependence, $\rho_{\mathbf{xx}}[k] = h_k = 0$ and the resulting prediction is $\hat{x}[n] = 0$.

This can be summarized as (see also (21.13) for interpretation)

$$0 \leq \mathcal{L}_{min}(h_k) \leq R_{\mathbf{xx}}[0] \tag{21.36}$$

The illustration of the correlation coefficient principles is presented in Fig. 21.1.

MSE and RMSE The corresponding MSE and RMSE metrics are given by

$$MSE(h_k) = \frac{1}{L}\mathcal{L}_{min}(h_k) \tag{21.37a}$$

$$RMSE(h_k) = \sqrt{\frac{1}{L}\mathcal{L}_{min}(h_k)} \tag{21.37b}$$

21.1.4. Auto-covariance

For simplicity, a zero-average, $\bar{x}[n] = 0$, was assumed. When the signals is non-zero mean, the subtraction of signal average (DC level) from the signal, $x[n] = x[n] - \bar{x}[n]$ before auto-correlation calculation is termed as auto-covariance.

Note, both auto-correlation and auto-covariance have similar abbreviation, ACF. Typically, ACF is used for auto-correlation, since majority of the signals are zero-mean.

Tip:

- ACF calculation may be significantly speed-up with appropriate algorithms and the bounded maximum value of $k \leq k_{max}$. The value of k_{max} may be decided by Eq. (21.22).

21.2. Power Spectral Density (PSD)

21.2.1. Relation between ACF and PSD

From the signal processing point of view, the interpretation of a DFT of some general random signal is non-trivial. For example, phases θ_k of some random origin will be quite different for each time-segment (or realization) of the signal. The Wiener–Khinchin theorem states that the PSD (pages 295 and 301) of stationary random signal is the Fourier transform of the ACF. For the raw ACF $R_{\mathbf{xx}}[m]$, with m denoting the lag,

$$S_{\mathbf{xx}}[k] = \mathrm{DFT}\left\{R_{\mathbf{xx}}[m]\right\} = \left|\mathrm{DFT}\left\{x[n]\right\}\right|^2, \quad k = 0, \ldots, N-1 \tag{21.38}$$

Transforming the biased ACF instead (Eq. (21.14)) scales the result by $1/L$ and gives the raw periodogram $\hat{S}_{\mathbf{xx}}[k] = \left|\mathrm{DFT}\left\{x[n]\right\}\right|^2/L$ used in Sec. 20.7.4; the two differ only by that constant.

Important properties are

- $S_{\mathbf{xx}}[k] \in \mathscr{R}$, due to the symmetry (Eq. (21.31)) of $R_{\mathbf{xx}}[k]$ it results $\theta_k = 0 \forall k$.
- $S_{\mathbf{xx}}[k] \geq 0 \quad \forall k$.

This relation shows, that random signal also includes spectral interpretation.

Example 21.1: Three signals of equal power are compared in Fig. 21.3, each with its ACF and the PSD obtained from it.

- *White noise*: the ACF is non-zero only at $m = 0$, and the PSD is flat. A signal that is uncorrelated with its own past spreads its power evenly over all frequencies.
- *Correlated noise*: the ACF decays slowly, and the power concentrates at low frequency. The slower the decay, that is the longer the correlation time (Eq. (21.21)), the narrower the peak.
- *Sinusoid in noise*: the ACF oscillates at the frequency of the sinusoid, and the PSD has a peak there.

Since all three signals carry the same total power, the panels differ only in how that power is spread over frequency.

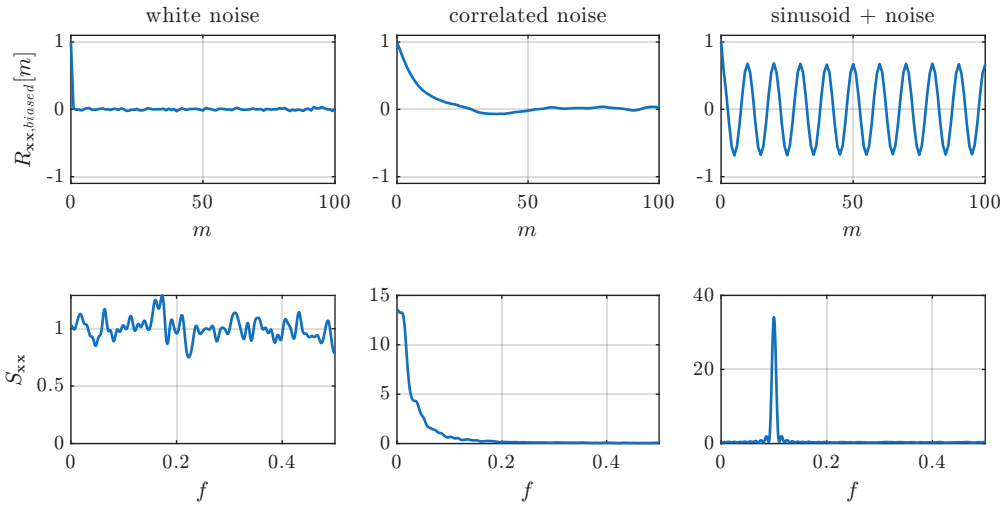


Figure 21.3.: Transform pairs of the Wiener–Khinchin theorem. Top: the biased ACF. Bottom: its transform, Eq. (21.38), computed from the very lags plotted above it. All three signals are normalized to unit variance, so the panels differ only in how the power is distributed over frequency. Left: white noise, a single non-zero lag and a flat PSD. Center: correlated noise, a slowly decaying ACF and power concentrated at low frequency. Right: a sinusoid in noise, an oscillating ACF and a spectral peak at its frequency. ACF is truncated to 100 lags.

Interpretation

Peaks The PSD shows where (in frequency) the signal has most of its energy. Peaks in the PSD correspond to frequencies at which the signal exhibits strong periodic or quasi-periodic components.

Periodicity Slowly decaying (long-memory) correlations in the time domain often translate into a PSD that has more energy at low frequencies (indicating slow variations in time), as in the center column of Fig. 21.3. Conversely, if the ACF shows periodicity, the PSD will have distinct peaks at the corresponding harmonic frequencies, as in the right column.

Time-shifts The ACF reveals how similar a signal is to itself at different time shifts. A slowly decaying ACF indicates strong long-term correlations, while a rapidly decaying ACF suggests only short-range predictability.

21.2.2. Frequency estimation via ACF peak

Goal: Estimate the dominant frequency of a (quasi-)periodic signal directly from the location of a peak of the auto-correlation function.

For a (quasi-)periodic signal, the ACF $R_{\mathbf{xx}}[k]$ inherits the period of $x[n]$: its largest non-trivial peak appears at the lag corresponding to one signal period. This provides a time-domain alternative to the periodogram-peak estimator of Sec. 20.4.

Using the biased ACF $R_{\mathbf{xx},biased}[k]$ (Eq. (21.14)), the estimator is

$$\hat{k}_0 = \arg\max_{k \geq k_{\min}} R_{\mathbf{xx},biased}[k], \quad \hat{F}_0 = \frac{F_s}{\hat{k}_0}, \quad \hat{\omega}_0 = \frac{2\pi}{\hat{k}_0}. \tag{21.39}$$

The lower bound $k_{\min}$ excludes the trivial peak at $k = 0$ (where $R_{\mathbf{xx}}[0] = E_{\mathbf{x}}$) and the surrounding main lobe. A practical choice is to set $k_{\min}$ just above the ACF main-lobe width, or to start the search at the first zero-crossing (or first local minimum) of $R_{\mathbf{xx}}[k]$.

Example 21.2: A sinusoid of $F_0 = 50$ Hz in additive noise is sampled at $F_s = 1000$ Hz, giving $L = 500$ samples. The biased ACF of Fig. 21.4 has its first non-trivial peak at $\hat{k}_0 = 20$ lags, so

$$\hat{F}_0 = \frac{F_s}{\hat{k}_0} = \frac{1000}{20} = 50 \text{ Hz}, \tag{21.40}$$

which recovers the true frequency. Equivalently, one period of the signal spans 20 samples.

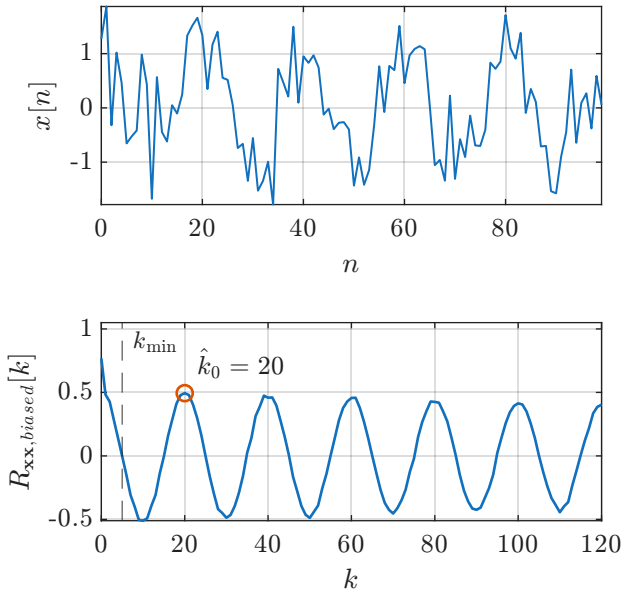


Figure 21.4.: Frequency estimation from the ACF peak. Top: the first 100 samples of $x[n]$. Bottom: the biased ACF; lags below $k_{\min}$ are excluded from the search, and the first peak above it, at $\hat{k}_0 = 20$, gives $\hat{F}_0 = F_s/\hat{k}_0 = 50$ Hz. Note the slow decay of the peak heights, caused by the $1/L$ normalization of the biased ACF.

Why the biased ACF The biased form is preferred because the unbiased ACF $R_{\mathbf{xx},unbiased}[k]$ divides by $L - k$ (Eq. (21.18)) and so inflates the variance for large k, producing spurious peaks near the tail. The biased ACF uses the constant denominator L, trading a small downward bias for a much better-behaved peak search.

ACF-peak vs. periodogram-peak

The two estimators are Wiener-Khinchin duals: peaks of $S_{\mathbf{xx}}[k]$ and peaks of $R_{\mathbf{xx}}[k]$ encode the same periodicity, sampled on the frequency axis or the lag axis respectively.

- *Periodogram-peak* (Eq. (20.47)): searches a (possibly zero-padded) frequency grid; high frequency resolution but sensitive to spectral leakage and windowing.

- *ACF-peak* (Eq. (21.39)): integer-lag resolution, $\Delta F \approx F_s/\hat{k}_0^2$, coarse for small $\hat{k}_0$ (high frequencies) and fine for large $\hat{k}_0$ (low frequencies).

Example 21.3: A signal contains only the harmonics $2F_0$, $3F_0$ and $4F_0$ of $F_0 = 50$ Hz, with amplitudes decaying in that order; the fundamental itself is absent. Its periodogram, the top panel of Fig. 21.5, has lines at 100, 150 and 200 Hz and nothing at 50 Hz, so a periodogram-peak search returns the strongest line, 100 Hz, which is twice the true value.

The sum of the harmonics is nevertheless periodic with period $1/F_0$, since 50 Hz is the greatest common divisor of the three frequencies. Its biased ACF, the bottom panel, therefore peaks at $\hat{k}_0 = 20$ and returns the correct $\hat{F}_0 = 50$ Hz.

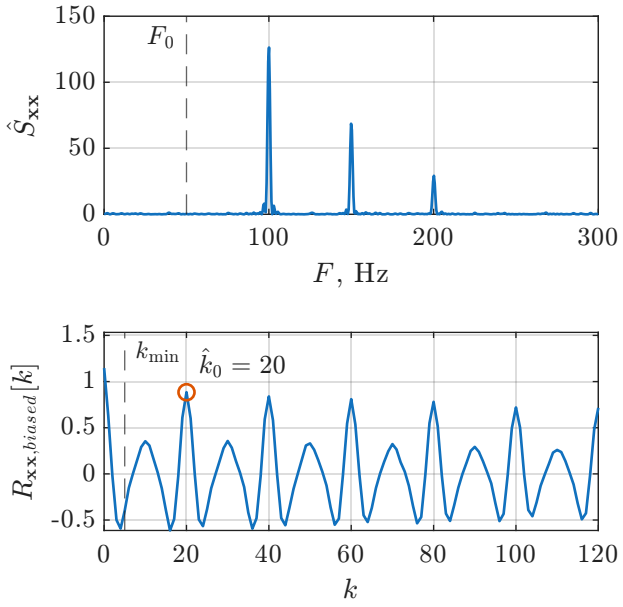


Figure 21.5.: Missing fundamental. Top: the periodogram has no line at F_0, so its peak lies at $2F_0 = 100$ Hz. Bottom: the biased ACF still peaks at the true period, $\hat{k}_0 = 20$ lags. The competing peak at $k = 40$ is equally valid in principle, and is rejected only because the biased normalization lowers it.

This is the *missing fundamental* case, and it is why the autocorrelation method is the canonical pitch-detection approach in speech and audio processing, where the fundamental period is read off as $\hat{k}_0$. Selecting a peak in the frequency domain instead risks returning a harmonic, an error of a whole octave for the $2F_0$ line.

Tip:

- Sub-sample lag refinement: fit a parabola to $R_{\mathbf{xx},biased}[\hat{k}_0 - 1], R_{\mathbf{xx},biased}[\hat{k}_0], R_{\mathbf{xx},biased}[\hat{k}_0 + 1]$ and use its analytic vertex as a fractional-lag estimate, analogous to parabolic peak of the periodogram in (20.48).

21.3. AR(p) Model

Goal: Extend $AR(1)$ to $AR(p)$ model.

The auto-regressive (AR) *signal* model describes a signal $y[n]$ as a linear combination of its p previous samples plus noise. Formally, an $AR(p)$ model is

$$\begin{aligned} y[n] &= h_1 y[n-1] + h_2 y[n-2] + \cdots + h_p y[n-p] + \epsilon[n] \\ &= \sum_{m=1}^{p} h_m y[n-m] + \epsilon[n] \end{aligned} \tag{21.41}$$

and the prediction is of the form

$$\hat{y}[n] = \sum_{m=1}^{p} h_m y[n-m] \tag{21.42}$$

where $h_1, \ldots, h_p$ are the AR model coefficients and p is the model order that is chosen as *hyper-parameter*.

This model can be easily formulated in the matrix form by using L sample of $y[n]$, by

$$\hat{\mathbf{y}} = \begin{bmatrix} \hat{y}[1] \\ \hat{y}[2] \\ \vdots \\ \hat{y}[L-2] \\ \hat{y}[L-1] \end{bmatrix}, \; \mathbf{X} = \begin{bmatrix} y[0] & 0 & \cdots & 0 \\ y[1] & y[0] & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ y[L-3] & y[L-4] & \cdots & y[L-p-2] \\ y[L-2] & y[L-3] & \cdots & y[L-p-1] \end{bmatrix}, \; \mathbf{h} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_p \end{bmatrix} \tag{21.43}$$

where the corresponding dimensions are $\hat{\mathbf{y}} \in \mathscr{R}^{L-1}, \mathbf{X} \in \mathscr{R}^{L-1 \times p}, \mathbf{h} \in \mathscr{R}^{p}$.

The AR coefficients $\mathbf{h}$ can be found by a LS regression that minimizes the MSE loss

$$\mathcal{L}(h_i) = \sum_{n} \big(y[n] - \hat{y}[n]\big)^2 = \|\mathbf{y} - \mathbf{X}\mathbf{h}\|^2. \tag{21.44}$$

The LS solution is straightforward,

$$\mathbf{h} = \Big(\mathbf{X}^T \mathbf{X}\Big)^{-1} \mathbf{X}^T \mathbf{y}. \tag{21.45}$$

Note, the general approach for evaluation of $\mathbf{h}$ is presented in the following section.

Example 21.4: Learn linear prediction of $y[8]$ for $p = 3$ and signal $y[0], y[1], \ldots, y[7]$.

Solution:

$$\mathbf{X} = \begin{bmatrix} y[0] & 0 & 0 \\ y[1] & y[0] & 0 \\ y[2] & y[1] & y[0] \\ y[3] & y[2] & y[1] \\ y[4] & y[3] & y[2] \\ y[5] & y[4] & y[3] \\ y[6] & y[5] & y[4] \end{bmatrix}, \; \mathbf{h} = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} \; \mathbf{y} = \begin{bmatrix} y[1] \\ y[2] \\ y[3] \\ y[4] \\ y[5] \\ y[6] \\ y[7] \end{bmatrix} \tag{21.46}$$

Finding vector $\mathbf{h}$ values is (almost) trivial by a minimum of the loss function of the form

$$\mathcal{L} = \|\mathbf{y} - \mathbf{X}\mathbf{h}\|^2. \tag{21.47}$$

Once found, $\hat{y}[8] = h_1 y[7] + h_2 y[6] + h_3 y[5]$.

21.3.1. Yule-Walker Form

The resulting $\mathbf{X}^T\mathbf{X}$ matrix is termed (Toeplitz) auto-correlation matrix and can be interpreted in terms of auto-correlation values,

$$\begin{aligned}\mathbf{R} &= \mathbf{X}^T\mathbf{X} \\ &= \begin{bmatrix} R_{\mathbf{yy}}[0] & R_{\mathbf{yy}}[1] & \cdots & R_{\mathbf{yy}}[p-1] \\ R_{\mathbf{yy}}[1] & R_{\mathbf{yy}}[0] & \cdots & R_{\mathbf{yy}}[p-2] \\ \vdots & \vdots & \ddots & \vdots \\ R_{\mathbf{yy}}[p-1] & R_{\mathbf{yy}}[p-2] & \cdots & R_{\mathbf{yy}}[0] \end{bmatrix},\end{aligned} \tag{21.48}$$

where unscaled (or biased) $R_{\mathbf{yy}}[k]$ is defined above. It is common practice to ignore the changes in the vector lengths in calculating (biased) ACF for the same time-shift for $p \ll L$, e.g. diagonal matrix values are

$$R_{\mathbf{yy}}[0] = \sum_{n=0}^{L-1} y^2[n] \approx \sum_{n=0}^{L-\mathbf{2}} y^2[n] \tag{21.49}$$

The vector $\mathbf{X}^T\mathbf{y}$ is also comprised of the corresponding $R_{\mathbf{yy}}[k]$ values,

$$\mathbf{r} = \mathbf{X}^T\mathbf{y} = \begin{bmatrix} R_{\mathbf{yy}}[1] \\ R_{\mathbf{yy}}[2] \\ \vdots \\ R_{\mathbf{yy}}[p] \end{bmatrix} \tag{21.50}$$

Using these formulations, the resulting coefficients are given by a solution of a set of linear equations,

$$\mathbf{Rh} = \mathbf{r} \tag{21.51}$$

and the result is similar to Eq. (21.45),

$$\mathbf{h} = \mathbf{R}^{-1}\mathbf{r} \tag{21.52}$$

The use of a $R_{\mathbf{xx},biased}[k]$ (biased ACF, Eq. (21.14)) guaranties the numerically stable solution [15, Theorem 6.1].

Squared error The resulting loss (squared error) is given by (following Eqs. (3.21) and (21.33))

$$\mathcal{L}_{min} = R_{\mathbf{xx}}[0] - \sum_{k=1}^{p} h_k R_{\mathbf{xx}}[k] \tag{21.53}$$

Theoretically, higher value of p results in lower loss in the presence of the sufficiently long correlation time (Eq. (21.21)). Note, the accuracy drops for high values of p to due to reduced accuracy of $R_{\mathbf{xx}}[p]$.

Example 21.5: A synthetic AR(3) signal of length $L = 1000$ is generated by driving the model with white noise of variance $\sigma_\epsilon^2 = 1$ and the true coefficients $\mathbf{h} = (1.2, -0.9, 0.6)$.

Solving Eq. (21.45) at the correct order, $p = 3$, returns $\hat{\mathbf{h}} = (1.19, -0.88, 0.57)$, within a few percent of the true values, and the resulting one-step prediction follows the signal closely (top panel of Fig. 21.6).

Repeating the fit for $p = 1, \ldots, 7$ gives the MSE, $\mathcal{L}_{min}/L$, of the bottom panel. It falls steeply while p is below the true order, from 1.57 at $p = 1$ to 0.96 at $p = 3$, and is flat afterwards, so the elbow of the curve marks the model order.

The flat part sits at $\approx \sigma_\epsilon^2$, the variance of the driving noise. Once every predictable component is captured, the residual is the noise itself, and no additional coefficient can lower the loss below it. The coefficients returned for $p > 3$ are indeed close to zero, and their estimates rely on $R_{\mathbf{yy}}[k]$ at the largest lags, where the accuracy is lowest.

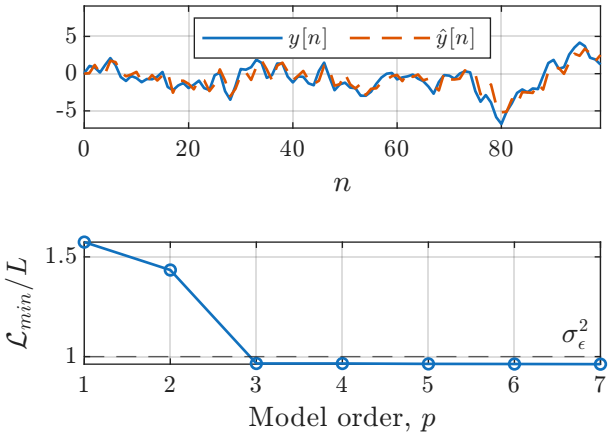


Figure 21.6.: LS fit of an AR(3) signal, $\mathbf{h} = (1.2, -0.9, 0.6)$ and $L = 1000$. Top: the first 100 samples of $y[n]$ and the one-step prediction $\hat{y}[n]$ of the $p = 3$ model. Bottom: the MSE as a function of the model order p. The curve flattens once p reaches the true order, at the variance σ_ϵ^2 of the driving noise.

The commonly adopted notation of the signal model in most of the books and software packages is

$$\sum_{m=0}^{p} a_m y[n-m] = \epsilon[n] \tag{21.54}$$

with $a_0 = 1$ and $a_m = -h_m$.
This definition follows discrete-time definition and the corresponding Z-transform of all-pole system. It also can be directly applied as a denominator coefficients for `filter` command.

Biased signal The presence of a non-zero average (bias) in the signal modifies the AR model formulation. Instead of assuming a zero-mean process, a constant bias term μ is introduced,

$$\hat{y}[n] = \mu + h_1 y[n-1] + h_2 y[n-2] + \cdots + h_p y[n-p] + \epsilon[n] \tag{21.55}$$

In matrix form, this requires adding a column of ones $\mathbf{1}_M$ to the data matrix $\mathbf{X}$, similarly to how it is done in multivariate LS (Eq.(3.7)).

Tips:

- For computational and memory efficiency, the Levinson-Durbin algorithm $O\left(L^2\right)$ is often used to solve for AR coefficients [1], instead of direct $O\left(L^3\right)$.

[1] For example, `solve_toeplitz`

- The AR parameters $\mathbf{h}$ are also referred to as linear prediction coefficients (LPC), emphasizing their role in predicting the current value from past samples.
- The auto-correlation matrix $\mathbf{R}$ is guaranteed to be positive-definite (and, therefore, invertible) for ordinary or biased definitions (normalization constants reduce each other in (21.45)).
- Theoretically, the AR model coefficients can be regularized to prevent overfitting.
- Although this section focuses on linear AR models, nonlinear variants also exist, allowing modeling of more complex dynamics.
- Note, there are other methods for LPC calculation, such as Burg's method, as it is done in `librosa.lpc`.

21.3.2. Moving Average Filter

A simple special case of an AR-like model is the moving average filter, where all coefficients are equal and sum to one, $h_i = \frac{1}{p}$. In this case,

$$\begin{aligned} \hat{y}[n] &= \frac{1}{p}\left(y[n-1] + \cdots + y[n-p]\right) \\ &= \frac{1}{p}\sum_{m=1}^{p} y[n-m] \end{aligned} \tag{21.56}$$

This is not strictly an AR model but shares the idea of using past samples. It smooths the signal by averaging the most recent p values.

21.3.3. Nearest Neighbor (Naïve)

A simple baseline is to use the immediate past sample as the prediction with $h_1 = 1$,

$$\hat{y}[n] = y[n-1] \tag{21.57}$$

This "1-nearest neighbor" approach ignores signal dynamics and history. While often not very accurate, it serves as a useful baseline to compare against more sophisticated models.

Example 21.6 (Naive baseline vs. AR(1)): The naive predictor differs from the AR(1) model of Eq. (21.1) in one respect only: it forces $h_1 = 1$ instead of fitting it. The cost of that shortcut follows from the ACF. Its error power is

$$\sum_n \big(x[n] - x[n-1]\big)^2 = 2\left(R_{\mathbf{xx}}[0] - R_{\mathbf{xx}}[1]\right) = 2R_{\mathbf{xx}}[0]\left(1 - R_{\mathbf{xx},norm}[1]\right), \tag{21.58}$$

while the fitted coefficient attains $\mathcal{L}_{min}(h_1) = R_{\mathbf{xx}}[0]\left(1 - R^2_{\mathbf{xx},norm}[1]\right)$ by Eq. (21.33). Writing $h_1 \approx R_{\mathbf{xx},norm}[1]$, their ratio is

$$\frac{\mathrm{MSE}_{naive}}{\mathrm{MSE}_{min}} = \frac{2\,(1-h_1)}{1-h_1^2} = \frac{2}{1+h_1}. \tag{21.59}$$

The penalty is therefore governed by h_1 alone, Fig. 21.7. For $h_1 \to 1$ the ratio tends to 1 and the naive predictor is as good as the fitted one. For $h_1 = 0.5$ it costs 4/3 in MSE, some 15% in

RMSE, and for a white signal, $h_1 = 0$, it doubles the error power, whereas the best one-coefficient prediction is $\hat{x}[n] = 0$. For $h_1 < 0$, that is a signal alternating in sign, it is worse still.

For the record of Fig. 21.1, with $h_1 = 0.9577$, the fitted predictor reaches RMSE $= 0.999$ and the naive one RMSE $= 1.010$, a gap of 1.1%, as predicted by Eq. (21.59).

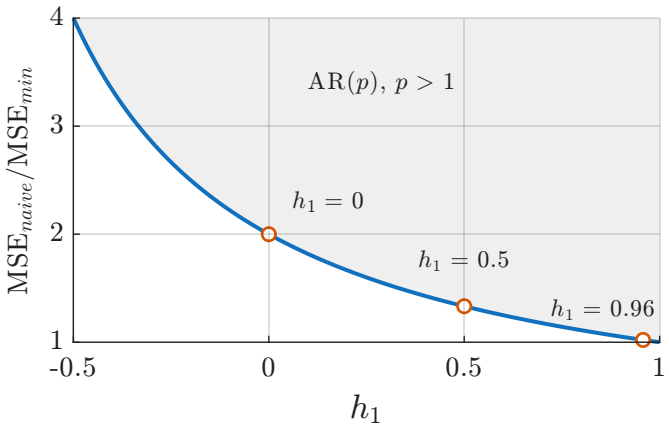


Figure 21.7.: The cost of the naive baseline, Eq. (21.59): the ratio of its MSE to that of the fitted one-coefficient predictor, as a function of h_1. The marked cases are a white signal, $h_1 = 0$, an intermediate one, $h_1 = 0.5$, and the record of Fig. 21.1, $h_1 = 0.96$. The curve is a lower bound, and the penalty against an AR(p) model with $p > 1$ lies in the shaded region above it.

A strongly correlated signal makes the naive predictor hard to beat: at $h_1 \approx 0.96$ it is within about 1% of the best achievable one-coefficient prediction. Reporting that a model beats the naive baseline by such a margin demonstrates almost nothing about the model.

21.4. Linear Prediction of Sinusoidal Signal

Goal: This section demonstrates that a second-order auto-regressive (AR(2)) model can represent a pure sinusoidal signal perfectly, predicting it without error from just the two previous samples.

Consider the noise-free sinusoidal signal

$$y[n] = \cos(\omega_0 n) \tag{21.60}$$

For simplicity, phase is initially taken as zero. We aim to model it with

$$\hat{y}[n+1] = h_0 y[n] + h_1 y[n-1] \tag{21.61}$$

The corresponding loss is

$$\begin{aligned} \mathcal{L}(h_0, h_1) &= \sum_n \big(y[n+1] - \hat{y}[n+1]\big)^2 \\ &= \sum_n \big(y[n+1] - h_0 y[n] - h_1 y[n-1]\big)^2 \end{aligned} \tag{21.62}$$

The required minimum is given by a solution of a system of normal equations,

$$\begin{cases} \dfrac{\partial}{\partial h_0}\mathcal{L}(h_0, h_1) = 0 \\ \dfrac{\partial}{\partial h_1}\mathcal{L}(h_0, h_1) = 0 \end{cases} \tag{21.63}$$

The resulting equations are

$$\begin{cases} 2\sum\limits_n \big(y[n+1] - h_0 y[n] - h_1 y[n-1]\big) \cdot (-y[n]) = 0 \\ 2\sum\limits_n \big(y[n+1] - h_0 y[n] - h_1 y[n-1]\big) \cdot (-y[n-1]) = 0 \end{cases} \tag{21.64}$$

These equations involve sums of trigonometrical functions. Using standard trigonometric identities and assuming a large number of samples L, these sums simplify due to the oscillatory nature of sine and cosine functions.

Some important quantities are

$$\begin{aligned} y[n+1]y[n] &= \cos\big(\omega_0[n]\big)\cos\big(\omega_0[n+1]\big) \\ &= \cos(\omega_0 n)\left[\cos(\omega_0)\cos(\omega_0 n) - \sin(\omega_0)\sin(\omega_0 n)\right] \\ &= \cos(\omega_0)\cos^2(\omega_0 n) - \sin(\omega_0)\cos(\omega_0 n)\sin(\omega_0 n) \end{aligned} \tag{21.65}$$

For $1/\omega_0 \ll L$, the common assumption is

$$\begin{aligned} \sum_n \cos^2(\omega_0 n) &\approx \frac{L}{2} \\ \sum_n \cos(\omega_0 n)\sin(\omega_0 n) &\approx 0 \\ \sum_n y[n+1]y[n] &\approx \frac{L}{2}\cos(\omega_0) \\ &\approx \sum_n y[n]y[n-1] \\ \sum_n y^2[n] &= \sum_n \cos^2(\omega_0 n) \\ &\approx \sum_n y^2[n-1] \approx \frac{L}{2} \\ \sum_n y[n+1]y[n-1] &= \frac{1}{2}\sum_n \cos(2\omega_0 n) + \frac{1}{2}\sum_n \cos(2\omega_0) \\ &\approx \frac{L}{2}\cos(2\omega_0) \end{aligned}$$

The approximations show that sum terms reduce to manageable forms. Substituting these approximations into the normal equations leads to a system

$$\begin{aligned} h_0\frac{L}{2} + h_1\frac{L}{2}\cos(\omega_0) &= \frac{L}{2}\cos(\omega_0) \\ h_0\frac{L}{2}\cos(\omega_0) + h_1\frac{L}{2} &= \frac{L}{2}\cos(2\omega_0) \end{aligned} \tag{21.66}$$

Solving these two linear equations for prediction coefficients yields

$$\begin{aligned} h_0 &= 2\cos(\omega_0) \\ h_1 &= -1 \end{aligned} \tag{21.67}$$

This result is exact in the idealized scenario of an infinite, noise-free sinusoid.

If the signal had a phase θ,

$$y[n] = \cos(\omega_0 n + \theta) \tag{21.68}$$

the same coefficients still apply, only the initial conditions (the first two samples) differ,

$$\begin{aligned} y[0] &= \cos(\theta) \\ y[1] &= \cos(\omega_0 + \theta). \end{aligned} \tag{21.69}$$

The AR(2) model still perfectly represents the sinusoidal sequence. The example of the resulting signal is presented in Fig. 21.8.

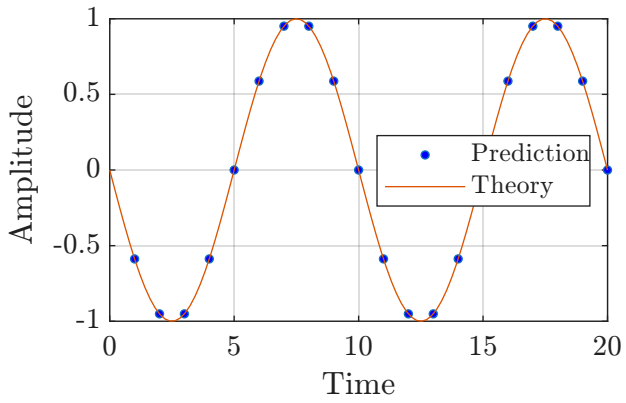


Figure 21.8.: Oscillator example.

Results interpretation:

- Any complex signal can be thought of as a sum of sinusoids. In theory, each sinusoidal component can be perfectly modeled by an AR(2) process corresponding to its frequency.
- In practice, signals are finite in length and often corrupted by noise. Under these realistic conditions, perfect prediction is not achievable, and the AR model's accuracy diminishes as noise and non-idealities increase. The prediction horizon is limited even for a periodic low-noise signals.
- Nevertheless, this analysis provides a foundational insight: sinusoids have a natural AR(p) representation, explaining why AR(p) models are often effective in capturing periodic components of signals for a small-step prediction.

21.5. Partial auto-correlation function

Goal: The partial autocorrelation function (PACF) measures the correlation between a time series and its lagged values after removing the influence of all shorter lags. In other words, the PACF at lag k shows the direct effect of $y[n-k]$ on $y[n]$, excluding any intermediary correlations through lags less than k.

The partial autocorrelation at k is the correlation that results after removing the effect of any correlations due to the terms at shorter lags,

$$\underline{x[0]}, \underbrace{\underline{x[1]}, x[2], \ldots, x[j-1]}_{\text{partial out}}, \underline{x[j]}, x[j+1], \ldots$$

The PACF at lag k can be extracted by fitting an AR(k) model and observing the coefficient associated with $y[n-k]$ in this model[2]. For each k,

$$\hat{y}_k[n] = \phi_{k,1} y[n-1] + \phi_{k,2} y[n-2] + ... + \phi_{k,k} y[n-k] + \epsilon[n] \tag{21.70}$$

The k-th partial autocorrelation, $\beta[k]$ is $\phi_{k,k}$, the coefficient of $y[n-k]$ in the AR(k) model. Each of these AR(k) models can be solved using standard LS methods.

The algorithm is as follows:

- For the first value of PACF, $\beta[1]$, fit AR(1) model

$$\hat{y}_1[n] = \phi_{1,1} y[n-1] + \epsilon[n] \tag{21.71}$$

  and the coefficient $\beta[1] = \hat{\phi}_{1,1}$ is given by the model solution, h_1 (Eq. (21.5)).
- For the second order PACF, it would be the $\beta[2] = \phi_{2,2}$ coefficient of AR(2) model,

$$\hat{y}_2[n] = \phi_{2,1} y[n-1] + \phi_{2,2} y[n-2] + \epsilon[n] \tag{21.72}$$

- Continue this process up to the desired lag, each time extracting the coefficient of the highest-order lag as the PACF value.

Tip:

- While the ACF reflects both direct and indirect correlations (where an earlier lag may influence a later lag through intermediate values), the PACF isolates the direct contribution of each individual lag once all shorter-term effects are factored out.
- Though conceptually the PACF is obtained by fitting multiple AR models, practically more efficient algorithms like the Levinson-Durbin recursion can compute these values quickly and without having to solve a new LS problem at every step.

21.5.1. Relation between PACF and AR(p)

The order p of AR(p) model is related to the statistically significant (over a confidence bound) coefficients of PACF. Observing where the PACF cuts off helps identify the appropriate order p for AR(p) model fitting. An example of a synthetic signal analysis with $p = 2$ is presented in Fig. 21.9.

21.6. MA model

Goal: The moving average (MA) model expresses the current output $y[n]$ as a linear combination of current and past noise terms. Unlike the AR model, which relies on past values of the output, the MA model directly models the output as filtered white noise.

[2] Stackexchange

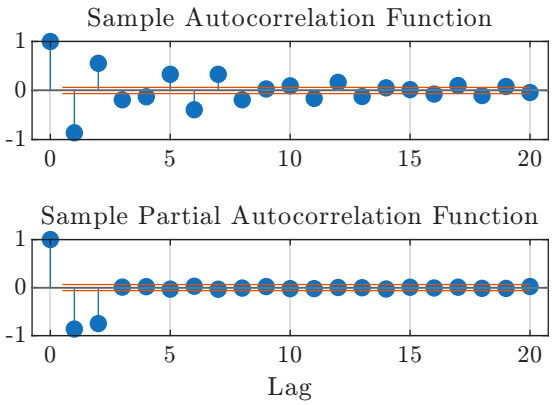


Figure 21.9.: ACF and PACF of AR(2) signal. Note short PACF and long ACF plots.

Another model is the moving average model (MA), where the output is a linear combination of the noise values at the different times [12, Example 4.3, pp. 90]

$$y[n] = \epsilon[n] + c_1\epsilon[n-1] + \cdots + c_q\epsilon[n-q] \tag{21.73}$$

Because the noise terms $\epsilon[n]$ are not directly observable (unlike the past outputs in AR modeling), deriving a closed-form solution for MA parameters through simple linear regression is not as direct and is not presented here. [3] While the underlying theory is well-established, practical estimation of MA parameters often involves advanced numerical methods rather than a simple closed-form solution.

21.6.1. MA and AR relations

There exists a duality between AR and MA processes in terms of infinite expansions:

Presenting AR(p) as MA(∞) An AR model can be represented as an infinite MA model when the autoregressive parameters are stable. Consider the recursive simple AR(1),

$$\begin{aligned} y[n] &= h_1 y[n-1] + \epsilon[n] \\ &= h_1(h_1 y[n-2] + \epsilon[n-1]) + \epsilon[n] \\ &= h_1^2 y[n-2] + h_1\epsilon[n-1] + \epsilon[n] \\ &= h_1^3 y[n-3] + h_1^2\epsilon[n-2] + h_1\epsilon[n-1] + \epsilon[n] \end{aligned} \tag{21.74}$$

Continuing this process indefinitely and assuming stability, $h_1 < 1$, $\lim_{k\to\infty} h_1^k \to 0$, we have

$$\begin{aligned} y[n] &= \epsilon[n] + h_1\epsilon[n-1] + h_1^2\epsilon[n-2] + h_1^3\epsilon[n-3] + \cdots \\ &= \sum_{i=0}^{\infty} h_1^i \epsilon[n-i] \end{aligned} \tag{21.75}$$

that it is MA(∞) model, where the coefficients of the MA representation are the infinite geometric sequence powers of h_1.

[3] e.g. A simple Estimator of an MA(1) Models.

Presenting MA(1) as AR(∞) An MA(1) process is given by

$$x[n] = \epsilon[n] + c_1\,\epsilon[n-1]. \tag{21.76}$$

We can express the process as an infinite-order AR(∞) model.

First, note that we can write

$$\begin{aligned}\epsilon[n] &= x[n] - c_1\,\epsilon[n-1].\\ \epsilon[n-1] &= x[n-1] - c_1\,\epsilon[n-2]\\ \epsilon[n-2] &= x[n-2] - c_1\,\epsilon[n-3]\end{aligned}$$

and so on. Substituting for $\epsilon[n-i]$ recursively, we have

$$\begin{aligned}\epsilon[n] &= x[n] - c_1\left(x[n-1] - c_1\,\epsilon[n-2]\right)\\ &= x[n] - c_1\,x[n-1] + c_1^2\,\epsilon[n-2]\\ &= x[n] - c_1\,x[n-1] + c_1^2\left(x[n-2] - c_1\,\epsilon[n-3]\right)\\ &= x[n] - c_1\,x[n-1] + c_1^2\,x[n-2] - c_1^3\,\epsilon[n-3]\\ &\vdots\\ &= \sum_{i=0}^{\infty}(-c_1)^i\,x[n-i].\end{aligned} \tag{21.77}$$

Rearranging, the AR(∞) representation becomes

$$x[n] = \epsilon[n] + \sum_{i=1}^{\infty}(-c_1)^i\,x[n-i]. \tag{21.78}$$

Note the invertibility condition, $|c_1| < 1$.

Interpretation AR and MA are not mutually exclusive categories. A stable AR process can be seen as a special case of an infinite MA, and a stable MA can be thought of as an infinite AR.

21.6.2. The relation between MA(q) and ACF

The MA(q) model has a distinctive fingerprint in terms of its ACF; it is nonzero for up to lag q and essentially zero afterward (except for sampling and noise effects). Just as the partial autocorrelation function (PACF) helps determine the order p of an AR(p) process by pinpointing where its PACF cuts off (Sec. 21.5.1), the ACF helps identify the order q of an MA(q) process.

An example of a synthetic MA(4) signal analysis is presented in Fig. 21.10. After lag 4, the ACF values remain within the confidence bounds, essentially zero, suggesting the data arise from an MA(4) process.

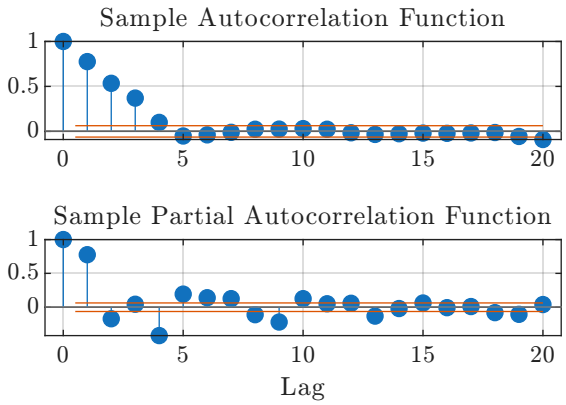


Figure 21.10.: ACF and PACF of a synthetic MA(4) signal. Note short ACF and long PACF plots.

21.7. ARMA

Goal: The ARMA model extends the concepts of AR and MA models by combining their elements to capture more complex dynamics.

The ARMA(p,q) model is given by

$$\begin{aligned}\hat{y}[n] &= h_1 y[n-1] + \cdots + h_p y[n-p] \\ &\quad + c_1 \epsilon[n-1] + \cdots + c_q \epsilon[n-q] + \epsilon[n] \\ &= \sum_{i=1}^{p} h_i y[n-i] + \sum_{k=0}^{q} c_k \epsilon[n-k]\end{aligned}, \tag{21.79}$$

where $c_0 = 1$ by definition.

One of the ways to describe MA part of ARMA, is that MA uses to model the difference unexplained by AR model,

$$\hat{y}[n] - \sum_{i=1}^{p} h_i y[n-i] = \sum_{k=0}^{q} c_k \epsilon[n-k] \tag{21.80}$$

Innovation

The *innovation* of a signal is the part of the current sample that the signal's own past cannot predict, that is, the error of the best linear one-step prediction of $y[n]$ from $y[n-1], y[n-2], \ldots$ Everything predictable has been taken out of it, so it is white, which is what makes it the natural driving noise of a model of that signal.
Its variance is the floor of the loss. It is what remains once every coefficient has done its work, and no choice of orders reaches below it. The driving noise of a model is thus the innovation of the signal the model generates, and a signal carrying more than one noise source has an innovation that is none of them separately, as the following example shows.

The MA part is not merely a modeling convenience. The following example shows a routine measurement setup in which it appears by itself, with no choice involved.

Example 21.7: An AR(1) signal is recorded by a noisy sensor. The hidden state follows

$$y[n] = h_1 y[n-1] + \epsilon[n], \quad |h_1| < 1, \tag{21.81}$$

and the sensor adds a white measurement noise $v[n]$ of variance σ_v^2, independent of the driving noise $\epsilon[n]$,

$$z[n] = y[n] + v[n]. \tag{21.82}$$

Show that $z[n]$ is an ARMA(1,1) process. The ACF values below are the per-sample averaged form of Eq. (21.14).

Solution: Apply the AR(1) difference of Eq. (21.81) to the *observed* signal and substitute Eq. (21.82),

$$u[n] \triangleq z[n] - h_1 z[n-1] = \epsilon[n] + v[n] - h_1 v[n-1], \tag{21.83}$$

using $y[n] - h_1 y[n-1] = \epsilon[n]$. The ACF of $u[n]$ is the per-sample average of the product $u[n]u[n-k]$. Both noises are white, zero-mean and mutually independent, so such a product averages to zero unless it is the square of one and the same noise sample. At lag 0 the three squares $\epsilon^2[n]$, $v^2[n]$ and $h_1^2 v^2[n-1]$ survive. At lag 1 the two factors share the single sample $v[n-1]$, which enters $u[n]$ as $-h_1 v[n-1]$ and $u[n-1]$ as $v[n-1]$. At $|k| \geq 2$ they share no sample at all, since $u[n]$ carries only the times n and $n-1$. Hence

$$\begin{aligned} R_{\mathbf{uu}}[0] &= \frac{1}{L}\sum_n u^2[n] = \sigma_\epsilon^2 + \sigma_v^2\left(1 + h_1^2\right) \\ R_{\mathbf{uu}}[1] &= \frac{1}{L}\sum_n u[n]u[n-1] = -h_1\sigma_v^2 \\ R_{\mathbf{uu}}[k] &= 0, \quad |k| \geq 2. \end{aligned} \tag{21.84}$$

An ACF vanishing beyond lag 1 is exactly the MA(1) fingerprint of Sec. 21.6.2. The reverse reading is the one used here, and it holds because the MA(1) family is rich enough to match Eq. (21.84): it has two free parameters, c_1 and σ_η^2, and its ACF has exactly two free values, at lags 0 and 1, with all higher lags zero, which is the shape found above. The representation is thus

$$u[n] = \eta[n] + c_1\eta[n-1], \tag{21.85}$$

where $\eta[n]$ is white of variance σ_η^2. It is neither $\epsilon[n]$ nor $v[n]$, but the innovation of the observed signal, and the equality is one of second-order statistics.

The ACF of the representation follows by the argument used for Eq. (21.84): at lag 0 the squares $\eta^2[n]$ and $c_1^2\eta^2[n-1]$ survive, and at lag 1 only the sample $\eta[n-1]$ shared by the two factors, entering them as $c_1\eta[n-1]$ and as $\eta[n-1]$,

$$R_{\mathbf{uu}}[0] = \sigma_\eta^2\left(1 + c_1^2\right), \qquad R_{\mathbf{uu}}[1] = \sigma_\eta^2 c_1. \tag{21.86}$$

These are two equations in the two unknowns, and their ratio cancels σ_η^2 and leaves a single equation in c_1,

$$\frac{c_1}{1 + c_1^2} = \rho_{\mathbf{uu}}[1] \triangleq \frac{R_{\mathbf{uu}}[1]}{R_{\mathbf{uu}}[0]}, \tag{21.87}$$

the lag-1 correlation coefficient of Eq. (21.24). Of the two roots of this quadratic, the invertible one, $|c_1| < 1$, gives

$$c_1 = \frac{1 - \sqrt{1 - 4\rho_{\mathbf{uu}}^2[1]}}{2\rho_{\mathbf{uu}}[1]}, \qquad \sigma_\eta^2 = \frac{R_{\mathbf{uu}}[1]}{c_1}. \tag{21.88}$$

The root is real for any sensor. As c_1 runs over $[-1, 1]$, the ratio $c_1/(1 + c_1^2)$ runs over $[-1/2, 1/2]$, and Eq. (21.84) keeps the correlation inside that range,

$$\left|\rho_{\mathbf{uu}}[1]\right| = \frac{|h_1|\sigma_v^2}{\sigma_\epsilon^2 + \sigma_v^2\left(1 + h_1^2\right)} \leq \frac{|h_1|}{1 + h_1^2} \leq \frac{1}{2}. \tag{21.89}$$

Substituting back into Eq. (21.83),

$$z[n] = h_1 z[n-1] + \eta[n] + c_1\eta[n-1], \tag{21.90}$$

which is an ARMA(1,1) model.

Three readings of Eq. (21.90) are worth noting.

- The AR coefficient survives untouched. The sensor does not change the dynamics of the state, it only contributes the MA term, whose coefficient c_1 carries the sign opposite to h_1.
- The limits behave as they should. For $\sigma_v^2 \to 0$ the correlation $\rho_{\mathbf{uu}}[1] \to 0$ and $c_1 \to 0$, recovering the AR(1) of Eq. (21.81). For $\sigma_v^2 \to \infty$ the correlation tends to $-h_1/(1 + h_1^2)$ and $c_1 \to -h_1$, so the AR and MA parts cancel and $z[n]$ is white, as expected when the record is all sensor noise.
- The order of the MA part counts the noisy measurements, not the states. One noisy sensor adds one MA coefficient.

One realization of this setup, with $h_1 = 0.9$, $\sigma_\epsilon^2 = 1$ and $\sigma_v^2 = 2$, is shown in Fig. 21.11. The sensor noise adds a fast jitter on top of the slow swings of the hidden state, and only the jittered record is available to the model. Its AR(1) difference $u[n] = z[n] - h_1 z[n-1]$ measured on the same record follows Eq. (21.84): the estimated correlation coefficient is $\hat{\rho}_{\mathbf{uu}}[1] = -0.398$ against the predicted -0.390, and beyond lag 1 it scatters about zero, never exceeding 0.042 in absolute value. This is the MA(1) fingerprint of Sec. 21.6.2, and it is what makes the MA term of Eq. (21.90) necessary.

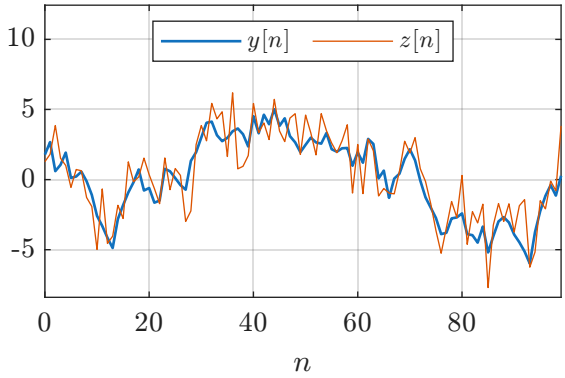


Figure 21.11.: The first 100 samples of the noisy-sensor signal of Example 21.7, with $h_1 = 0.9$, $\sigma_\epsilon^2 = 1$ and $\sigma_v^2 = 2$: the hidden state $y[n]$ and the observation $z[n] = y[n] + v[n]$. The sensor noise leaves the slow dynamics of the state in place and adds a sample-to-sample jitter to it, which is the MA part of the equivalent ARMA(1,1) model.

Measurement noise attenuates AR coefficients

Fitting an AR(1) model to $z[n]$ instead of the ARMA(1,1) of Eq. (21.90) returns, by

Eq. (21.11),

$$\hat{h}_1 = \frac{R_{\mathbf{zz}}[1]}{R_{\mathbf{zz}}[0]} = \frac{\sigma_y^2 h_1}{\sigma_y^2 + \sigma_v^2} < h_1, \qquad \sigma_y^2 = \frac{\sigma_\epsilon^2}{1 - h_1^2}, \tag{21.91}$$

because the measurement noise inflates the lag-0 term without touching the lag-1 one. For $h_1 = 0.9$, $\sigma_\epsilon^2 = 1$ and $\sigma_v^2 = 2$ this gives $\hat{h}_1 = 0.65$ against the true 0.9: the process looks considerably less predictable than it is. The attenuation grows with the sensor noise and is not cured by a longer record. Adding the MA term is what restores the correct dynamics.

As with AR models (Eq. (21.55)), a constant bias μ can be included,

$$\hat{y}[n] = \sum_{i=1}^{p} h_i y[n-i] + \sum_{k=0}^{q} c_k \epsilon[n-k] + \mu \tag{21.92}$$

One of the adopted notation of the ARMA model is

$$\sum_{m=0}^{p} a_m y[n-m] = \sum_{k=0}^{q} c_k \epsilon[n-k] \tag{21.93}$$

with $a_0 = 1$ and $a_m = -h_m$.

21.7.1. Order Selection

Goal: Choose the orders p and q by reading a grid of fitted losses.

Both p and q are hyper-parameters. They are fixed by fitting a grid of candidate pairs and comparing the resulting losses.

Example 21.8: A hidden AR(1) state with $h_1 = 0.9$ and driving noise of variance $\sigma_\epsilon^2 = 1$ is read by a sensor adding white noise of variance $\sigma_v^2 = 2$, and $L = 2000$ samples of the observation $z[n]$ are kept after the transient. This is the setup of Example 21.7, so the true order is known: $z[n]$ is ARMA(1,1) with $c_1 = -0.479$ and an innovation variance $\sigma_\eta^2 = 3.76$, by Eq. (21.88).

Every pair (p, q) with $p = 0, \ldots, 5$ and $q = 0, \ldots, 3$ is fitted. All fits share same sample set, so that the losses are comparable, and the grid is averaged over 200 realizations. The results are given in Table 21.2 and Fig. 21.12, and read as follows.

- *An MA coefficient buys more than an AR one.* At $p = 1$ the loss drops from 4.151 to 3.742 with the first MA coefficient, while the pure AR route needs $p = 3$ to come within 1% of that value. This is the finite-sample face of the MA(1) to AR(∞) expansion: an MA term can always be traded for AR terms, but only at the price of extra order, and here two coefficients do the work of three.
- *The loss floor is the innovation variance of the observed signal.* It is $\sigma_\eta^2 = 3.76$, and not the $\sigma_\epsilon^2 = 1$ of the hidden state, because the measurement noise is not predictable from the past and no model order can remove it. The plateau of the grid sits on it, the $(1, 1)$ entry being 3.742. The opposite corner reads the same way: a model of no order predicts nothing, so the $(0, 0)$ entry 7.279 is the power of the observation itself, $\sigma_z^2 = \sigma_\epsilon^2/(1 - h_1^2) + \sigma_v^2 = 7.263$.

- *The plateau, not the minimum, marks the order.* The loss never increases with p or q, so the smallest entry is always the bottom-right corner, $(5, 3)$ here. What the grid does show is where the surface goes flat, and the order to report is the cheapest one reaching that plateau, $(p, q) = (1, 1)$, the true order of $z[n]$.

Table 21.2.: In-sample loss $\mathcal{L}_{min}/L$ of the LS fit of Example 21.8, averaged over 200 realizations, for the noisy-sensor signal whose true order is ARMA(1,1). The cheapest order within 1% of the smallest entry is underlined.

AR order, p	$q = 0$	$q = 1$	$q = 2$	$q = 3$
0	7.279	6.619	6.082	5.637
1	4.151	<u>3.742</u>	3.740	3.738
2	3.820	3.739	3.732	3.730
3	3.749	3.735	3.730	3.727
4	3.734	3.730	3.728	3.725
5	3.728	3.726	3.725	3.723

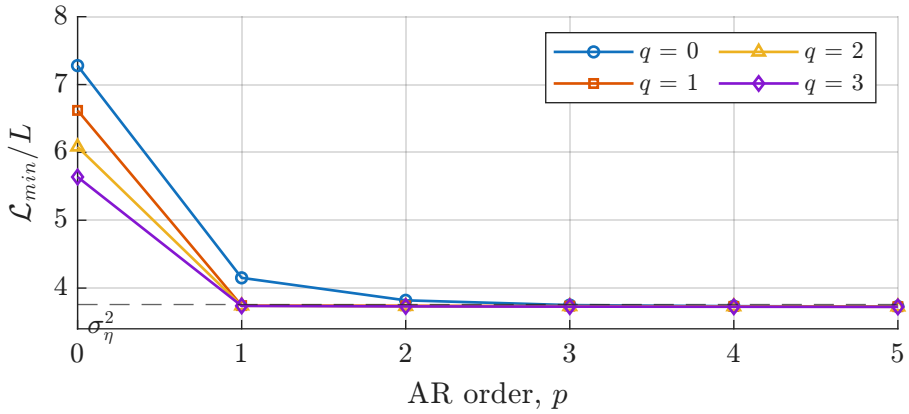


Figure 21.12.: Loss as a function of the AR order p, one curve per MA order q, for the noisy-sensor signal of Example 21.8 with $h_1 = 0.9$, $\sigma_\epsilon^2 = 1$, $\sigma_v^2 = 2$ and $L = 2000$, averaged over 200 realizations. Any $q \geq 1$ reaches the floor already at $p = 1$, whereas the pure AR curve, $q = 0$, approaches it only gradually. The dashed line is the innovation variance σ_η^2 of the equivalent ARMA(1,1) model, Eq. (21.88).

Tips:

- The ACF and PACF fingerprints used for the pure models (Sec. 21.5.1 and Sec. 21.6.2) do not identify an ARMA model. Both of its tails decay without cutting off, so neither plot marks an order, and the two-dimensional search of Table 21.2 is used instead.
- Fitting the largest (p, q) affordable is never penalized by the in-sample loss. Comparing models of different order calls for a criterion that charges for parameters, or for a held-out record.

21.7.2. Prediction Horizon

Goal: Predict more than one step ahead, and see what separates the three model classes once the horizon grows.

Every fit so far was compared one step ahead. A prediction ℓ steps ahead uses $y[n], y[n-1], \ldots$ and nothing later, so the model is iterated forward one sample at a time,

$$\hat{y}[m] = \sum_{i=1}^{p} h_i \hat{y}[m-i] + \sum_{k=1}^{q} c_k \hat{\epsilon}[m-k], \qquad m = n+1, \ldots, n+\ell, \tag{21.94}$$

where the terms of the two sums are

$$\hat{y}[j] = y[j], \quad j \le n \qquad\qquad \hat{\epsilon}[j] = \begin{cases} \epsilon[j], & j \le n \\ 0, & j > n \end{cases} \tag{21.95}$$

A sample past the origin n has not been observed, so the model uses its own forecast of it, supplied by the recursion itself. An innovation past the origin is white and independent of the record, so its best estimate is 0. These two rules are the whole of multi-step prediction, and reading them per model class is what separates the classes.

- *MA(q) forgets after q steps.* Its sum has no $\hat{y}$ terms at all, and once $\ell > q$ every index $m-k$ is past the origin, so every term is zeroed and the forecast is the process mean. Its memory is q samples, no fit can extend it, and beyond that its error is the full power σ_y^2 of the signal.
- *AR(p) decays but never stops.* The recursion feeds on its own output, so a forecast exists at every horizon. It decays with the poles of the model, geometrically as h_1^{ℓ} for AR(1), and the error climbs toward σ_y^2 gradually rather than reaching it at a fixed lag.
- *ARMA(p, q) uses both.* The MA part corrects the first q steps and the AR recursion carries everything after that. Its price is that Eq. (21.94) needs $\epsilon[j]$ for $j \le n$, which the record does not hold: the innovations are reconstructed by the AR(∞) expansion of Sec. 21.6.1, the first stage of the two-stage fit.

Example 21.9: The noisy-sensor record of Example 21.7 is predicted at horizons $\ell = 1, \ldots, 8$ by three models of two coefficients each: AR(2), MA(2) and ARMA(1,1). Eq. (21.94) is applied here to the observation $z[n]$ and its innovation $\eta[n]$. The forecast MSE, averaged over 200 realizations, is shown in Fig. 21.13.

- *The MA model stops predicting at $\ell = 3$.* Its curve is flat from there on at 7.280, the power of the record, against $\sigma_z^2 = 7.263$ for the process. This is not a fitting failure but the structure of Eq. (21.94): past lag $q = 2$ there is nothing left to sum. Raising q moves the wall further out and never removes it.
- *The models with an AR part keep predicting.* AR(2) has a forecast at every horizon, but at the same budget it trails ARMA(1,1) throughout, from 3.820 against 3.743 at $\ell = 1$ to 6.659 against 6.467 at $\ell = 8$, the same order price as at one step, where Table 21.2 needs $p = 3$ to reach the ARMA(1,1) loss. Both approach σ_z^2 only gradually, the ARMA(1,1) curve following the closed form $\sigma_\eta^2 \left(1 + \sum_{j=1}^{\ell-1} \psi_j^2\right)$ with $\psi_j = h_1^{j-1}\,(h_1 + c_1)$, so the horizon over which a model stays below σ_z^2 is what its class buys.

Tips:

- The horizon is chosen by the application, and it decides the model class before any order is fitted: a model with no AR part cannot serve a horizon longer than its own order.
- Two models that tie at $\ell = 1$ can still differ at $\ell = 5$, so a model meant to forecast far ahead is compared at the horizon it will serve.

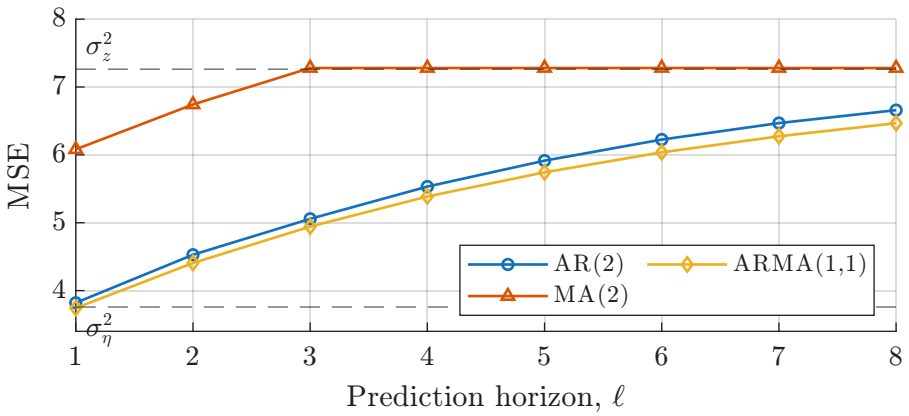


Figure 21.13.: Forecast MSE as a function of the prediction horizon ℓ for the noisy-sensor signal of Example 21.7 with $h_1 = 0.9$, $\sigma_\epsilon^2 = 1$, $\sigma_v^2 = 2$ and $L = 2000$, averaged over 200 realizations. All three models carry two coefficients. The lower dashed line is the innovation variance σ_η^2, the floor at $\ell = 1$, and the upper one is the signal power σ_z^2, reached by any model that has stopped predicting. MA(2) reaches it at $\ell = 3$ and stays there, whereas the models with an AR part approach it only gradually.

21.7.3. Rational PSD (*)

Goal: Read the ARMA coefficients in the frequency domain, and see what the MA part adds to an all-pole model.

Written in the a_m notation of Eq. (21.93), the two coefficient sets define a pair of frequency functions on the DFT grid $\omega_k = 2\pi k/N$,

$$A(\omega_k) = \sum_{m=0}^{p} a_m \exp(-j\omega_k m), \qquad C(\omega_k) = \sum_{m=0}^{q} c_m \exp(-j\omega_k m), \tag{21.96}$$

that is, the DTFT of each coefficient sequence. The driving noise is white, so its PSD is flat (Sec. 21.2 and the left column of Fig. 21.3), and the PSD of the ARMA output is

$$S_{\mathbf{yy}}[k] = \sigma_\epsilon^2 \frac{\left|C(\omega_k)\right|^2}{\left|A(\omega_k)\right|^2}. \tag{21.97}$$

The PSD is thus a *ratio* of two polynomials, which is what the ARMA coefficients buy in the frequency domain.

- The roots of the denominator, contributed by the AR coefficients and termed *poles*, drive $S_{\mathbf{yy}}[k]$ up near their frequency and produce resonance peaks.
- The roots of the numerator, contributed by the MA coefficients and termed *zeros*, drive it down and produce notches.
- A root pair of radius r at normalized frequency f_0 corresponds to the coefficient triple $\left(1, -2r\cos(2\pi f_0), r^2\right)$. Its angle sets the frequency, and its radius, approaching 1, sets how sharp the peak or how deep the notch becomes. A zero of radius exactly 1 gives an exact null, but sits on the boundary of the invertibility condition met above.
- An AR(p) model is all-pole, $C(\omega_k) = 1$. It builds peaks directly and reproduces a notch only by spending several poles on it, which is the frequency-domain form of the order trade-off of Table 21.2. Symmetrically, an MA(q) model is all-zero and cannot produce a sharp resonance.

Example 21.10: An ARMA(2,2) model is built from one pole pair of radius 0.95 at $0.15F_s$ and one zero pair of the same radius at $0.35F_s$, that is

$$\mathbf{h} = (1.117,\ -0.902), \qquad (c_1, c_2) = (1.117,\ 0.902). \tag{21.98}$$

The two root pairs are shown in the left panel of Fig. 21.14: both sit at radius 0.95, the poles at the angle $2\pi \cdot 0.15$ and the zeros at $2\pi \cdot 0.35$. A segment of one realization is drawn in the right panel, where the pole pair is visible as a narrowband oscillation of about $1/0.15 \approx 7$ samples per period.

Its PSD, Eq. (21.97), is drawn in the top panel of Fig. 21.15 together with a Welch estimate (Sec. 20.7.4) of one realization, which follows it closely. The peak sits at $0.15F_s$ and the notch at $0.35F_s$, exactly at the root frequencies, and they are separated by 58 dB.

The bottom panel separates the two factors of Eq. (21.97). The peak comes entirely from $1/\left|A(\omega_k)\right|^2$ and the notch entirely from $\left|C(\omega_k)\right|^2$, each factor being flat where the other acts. Removing the MA part would leave the peak untouched and the notch gone.

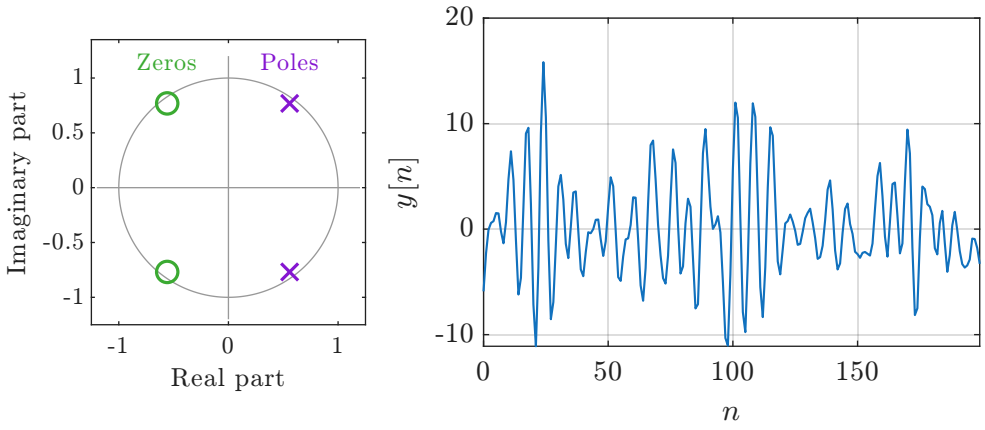


Figure 21.14.: The ARMA(2,2) model of the example. Left: pole (×) and zero (○) pairs in the z-plane, both of radius 0.95, at $0.15F_s$ and $0.35F_s$ respectively. Right: 200 samples of one realization.

This combination is why ARMA is the natural model for signals whose spectra carry both resonances and anti-resonances, whereas an AR model of comparable order captures only the resonances.

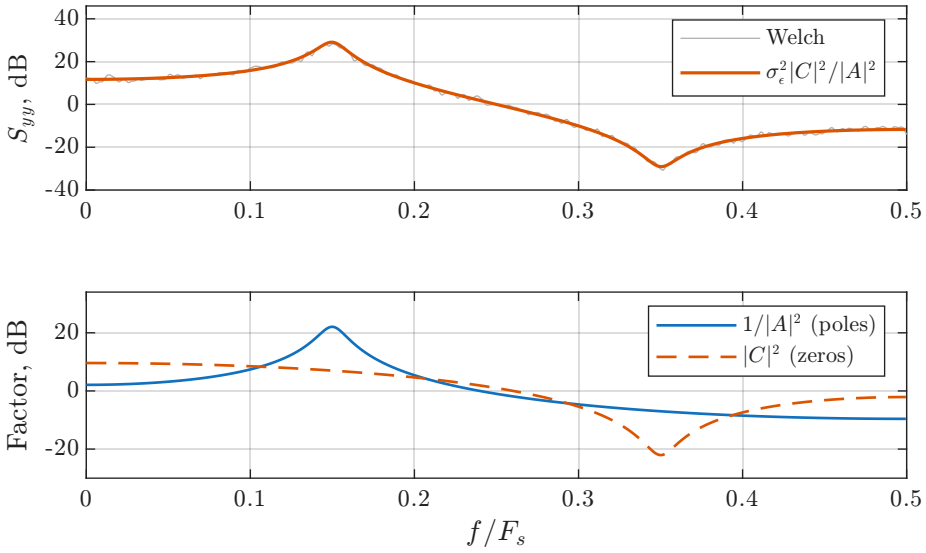


Figure 21.15.: Rational PSD of an ARMA(2,2) model with a pole pair at $0.15F_s$ and a zero pair at $0.35F_s$, both of radius 0.95. Top: the closed form of Eq. (21.97) and a Welch estimate of one realization of $L = 8000$ samples. Bottom: the two factors separately, showing that the poles account for the peak and the zeros for the notch.

22. ARX

Goal: Extension for AR model to ARX model.

The ARX (**A**uto-**R**egressive with e**X**tra input or **A**uto-**R**egressive e**X**ogenic) model extends the AR approach by incorporating an additional input signal $x[n]$ that influences the output $y[n]$. The term "exogenous" indicates that the additional input signal comes from outside the system, unlike an AR model that relies solely on past values of the output.

Systems classification Two class of models:

- **Endogenic/endogenous** system is a system without inputs.
- **Exogenic/exogenous** is a system with inputs.

The $ARX(p, q)$ model is given by

$$\begin{aligned} y[n] = & h_1 y[n-1] + \cdots + h_p y[n-p] \\ & + b_1 x[n-1] + \cdots + b_q x[n-k] + \epsilon[n], \end{aligned} \tag{22.1}$$

where

- h_i are AR coefficients related to the past of $y[n]$,
- b_i are he coefficients that relate the current output $y[n]$ to past values of the exogenous input $x[n]$
- $\epsilon[n]$ is noise or modeling error.

Contents

22.1. Cross-Correlation Function (CCF) . . . **349**
22.1.1. Time-Difference Estimation . . . 351
22.2. Cross-Spectral Density (CSD) . . . **354**
22.2.1. Definition . . . 354
22.2.2. Cross-Coherence . . . 355
22.3. ARX(0,q) model . . . **356**
22.4. General ARX model . . . **357**
22.5. Time-Domain Filtering . . . **358**
22.6. ARMAX . . . **359**
22.7. ARI, ARIMA, ARIMAX . . . **359**
22.7.1. De-trending/Differencing . . . 359
22.7.2. ARI Family . . . 360
22.8. Non-linear AR and ARX Models . . . **360**
22.9. Vector AR (VAR) . . . **360**

Adding an exogenous input changes what goes into the regressor matrix. Each section of this chapter adds one more ingredient to it: past values of the input, past noise terms, removal of a trend, several signals at once, or a non-linear mapping in place of the weighted sum. Table 22.1 lists the resulting models and the ingredients each one uses. The first two rows are the models of the previous chapter, repeated as the starting point.

Table 22.1.: Models of this chapter and the ingredients of their regressor matrix. The first two rows are carried over from the previous chapter.

Model	past $y[n-i]$	past $x[n-k]$	noise $\epsilon[n-k]$	trend removal	vector	non-linear	Sec.
AR(p)	✓						21.3
ARMA(p, q)	✓		✓				21.7
ARX(p, q)	✓	✓					22.4
ARMAX	✓	✓	✓				22.6
ARI(p, d)	✓			✓			22.7
ARIMA(p, d, q)	✓		✓	✓			22.7
ARIMAX(p, d, q)	✓	✓	✓	✓			22.7
NARX	✓	✓			✓	✓	22.8
VAR(p)	✓				✓		22.9

22.1. Cross-Correlation Function (CCF)

Goal: Whereas the ACF measures how a single signal correlates with its own time-shifted versions, the cross-correlation function measures the relationship between two different signals

The goal is to predict $y[n]$ from $x[n-k]$ with a single exogenous term at lag k by b_k coefficient,

$$\hat{y}[n] = b_k x[n-k], \tag{22.2}$$

The resulting MSE-based loss function is of the form

$$\mathcal{L}(b) = \frac{1}{2}\sum_n \big(y[n] - b_k x[n-k]\big)\big)^2 \tag{22.3}$$

with the solution by

$$\frac{d\mathcal{L}(b)}{db} = \sum_n (y[n] - b_k x[n-k])(-x[n-k]) = 0 \tag{22.4}$$

The corresponding solution is

$$b_k = \frac{\sum_n y[n]x[n-k]}{\sum_n x^2[n-k]}. \tag{22.5}$$

Cross-Correlation Function The resulting coefficients are related to the cross-correlation function,

$$R_{\mathbf{xy}}[k] = \sum_n y[n]x[n-k], k = -L+1, \ldots, L-1 \tag{22.6}$$

so that the numerator of Eq. (22.5) is $R_{\mathbf{xy}}[k]$ itself, and a peak at a positive lag k marks an input leading the output by k samples, which is the lag worth adding to the model.

Similar to the ACF, cross-correlation can also be defined in biased, unbiased, or normalized forms:

$$R_{\mathbf{xy},biased}[k] = \frac{1}{L} R_{\mathbf{xy}}[k] \tag{22.7}$$

$$R_{\mathbf{xy},unbiased}[k] = \frac{1}{L-|k|} R_{\mathbf{xy}}[k] \tag{22.8}$$

$$R_{\mathbf{xy},norm}[k] = \frac{R_{\mathbf{xy}}[k]}{\sqrt{R_{\mathbf{x}}[0]R_{\mathbf{y}}[0]}} \tag{22.9}$$

Note, these modification are available only if $x[n]$ and $y[n]$ are of the same length. Otherwise, only $R_{\mathbf{xy}}[k]$ (22.6) is used.

The normalized cross-correlation function has correlation coefficient interpretation,

$$R_{\mathbf{xy},norm}[k] \approx \rho_{\mathbf{xy}}[k] \tag{22.10}$$

Properties:

$$R_{\mathbf{xy}}[k] = R_{\mathbf{yx}}[-k] \tag{22.11}$$

$$R_{\mathbf{xy}}[-k] = R_{\mathbf{yx}}[k] \tag{22.12}$$

$$\left|R_{\mathbf{xy}}[k]\right| \leqslant \sqrt{R_{\mathbf{x}}[0]R_{\mathbf{y}}[0]} \tag{22.13}$$

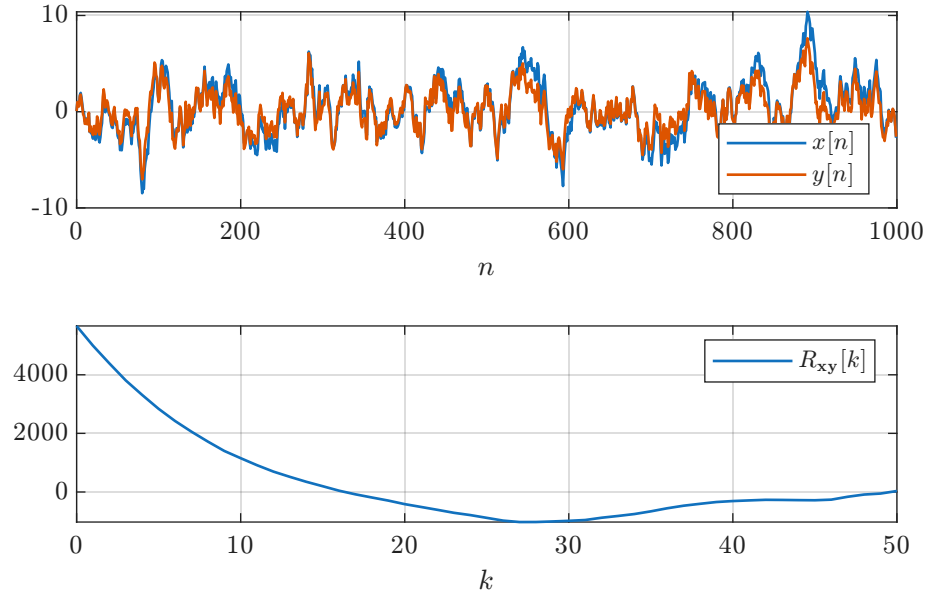
$$\left|R_{\mathbf{xy}}[k]\right| \leqslant \frac{1}{2}\left[R_{\mathbf{x}}[0] + R_{\mathbf{y}}[0]\right] \tag{22.14}$$

Interpretation If there is a strong correlation at some lag k, it suggests that $y[n]$ is influenced by $x[n-k]$. In an ARX setting, identifying the lag k at which the cross-correlation peaks can guide the selection of q and help determine which past inputs are most relevant for predicting $y[n]$.

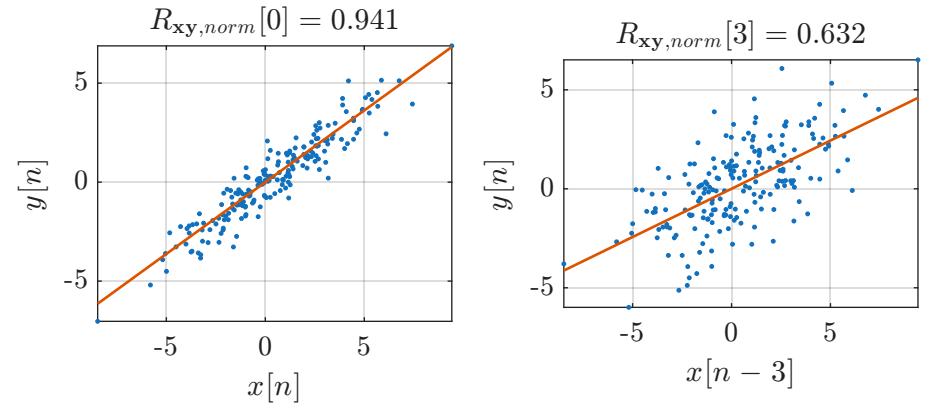
Example 22.1: The solution in Eq. (20.23) is

$$\hat{A} = \frac{R_{\mathbf{xy}}[0]}{R_{\mathbf{x}}[0]} \tag{22.15}$$

Cross-Covariance Function For simplicity, a zero-average, $\bar{x}[n] = \bar{y}[n] = 0$, was assumed. When either of the signals is non-zero mean, the subtraction of signal average from the signal before cross-correlation calculation is termed as cross-covariance. It is similar to auto-correlation and auto-covariance functions in Sec. 21.1.4.



(a) Cross-correlation



(b) $y[n]$ vs. $x[n-k]$

Figure 22.1.: Illustration of the linear dependence between $y[n]$ and $x[n-k]$. In (b) the straight line is the least-squares fit of Eq. (22.5), so its slope is b_k, while the scatter of the points about it is what the normalized cross-correlation printed above each panel measures.

22.1.1. Time-Difference Estimation

Goal: Measure the delay between two signals from the location of the cross-correlation peak.

A common special case is that the output is nothing but a scaled, delayed and noisy copy of the input,

$$y[n] = A\,x[n-n_0] + \epsilon[n], \tag{22.16}$$

where the delay n_0 is the unknown of interest.

This is the common model of a signal recorded by two sensors at different distances from the same source, of a transmitted pulse and its echo, or of a reference channel and a propagated one.

The lag search is the least-squares fit The section already solved the fit at a *given* lag: b_k (22.5) returns the best gain b_k for the single-term predictor $\hat{y}[n] = b_k x[n-k]$. For $|k| \ll L$ its

denominator is the input energy $R_{\mathbf{xx}}[0]$, so

$$b_k \approx \frac{R_{\mathbf{xy}}[k]}{R_{\mathbf{xx}}[0]}. \tag{22.17}$$

What remains is to choose k. Substituting b_k back into the loss gives the residual it leaves (see also auto-correlation result in Eq. (21.33)),

$$\mathcal{L}_{min}(k) = \frac{1}{2}\left(R_{\mathbf{yy}}[0] - \frac{R_{\mathbf{xy}}^2[k]}{R_{\mathbf{xx}}[0]}\right), \tag{22.18}$$

Neither $R_{\mathbf{yy}}[0]$ nor $R_{\mathbf{xx}}[0]$ depends on k, so minimizing the residual over the lag is the same as maximizing $R_{\mathbf{xy}}^2[k]$,

$$\hat{n}_0 = \arg\max_k \left|R_{\mathbf{xy}}[k]\right|, \qquad \hat{A} = \frac{R_{\mathbf{xy}}[\hat{n}_0]}{R_{\mathbf{xx}}[0]}, \qquad \hat{\tau} = \frac{\hat{n}_0}{f_s}. \tag{22.19}$$

The absolute value admits a polarity inversion, $A < 0$.

Example 22.2: A broadband $x[n]$ of $L = 1000$ samples, sampled at $f_s = 100$ Hz, is delayed by $n_0 = 20$ samples and scaled by $A = 0.8$ before independent noise is added, as in Eq. (22.16). Searching $|k| \le 50$ in the left column of Fig. 22.2 gives $\hat{n}_0 = 20$ samples, a peak of $R_{\mathbf{xy},norm}[\hat{n}_0] = 0.876$, and

$$\hat{A} = \frac{R_{\mathbf{xy}}[20]}{R_{\mathbf{xx}}[0]} = 0.814, \qquad \hat{\tau} = \frac{20}{100} = 0.2 \text{ s}, \tag{22.20}$$

recovering both the delay and the gain.

Example 22.3: Repeat the estimate for a narrowband input,

$$x[n] = \cos\left(2\pi f_0 n/f_s\right) + 0.5\epsilon_x[n], \qquad f_0 = 5 \text{ Hz},$$

sampled at $f_s = 100$ Hz over $L = 1000$ samples and delayed by $n_0 = 6$ samples as in Eq. (22.16). The right column of Fig. 22.2 shows the pair, the spectrum of the input, and the CCF.

Solution: The middle panel already decides the outcome. The input power sits in a single line at f_0, standing 22.4 dB above the median level of the spectrum, against 10.6 dB for the broadband input of the previous example. Such a spectrum has an auto-correlation that oscillates at f_0 without decaying, and the CCF of Eq. (22.16) is that auto-correlation shifted to $k = n_0$.

The bottom panel is therefore not a peak but a train of lobes, alternating sign every $f_s/2f_0 = 10$ lags and repeating every $f_s/f_0 = 20$ lags. Their heights are

$$\begin{aligned} R_{\mathbf{xy},norm}[6] &= 0.669, & R_{\mathbf{xy},norm}[-4] &= -0.678, \\ R_{\mathbf{xy},norm}[-14] &= 0.663, & R_{\mathbf{xy},norm}[26] &= 0.638, \end{aligned} \tag{22.21}$$

four candidates within 0.04 of one another. The estimator of Eq. (22.19) takes the largest in magnitude and returns

$$\hat{n}_0 = -4, \tag{22.22}$$

missing the true delay by 10 samples, exactly half a carrier period. Nothing but noise separates the lobes, so another realization of the same experiment hands the win to a different one.

Knowing the polarity of the coupling helps only halfway. With $A > 0$ assumed, the search is restricted to the positive lobes and returns $\hat{n}_0 = 6$ here, but those lobes are themselves 20 lags apart and equally tied. The ambiguity is reduced from half a carrier period to a full one, not removed.

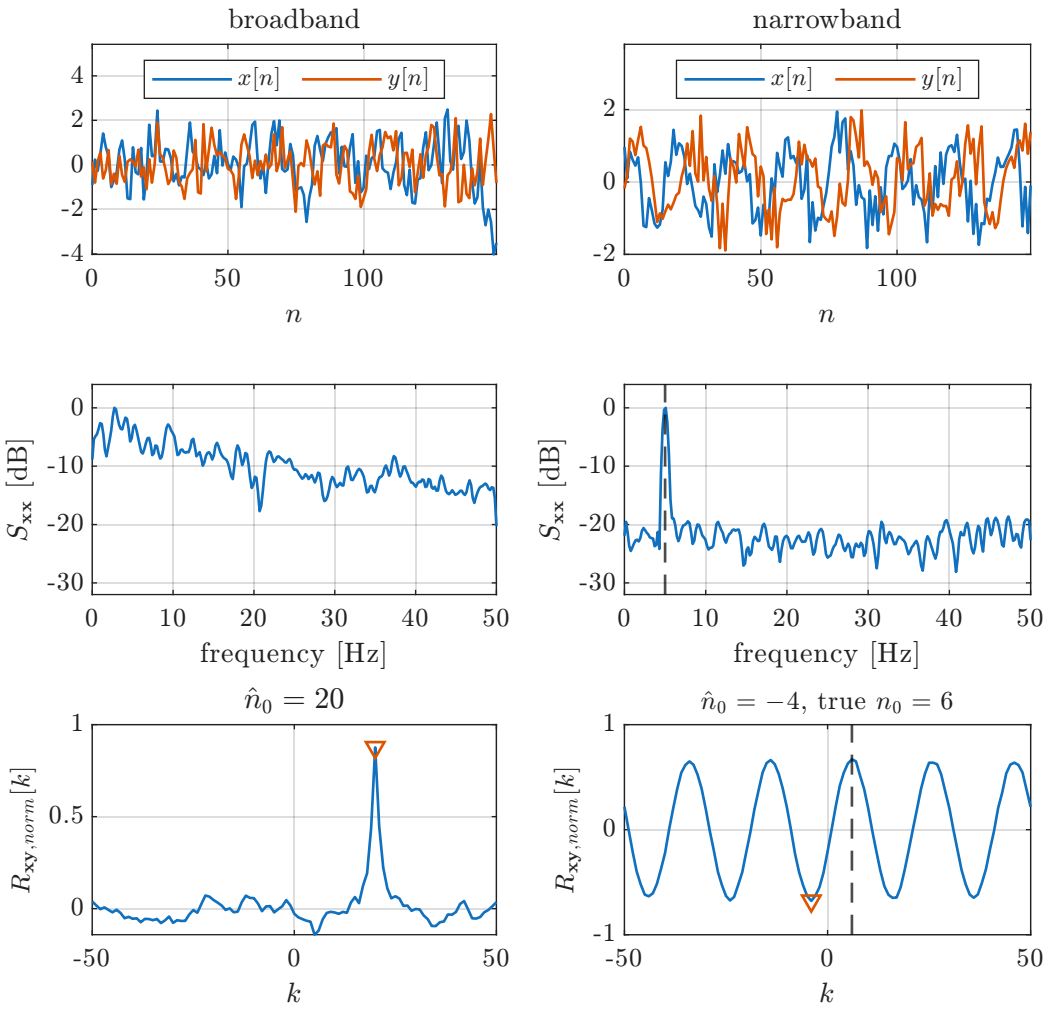


Figure 22.2.: Delay estimation from the CCF peak. Top row: an input and its delayed, scaled and noisy copy. Middle row: the Welch estimate of the input spectrum $S_{\mathbf{xx}}$, normalized to a 0 dB peak. Bottom row: the corresponding normalized cross-correlation. In the broadband case (left) the power is spread over the whole band and the CCF collapses to a single sharp peak, marked at $\hat{n}_0 = 20$, the true delay. In the narrowband case (right) the power sits in a line at the carrier $f_0 = 5$ Hz (dashed) and the CCF inherits the carrier period: lobes of alternating sign recur every $f_s/2f_0 = 10$ lags, none of them decaying, and in this realization the search settles on $\hat{n}_0 = -4$ rather than on the true $n_0 = 6$ (dashed), an error of half a period.

Which normalization The biased and normalized forms of Eq. (22.6) rescale $R_{\mathbf{xy}}[k]$ by a constant, so they leave the location of the peak untouched and only change the number printed on the axis; the normalized form is the convenient one, since its peak height is directly the correlation coefficient of the fit. The unbiased form is the one to avoid: dividing by $L - |k|$ inflates the tail of the lag axis, where few terms are averaged, and manufactures spurious peaks far from the true delay. This is the same argument that favors the biased ACF in the peak search of Sec. 21.2.2.

Tip:

- Sub-sample refinement: a true delay is rarely an integer number of samples, whereas Eq. (22.19) searches a grid of spacing $1/f_s$. Fitting a parabola through $R_{\mathbf{xy}}[\hat{n}_0-1]$, $R_{\mathbf{xy}}[\hat{n}_0]$ and $R_{\mathbf{xy}}[\hat{n}_0+1]$

and taking its vertex gives the fractional correction

$$\delta = \frac{1}{2} \cdot \frac{R_{\mathbf{xy}}[\hat{n}_0 - 1] - R_{\mathbf{xy}}[\hat{n}_0 + 1]}{R_{\mathbf{xy}}[\hat{n}_0 - 1] - 2R_{\mathbf{xy}}[\hat{n}_0] + R_{\mathbf{xy}}[\hat{n}_0 + 1]}, \tag{22.23}$$

so that $\hat{\tau} = (\hat{n}_0 + \delta)/f_s$. This is the lag-axis counterpart of the parabolic peak refinement of the periodogram in Eq. (20.48).

The sign of the lag is a convention

Which of the two signals is shifted in Eq. (22.6) is a choice, not a property of the data, and books and software packages do not agree on it. Reversing the choice mirrors the lag axis, $R_{\mathbf{xy}}[k] = R_{\mathbf{yx}}[-k]$, so the same pair of signals can show its peak at $k = +6$ in one source and at $k = -6$ in another.

The convention adopted here shifts the input, so that a peak at a positive lag means the input leads the output. For example, in MATLAB it is `xcorr(y,x)`, the output passed first; `xcorr(x,y)` returns the mirrored axis.

Before reading a lag off any implementation, check the documentation or just cross-correlate a signal with a delayed copy of itself, and see which side the peak lands on.

22.2. Cross-Spectral Density (CSD)

22.2.1. Definition

The Cross-Spectral Density (CSD) is the frequency-domain counterpart to the cross-correlation function (CCF). Following the Wiener-Khinchin theorem discussed in Sec. 21.2, the CSD is defined as the Fourier transform of the cross-correlation sequence $R_{\mathbf{xy}}[n]$:

$$S_{\mathbf{xy}}[k] = \mathrm{DFT}\left\{R_{\mathbf{xy}}[n]\right\} \tag{22.24}$$

For finite-time signals, using the Discrete Fourier Transform (DFT), it is practically estimated using the cross-periodogram:

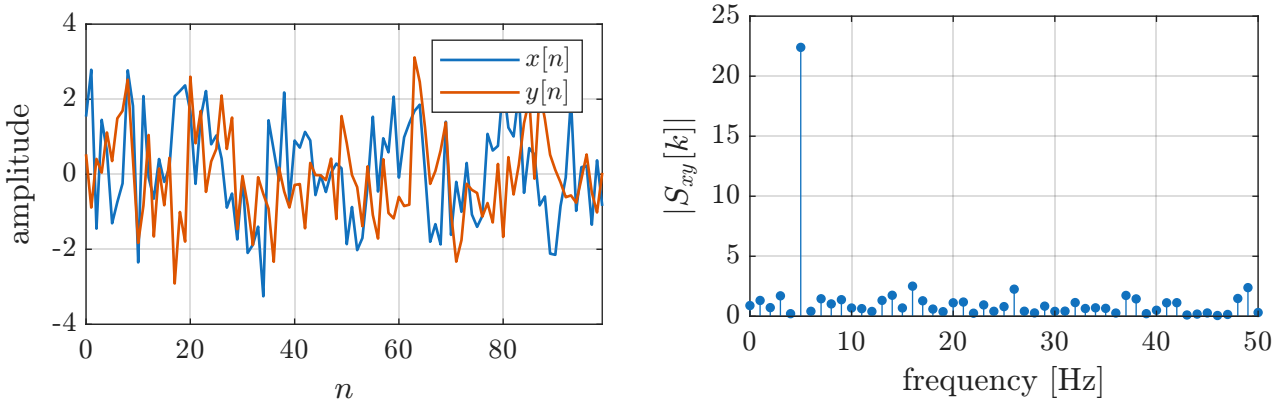
$$S_{\mathbf{xy}}[k] = X^*[k]Y[k] \tag{22.25}$$

where $X[k]$ and $Y[k]$ are the DFTs of the signals $x[n]$ and $y[n]$, and $X^*[k]$ denotes the complex conjugate of $X[k]$.

Example 22.4: Consider two short noisy sinusoids of frequency $f_0 = 5$ Hz sampled at $f_s = 100$ Hz, where $y[n]$ is a delayed copy of $x[n]$ with delay $n_0 = 6$ samples,

$$\begin{aligned} x[n] &= \cos\big(2\pi f_0 n/f_s\big) + 0.5\,\epsilon_x[n] \\ y[n] &= \cos\big(2\pi f_0 (n - n_0)/f_s\big) + 0.5\,\epsilon_y[n] \end{aligned}$$

with $L = 100$ samples. The signals are shown in Fig. 22.3(a). Because the records are short, the cross-periodogram is a noisy estimate of the true CSD; nevertheless, $\left|S_{\mathbf{xy}}[k]\right|$ exhibits a clear peak at the sinusoid frequency, as seen in Fig. 22.3(b).



(a) $x[n]$ and $y[n]$ (b) $\left|S_{\mathbf{xy}}[k]\right|$

Figure 22.3.: Cross-periodogram example: noisy sinusoids related by a delay of $n_0 = 6$ samples and the magnitude of their cross-spectral density estimate.

Properties

Unlike the PSD ($S_{\mathbf{xx}}[k]$), which is strictly real and non-negative, the CSD is generally a complex-valued function.

- *Magnitude:* $\left|S_{\mathbf{xy}}[k]\right|$ indicates the strength of the shared power between the two signals at frequency bin k.
- *Phase:* $\angle S_{\mathbf{xy}}[k]$ represents the phase difference between $y[n]$ and $x[n]$ at that specific frequency. For the pure delay of Eq. (22.16) this phase is linear in k, $\angle S_{\mathbf{xy}}[k] = -2\pi k n_0/L$, so the delay estimated in Sec. 22.1.1 from the CCF peak can equivalently be read off the slope of the phase.
- *Conjugate Symmetry:* $S_{\mathbf{xy}}[k] = S^*_{\mathbf{yx}}[k]$.

While the CCF shows the linear relationship across time lags, the CSD reveals how that relationship is distributed across different frequencies.

22.2.2. Cross-Coherence

The **cross-coherence** (or magnitude-squared coherence) is the frequency-domain counterpart of the normalized CCF: it expresses, at each frequency bin k, how strongly the two signals share a stable linear relation.

Built directly from the CSD $S_{\mathbf{xy}}[k]$ and the two PSDs $S_{\mathbf{xx}}[k]$, $S_{\mathbf{yy}}[k]$, it is defined by

$$\gamma^2_{\mathbf{xy}}[k] = \frac{\left|S_{\mathbf{xy}}[k]\right|^2}{S_{\mathbf{xx}}[k]\, S_{\mathbf{yy}}[k]} \tag{22.26}$$

which can be read as the squared correlation coefficient at frequency k, mirroring the time-domain identity $R^2_{\mathbf{xy},norm}[k] \approx \rho^2_{\mathbf{xy}}[k]$ [1].

The cross-coherence is bounded between 0 and 1 for all frequency bins:

$$0 \leqslant \gamma^2_{\mathbf{xy}}[k] \leqslant 1 \tag{22.27}$$

[1] The discussion on the complex-valued $\gamma_{\mathbf{xy}}[k] = \frac{S_{\mathbf{xy}}[k]}{\sqrt{S_{\mathbf{xx}}[k]\, S_{\mathbf{yy}}[k]}}$ is beyond the scope of this chapter.

Interpretation

- A value near 1 means that, across realizations, the two signals oscillate at frequency k with a consistent gain and phase offset (a stable linear coupling).
- A value near 0 means no consistent coupling at that frequency: gain or phase varies from segment to segment.

A typical estimate is shown in Fig. 22.4, where $y[n]$ is a band-limited filtered version of $x[n]$ corrupted by independent noise.

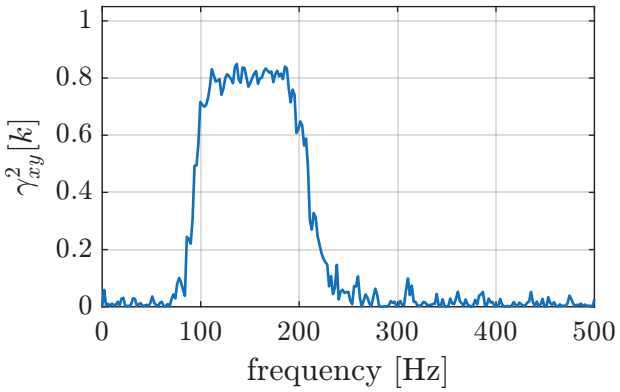


Figure 22.4.: Estimated cross-coherence $\gamma_{\mathbf{xy}}[k]$ between an input $x[n]$ and a band-pass filtered noisy output $y[n]$, computed via Welch's method. Coherence is close to 1 inside the pass band and drops to near 0 outside.

The requirement for averaging

For a single periodogram, $S_{\mathbf{xy}}[k] = X^*[k]Y[k]$, $S_{\mathbf{xx}}[k] = \big|X[k]\big|^2$, $S_{\mathbf{yy}}[k] = \big|Y[k]\big|^2$, so

$$\gamma^2_{\mathbf{xy}}[k] = \frac{\big|X^*[k]Y[k]\big|^2}{\big|X[k]\big|^2\big|Y[k]\big|^2} = 1 \tag{22.28}$$

identically, regardless of the underlying signals. Coherence becomes meaningful only after averaging $M > 1$ segments (Welch / Bartlett, Sec. 20.7.4).

22.3. ARX(0,q) model

The ARX(0,q) model describes a scenario where the output $y[n]$ depends purely on the past values of an external (exogenous) input $x[n]$, without feedback from its own past outputs [12, Example 4.3, pp. 90]

$$\begin{aligned} y[n] &= b_1 x[n-1] + \cdots + b_q x[n-q] + \epsilon[n] \\ &= \sum_{k=1}^{q} b_k x[n-k] + \epsilon[n] \end{aligned} \tag{22.29}$$

In matrix form, the past values are arranged into a matrix $\mathbf{X}$,

$$\underbrace{\begin{bmatrix} \hat{y}[1] \\ \hat{y}[2] \\ \vdots \\ \hat{y}[L-1] \end{bmatrix}}_{\hat{\mathbf{y}}} = \underbrace{\begin{bmatrix} \begin{bmatrix} x[0] & 0 & \cdots & 0 \\ x[1] & x[0] & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ x[L-2] & x[L-3] & \vdots & x[L-m-2] \end{bmatrix} \end{bmatrix}}_{\mathbf{X}} \underbrace{\begin{bmatrix} b_1 \\ \vdots \\ b_q \end{bmatrix}}_{\mathbf{b}} \tag{22.30}$$

with $\hat{\mathbf{y}} \in \mathscr{R}^{L-1}, \mathbf{X} \in \mathscr{R}^{(L-1)\times q}, \mathbf{b} \in \mathscr{R}^{q}$. The resulting $\mathbf{b}$ coefficients are found by the corresponding LS minimization. Similar ot AR model, the solution is also comprised of the corresponding auto-correlations $R_{\mathbf{xx}}[k]$ resulted for $\mathbf{X}^T\mathbf{X}$ and cross-correlations $R_{\mathbf{xy}}[k]$ resulted from $\mathbf{X}^T\mathbf{y}$. This reveals that the solution leverages both the structure of the input's autocorrelation and the input-output cross-correlation.

Interpretation The model is essentially a linear filter of $x[n]$.

22.4. General ARX model

In a general ARX(p,q) model, the output is represented as a linear combination of both its own past values and the past values of an exogenous input. LS formulation involves matrix $\mathbf{X}$ that is constructed from past values of $x[n]$ and $y[n]$, shifted according to the lags involved. The vector $\mathbf{w}$ includes both h_i and b_k values.

Example 22.5: ARX(3,3) model with signals

$$x[n] = x[0], x[1], \ldots, x[7]$$
$$y[n] = y[0], y[1], \ldots, y[7]$$

The required difference equation is

$$\begin{aligned} \hat{y}[n] = {} & h_1 y[n-1] + h_2 y[n-2] + h_3 y[n-2] \\ & + b_1 x[n-1] + b_2 x[n-2] + b_3 x[n-3] \end{aligned} \tag{22.31}$$

Find prediction of $\hat{y}[8]$.

Solution: The coefficients are given by

$$\arg\min_{\mathbf{w}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\| \tag{22.32}$$

where

$$\mathbf{X} = \begin{bmatrix} x[0] & 0 & 0 & y[0] & 0 & 0 \\ x[1] & x[0] & 0 & y[1] & y[0] & 0 \\ x[2] & x[1] & x[0] & y[2] & y[1] & y[0] \\ x[3] & x[2] & x[1] & y[3] & y[2] & y[1] \\ x[4] & x[3] & x[2] & y[4] & y[3] & y[2] \\ x[5] & x[4] & x[3] & y[5] & y[4] & y[3] \\ x[6] & x[5] & x[4] & y[6] & y[5] & y[4] \end{bmatrix}, \tag{22.33}$$

$$\mathbf{w} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix}, \mathbf{y} = \begin{bmatrix} y[1] \\ y[2] \\ y[3] \\ y[4] \\ y[5] \\ y[6] \\ y[7] \end{bmatrix}$$

The prediction of $\hat{y}[8]$ is straightforward after finding the prediction coefficients by LS minimization. The resulting calculation is comprised of the corresponding $R_{\mathbf{xx}}[k]$ and $R_{\mathbf{xy}}[k]$ values.

22.5. Time-Domain Filtering

Goal: Use AR model for signal enhancement. In this problem there two common versions that include signal and noise combinations that differs by signal availability during the training. We assume both signal and noise are stationary. In he first one, clean and another noisy versions are available,

$$\begin{cases} x[n] \\ y[n] = x[n] + \epsilon[n] \end{cases} \tag{22.34}$$

In another one (Wiener filter), noise and noisy versions are available,

$$\begin{cases} \varepsilon[n] \\ y[n] = x[n] + \epsilon[n] \end{cases} \tag{22.35}$$

In either case, the learned AR(p) model can be used.

Noise filtering (denoising)

We have training data $\left\{x[n], y[n]\right\}_{n=0}^{L-1}$ and we are interested to learn coefficients h_k, such that

$$\hat{x}[n] = \sum_{k=0}^{p} h_k y[n-k] \tag{22.36}$$

The key idea is to construct the AR matrix $\mathbf{X}$ (as in (21.43)) from the shifted versions of $y[n]$ rather than $x[n]$. By fitting an AR model the resulting coefficients effectively learn how to reconstruct the clean signal from the noisy input. The matrices are

$$\begin{aligned} \mathbf{X} &= \begin{bmatrix} y[p] & y[p-1] & \cdots & y[0] \\ y[p+1] & y[p] & \cdots & y[1] \\ \vdots & \cdots & \ddots & \vdots \\ y[L-1] & y[L-2] & \cdots & y[L-1-p] \end{bmatrix}, \\ \mathbf{y} &= \begin{bmatrix} x[p] \\ x[p+1] \\ \vdots \\ x[L-1] \end{bmatrix} \end{aligned} \tag{22.37}$$

and the resulting coefficients can be found by the normal equation.

Using notation in Sec. 21.3.1, the problem can be rewritten as matrix $\mathbf{R}$ of $R_{\mathbf{yy}}[k]$ elements and vector $\mathbf{r}$ with $R_{\mathbf{xy}}[k]$ elements. The resulting h_k values are optimal in the sense of MSE.

This approach sets the foundation for adaptive filtering methods, where the model continuously adjusts its parameters to best estimate the clean signal under changing noise conditions.

22.6. ARMAX

Goal: The ARMAX model extends the concepts of ARMA, and ARX models by combining their elements to capture more complex dynamics.

The model that combines ARMA (signal history and noise) together with exogenous input (ARX model),

$$\begin{aligned} y[n] &= h_1 y[n-1] + \cdots + h_{n_a} y[n-n_a] \\ &+ b_1 x[n-1] + \cdots + b_{n_b} x[n-n_b] \\ &+ c_1 \epsilon[n-1] + \cdots + c_{n_c} \epsilon[n-n_c] + \epsilon[n] \end{aligned} \tag{22.38}$$

22.7. ARI, ARIMA, ARIMAX

Goal: Handle model with trend.

When time series data exhibit trends, the basic AR, MA, and ARMA models may not be directly suitable. Trends refer to a systematic change in the mean level of the series, often increasing or decreasing over time. To accommodate such trends, de-trending can be applied before fitting ARMA-type models.

22.7.1. De-trending/Differencing

The basic model with linear trend is

$$y[n] = A + Bn + \epsilon[n], \tag{22.39}$$

where B is the slope of the trend. Let's define

$$\begin{aligned} y[n] - y[n-1] &= A + Bn + \epsilon[n] - A - B(n-1) - \epsilon[n-1] \\ &= B + \epsilon[n] - \epsilon[n-1] \end{aligned} \tag{22.40}$$

This is known as first-order differencing that effectively removes the constant slope.

The quadratic (or parabolic) trend is given by

$$y[n] = an^2 + bn + c \tag{22.41}$$

Applying the differencing twice,

$$y'[n] = y[n] - y[n-1]$$

$$\begin{aligned} y''[n] &= y'[n] - y'[n-1] \\ &= y[n] - y[n-1] - (y[n-1] - y[n-2]) \\ &= y[n] - 2y[n-1] + y[n-2] \end{aligned}$$

can remove a quadratic trend.

In practice, differencing is often done as a preliminary step. If a single differencing is needed to achieve stationarity, this is referred to as $d = 1$; if twice, $d = 2$, and so forth.

22.7.2. ARI Family

ARI(p,d) If an AR model (p) is applied to data that have been differenced d times to remove trend. The "I" stands for "Integrated", indicating differencing to remove trend.

ARIMA(p,d,q) ARMA model with de-trending is termed ARIMA(p,d,q). ARIMA(p,0,q) is actually ARMA(p,q).

ARIMAX(p,d,q) Similar to ARMAX, ARIMAX includes exogenous inputs (X) along with the ARIMA model.

22.8. Non-linear AR and ARX Models

A *non-linear autoregressive* model replaces the linear combination $h_1 y[n-1] + \cdots + h_p y[n-p]$ with a non-linear mapping implemented by a non-linear model approximation, e.g. by neural network:

$$\hat{y}[n] = f(y[n-1], \ldots, y[n-p]) + \epsilon[n] \tag{22.42}$$

where $f(\cdot; \mathbf{w})$ is the model and $\mathbf{w}$ its weights.

For neural network with purely *linear* activations the NAR(p) collapses to the classical AR(p).

The corresponding NARX model is

$$\hat{y}[n] \;=\; f(\mathbf{x}, \mathbf{y}) + \epsilon[n] \tag{22.43}$$

The network learns an arbitrary non-linear mapping from the selected lags of $y[n]$ and $x[n]$ to the current output.

22.9. Vector AR (VAR)

The goal is to use AR multivariate prediction of an N-dimensional signal $\mathbf{y}[n]$ by its L historic values,

$$\left\{ \begin{bmatrix} y_0[0] \\ y_1[0] \\ \vdots \\ y_{N-1}[0] \end{bmatrix}, \begin{bmatrix} y_0[1] \\ y_1[1] \\ \vdots \\ y_{N-1}[1] \end{bmatrix} \cdots \begin{bmatrix} y_0[L-1] \\ y_1[L-1] \\ \vdots \\ y_{N-1}[L-1] \end{bmatrix} \right\} \tag{22.44}$$

VAR(1) Model

For example, the VAR(1) model of 2-dimensional signal ($N = 2$) is

$$\begin{bmatrix} \hat{y}_0[n] \\ \hat{y}_1[n] \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \begin{bmatrix} y_0[n-1] \\ y_1[n-1] \end{bmatrix} + \begin{bmatrix} \epsilon_0[n] \\ \epsilon_1[n] \end{bmatrix} \tag{22.45}$$

In compressed matrix notation,

$$\hat{\mathbf{y}}[n] = \mathbf{A}\mathbf{y}[n-1] + \boldsymbol{\epsilon}[n] \tag{22.46}$$

where $\mathbf{A} \in \mathbb{R}^{N \times N}, \boldsymbol{\epsilon}[n] \in \mathbb{R}^N$.

History The relation to MA model is similar to univariate case,

$$\begin{aligned} \hat{\mathbf{y}}[n] &= \mathbf{A}\mathbf{y}[n-1] + \boldsymbol{\epsilon}[n] \\ &= \mathbf{A}(\mathbf{A}\mathbf{y}[n-2] + \boldsymbol{\epsilon}[n-1]) + \boldsymbol{\epsilon}[n] \\ &= \mathbf{A}^2\mathbf{y}[n-2] + \mathbf{A}\boldsymbol{\epsilon}[n-1] + \boldsymbol{\epsilon}[n] \\ &= \mathbf{A}^3\mathbf{y}[n-3] + \mathbf{A}^2\boldsymbol{\epsilon}[n-2] + \mathbf{A}\boldsymbol{\epsilon}[n-1] + \boldsymbol{\epsilon}[n] \end{aligned} \tag{22.47}$$

The stability criterion is $\left|\lambda_{max}(\mathbf{A})\right| < 1$.

Biased form In biased form,

$$\hat{\mathbf{y}}[n] = \boldsymbol{\mu} + \mathbf{A}\mathbf{y}[n-1] + \boldsymbol{\epsilon}[n] \tag{22.48}$$

biases $\boldsymbol{\mu} = \begin{bmatrix} \mu_0 \\ \mu_1 \end{bmatrix}$.

VAP(p) Model

The further extension to p values history is straightforward,

$$\hat{\mathbf{y}}[n] = \mathbf{A}_1\mathbf{y}[n-1] + \mathbf{A}_2\mathbf{y}[n-2] + \ldots + \mathbf{A}_p\mathbf{y}[n-p]\boldsymbol{\epsilon}[n] \tag{22.49}$$

Again, biased version is possible.

Coefficients

LS solution is straightforward by the organization of $\mathbf{y}[n]$ values into the corresponding LS problem matrices.

Diagonal A matrix If matrix $\mathbf{A}$ is diagonal, the model reduces to a set of univariate independent models.

Trending The standard VAR models does not supply de-trending capabilities similar to the univariate ARI model. If de-trending is required, it is a part of signal pre-processing pipeline.

Non-linear extension The vector and non-linear extensions are not mutually exclusive. A NARX model (Sec. 22.8) whose output layer has N units maps the selected lags to the whole N-dimensional output at once,

$$\hat{\mathbf{y}}[n] = f\left(\mathbf{y}[n-1], \ldots, \mathbf{y}[n-p], \mathbf{x}[n-1], \ldots, \mathbf{x}[n-q]\right) + \boldsymbol{\epsilon}[n], \tag{22.50}$$

and reduces to VAR(p) when $f(\cdot)$ is linear, exactly as NAR(p) reduces to AR(p).

23. Statistical Inference

Goal: Quantify how much of an observed signal-processing estimate is signal versus sampling noise, and provide thresholds that separate the two.

Contents

23.1. Overview **363**
23.2. ACF **366**
23.2.1. Simple Case: White-Noise Signal 366
23.2.2. General Case: Bartlett's Formula 368
23.3. CCF **371**
23.3.1. Simple Case: at Least One White-Noise Signal 371
23.3.2. General Case: Bartlett's Formula 372
23.4. Cross-Coherence **374**
23.4.1. Hypothesis Test 375
23.4.2. Estimators 375
23.5. Granger Causality **376**
23.5.1. Test statistic 377
23.5.2. Relation to VAR 377
23.5.3. Granger causality comparisons 378

23.1. Overview

Every quantity computed from a finite record of length L (autocorrelation, cross-correlation, spectral density, fitted coefficient, multi-step forecast) is a random estimate, not a fixed number. A *confidence bound* is the threshold beyond which the estimate is unlikely to lie under a stated null hypothesis, allowing significant structure to be separated from finite-sample noise.

In the context of the ARMA and ARX chapters, several distinct confidence-bound problems arise. Their construction follows the same recipe (sampling distribution under H_0, plus a threshold $z_{\alpha/2}\sqrt{\mathrm{Var}\left[\cdot\right]}$), but the variance formula differs:

- **ACF bound** (Sec. 23.2). Used to identify MA order, validate model residuals, and decide whether a lag is significant. Variance under white-noise H_0: $1/L$ (Bartlett's formula in its simple form, generalized for autocorrelated processes).

- **CCF bound** (Sec. 23.3). Used in ARX model identification to pick relevant input lags. Bartlett's general formula accounts for the autocorrelation of both signals to avoid spurious detections.
- **PSD confidence interval**. The bound is *multiplicative* (asymmetric on linear scale, symmetric on dB scale) rather than additive.
- **Coefficient confidence intervals**. AR/ARMA/ARX parameter estimates from least squares or Yule-Walker have approximately Gaussian sampling distributions. The resulting interval $\hat{w}_i \pm z_{\alpha/2}\sqrt{\mathrm{Var}[\hat{w}_i]}$ tests whether a coefficient is meaningfully nonzero.
- **Forecast/prediction intervals**. The h-step-ahead prediction error variance grows with the horizon and depends on the model's MA representation; the resulting interval widens with h.

This chapter develops the ACF and CCF cases in detail.

Hypotheses

A confidence bound is most often used as a decision rule in a hypothesis test. Two complementary statements are formulated about the unknown parameter θ:

- **Null hypothesis** H_0: the "no effect" or baseline statement. In this chapter, H_0 typically asserts the absence of structure, e.g. $\rho_{\mathbf{xx}}[k] = 0$ (white-noise ACF), $\rho_{\mathbf{xy}}[k] = 0$ (independent signals), or $w_i = 0$ (an irrelevant model coefficient).
- **Alternative hypothesis** H_1: the complementary statement that some structure is present, e.g. $\rho_{\mathbf{xx}}[k] \neq 0$, $\rho_{\mathbf{xy}}[k] \neq 0$, or $w_i \neq 0$.

The sampling distribution of $\hat{\theta}$ (and hence its SE) is derived *under* H_0. The CI built from this distribution acts as the acceptance region: if the estimate $\hat{\theta}$ falls outside the bound, H_0 is rejected in favor of H_1 at significance level α; otherwise, H_0 is retained as consistent with the data.

Standard error vs. confidence interval

The sampling distribution of $\hat{\theta}$ under H_0 is summarized by two related but distinct quantities:

- The **standard error (SE)** is its standard deviation,

$$\mathrm{SE}(\hat{\theta}) = \sqrt{\mathrm{Var}\left[\hat{\theta} \mid H_0\right]}. \tag{23.1}$$

  It is a single non-negative number describing the typical spread of $\hat{\theta}$ around the value asserted by H_0, and depends only on the estimator and the sample size L, not on the chosen significance level α.
- The **confidence interval (CI)** at level $1 - \alpha$ is the acceptance region introduced above, expressed via the SE,

$$\hat{\theta} \pm z_{\alpha/2} \cdot \mathrm{SE}(\hat{\theta}), \tag{23.2}$$

  under the approximately Gaussian sampling distribution implied by the CLT. The half-width $c_\alpha = z_{\alpha/2} \cdot \mathrm{SE}(\hat{\theta})$ is the threshold whose violation rejects H_0.

The two are linked but not interchangeable:

- SE is a scale (one number); CI is an interval (two endpoints).
- Tightening the test (smaller α) widens the CI but leaves the SE unchanged.

- Increasing the sample size L shrinks the SE, and therefore the CI, at a rate proportional to $1/\sqrt{L}$.

Theoretical vs. estimated SE The SE in (23.1) is a population quantity: $\mathrm{Var}\left[\hat{\theta} \mid H_0\right]$ is computed from the true (typically unknown) distribution of the data, so the resulting expression involves population parameters such as the true ACFs $\rho_{\mathbf{xx}}[m]$, $\rho_{\mathbf{yy}}[m]$. We refer to this as the *theoretical SE* and write it $\mathrm{SE}(\hat{\theta})$. To actually evaluate a CI on data, the population parameters are replaced by their sample estimates (e.g. $\rho_{\mathbf{xx}}[m] \to R_{\mathbf{xx},norm}[m]$). The resulting plug-in quantity is the *estimated SE*,

$$\widehat{\mathrm{SE}}(\hat{\theta}) = \sqrt{\mathrm{Var}\left[\hat{\theta} \mid H_0\right]}\Big|_{\text{sample plug-in}}, \tag{23.3}$$

and is what enters the operational confidence bound $c_\alpha = z_{\alpha/2} \cdot \widehat{\mathrm{SE}}(\hat{\theta})$. The two coincide in the limit $L \to \infty$, but for finite L the estimated SE itself carries sampling error. In the white-noise case (e.g. the simple $1/\sqrt{L}$ rule) no plug-in is needed, since $\mathrm{Var}\left[\hat{\theta} \mid H_0\right]$ depends only on L; in Bartlett's formulas the distinction is non-trivial because the variance depends on unknown ACFs.

All bound formulas in this chapter (the simple $1/\sqrt{L}$ rule, Bartlett's formulas, coefficient and forecast intervals) are recipes for $\mathrm{SE}(\hat{\theta})$ under the relevant H_0; substituting sample quantities yields $\widehat{\mathrm{SE}}(\hat{\theta})$, and multiplying by $z_{\alpha/2}$ yields the CI half-width c_α used to accept or reject H_0.

Notation

Throughout the chapter we distinguish theoretical (population) quantities from their sample-based estimates:

- Normalized correlations:
  - $\rho_{\mathbf{xx}}[k]$, $\rho_{\mathbf{xy}}[k]$ denote the *population* correlation coefficients, defined as the $L \to \infty$ limit of the sample Cauchy–Schwarz form of Eq. (21.24),

$$\rho_{\mathbf{xx}}[k] \triangleq \lim_{L\to\infty} \frac{R_{\mathbf{xx}}[k]}{\sqrt{R_{\mathbf{xx}}[0]\,\sum_n x^2[n-k]}}, \tag{23.4}$$

    and similarly for $\rho_{\mathbf{xy}}[k]$. This is the standard time-series convention used in Bartlett's formulas below.
  - $R_{\mathbf{xx},norm}[k]$, $R_{\mathbf{xy},norm}[k]$ denote the sample estimates computed from a finite record of length L, as defined in Sec. 21.1 and Sec. 22.1. Under the stationarity approximation Eq. (21.19), the sample $\rho_{\mathbf{xx}}[k]$ of (21.24) reduces to $R_{\mathbf{xx},norm}[k]$, so the latter serves as the operational plug-in for $\rho_{\mathbf{xx}}[k]$ in this chapter.
- Standard error:
  - $\mathrm{SE}(\hat{\theta})$ for the theoretical SE (function of population parameters such as $\rho_{\mathbf{xx}}[m]$, $\rho_{\mathbf{yy}}[m]$)
  - $\widehat{\mathrm{SE}}(\hat{\theta})$ for the plug-in (estimated) SE in which population parameters are replaced by their sample counterparts; see (23.1) and (23.3).

Stationarity prerequisite

The ACF and CCF sampling distributions and the corresponding analysis are derived under second-order stationarity: constant mean, and ACF/CCF that depend only on the lag, not on absolute

time. For example, in the cross-correlation case, two independent random walks routinely produce large $R_{\mathbf{xy},norm}[k]$ (the spurious-regression effect).

23.2. ACF

Goal: Determine whether the estimated ACF (Sec. 21.1) at lag $k \neq 0$ is statistically significant or merely a result of random chance.

Practical summary

- White-noise series (or model residuals tested against white noise): compare $\left|R_{\mathbf{xx},norm}[k]\right|$ to the simple bound $z_{\alpha/2}/\sqrt{L}$ (from (23.6)).
- Autocorrelated series:
  - Pick a truncation lag M (rule of thumb $M \approx \lfloor\sqrt{L}\rfloor$ or $M \approx 10\log_{10} L$),
  - compute the plug-in Bartlett $\widehat{\mathrm{SE}}[k] = \sqrt{\frac{1}{L}\left(1 + 2\sum_{j=1}^{M} R^2_{\mathbf{xx},norm}[j]\right)}$,
  - bound $c_\alpha[k] = z_{\alpha/2}\,\widehat{\mathrm{SE}}[k]$ (Sec. 23.2.2).
- Decision rule: reject white-noise H_0 at lag k iff $\left|R_{\mathbf{xx},norm}[k]\right| > c_\alpha[k]$.
- Common choice $\alpha = 0.05$, $z_{\alpha/2} = 1.96$.

The ACF estimate at lag k averages products $x[n]x[n-k]$ across many time indices n. Even for pure white noise, where the theoretical ACF is zero for all $k \neq 0$, the *estimated* ACF will not be exactly zero due to finite sample effects. A confidence bound quantifies how large these random fluctuations can be, so that values exceeding the bound can be attributed to genuine correlation rather than chance.

The bounds below presuppose stationarity.

23.2.1. Simple Case: White-Noise Signal

Null hypothesis and distribution Consider the null hypothesis H_0: the underlying process is white noise, i.e., $\rho_{\mathbf{xx}}[k] = 0$ for all $k \neq 0$ (population coefficient, Eq. (23.4)). Under the assumptions of stationarity and weak dependence, the Central Limit Theorem (CLT) applies: for large L, the ACF estimate at each lag is approximately normally distributed, regardless of the original distribution of $x[n]$. Specifically, under H_0, the variance of the normalized sample ACF is approximately $\frac{1}{L}$ (a special case of Bartlett's formula derived below), so that

$$R_{\mathbf{xx},norm}[k] \sim \mathcal{N}\left(0,\ \frac{1}{L}\right), \quad k \neq 0. \tag{23.5}$$

The corresponding standard error is

$$\mathrm{SE} = \frac{1}{\sqrt{L}}. \tag{23.6}$$

Confidence bound For a significance level α, the confidence bound is $\pm c_\alpha$, where

$$c_\alpha = z_{\alpha/2} \cdot \mathrm{SE} = \frac{z_{\alpha/2}}{\sqrt{L}} \tag{23.7}$$

and $z_{\alpha/2}$ is the z-score (the number of standard deviations from the mean, (4.15)) obtained from the inverse of the standard normal CDF, $z_{\alpha/2} = \Phi^{-1}(1 - \alpha/2)$. Common values are

Confidence level	α	$z_{\alpha/2}$
90%	0.10	1.645
95%	0.05	1.960
99%	0.01	2.576

For the most common choice of $\alpha = 0.05$ (95% confidence), the bound becomes $c_{0.05} = 1.96/\sqrt{L}$.

Interpretation If the estimated ACF at a particular lag exceeds the confidence bound, i.e.,

$$\left|R_{\mathbf{xx},norm}[k]\right| > c_\alpha, \tag{23.8}$$

the null hypothesis is rejected, the correlation at that lag is statistically significant and unlikely to arise by chance alone. If the value remains within the bound, the observed correlation is consistent with white noise.

An example of the ACF with 95% and 99% confidence bounds for white noise is shown in Fig. 23.1.

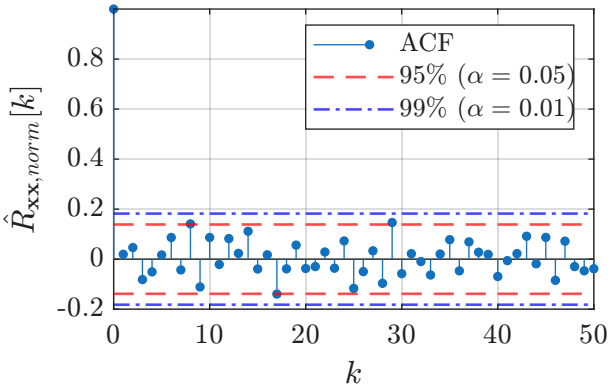


Figure 23.1.: Normalized ACF of white noise ($L = 200$) with 95% ($\alpha = 0.05$) and 99% ($\alpha = 0.01$) confidence bounds. A few values exceed the 95% bound, which is expected: testing 50 lags at $\alpha = 0.05$ yields approximately $50 \times 0.05 = 2.5$ false exceedances by chance. None exceed the stricter 99% bound.

The variance $1/L$ is a large-sample approximation. The exact (unapproximated) variance at lag k is $1/(L - |k|)$, giving a lag-dependent standard error and bound

$$\mathrm{SE}[k] = \frac{1}{\sqrt{L - |k|}}, \qquad c_\alpha[k] = z_{\alpha/2} \cdot \mathrm{SE}[k] = \frac{z_{\alpha/2}}{\sqrt{L - |k|}} \tag{23.9}$$

which widens at larger lags where fewer sample pairs contribute.

23.2.2. General Case: Bartlett's Formula

The simple $1/L$ variance assumes the process is white noise under H_0. When the process is auto-correlated (e.g., an MA or AR signal), this assumption breaks down: the simple bound becomes anti-conservative at higher lags and can flag spurious correlations.

Bartlett's formula (1946) corrects for this by accounting for the internal autocorrelation structure of the process. For a stationary process with population ACF $\rho_{\mathbf{xx}}[m]$ (the $L \to \infty$ limit, Eq. (23.4)), the variance of the normalized sample ACF at lag k is

$$\mathrm{Var}\left[R_{\mathbf{xx},norm}[k]\right] \approx \frac{1}{L}\sum_{m=-\infty}^{\infty}\Big(\rho_{\mathbf{xx}}^2[m]+\rho_{\mathbf{xx}}[m+k]\,\rho_{\mathbf{xx}}[m-k]-4\rho_{\mathbf{xx}}[k]\,\rho_{\mathbf{xx}}[m]\,\rho_{\mathbf{xx}}[m-k]+2\rho_{\mathbf{xx}}^2[k]\,\rho_{\mathbf{xx}}^2[m]\Big). \tag{23.10}$$

The four terms arise because the same series enters the estimator twice: the leading $\rho_{\mathbf{xx}}^2[m]$ matches the white-noise variance, the cross product $\rho_{\mathbf{xx}}[m+k]\rho_{\mathbf{xx}}[m-k]$ couples symmetric lags, and the remaining $\rho_{\mathbf{xx}}[k]$-dependent terms correct for the bias introduced when the true ACF at lag k is itself nonzero.

Time-limited (truncated) variant

The expression (23.10) is theoretical: it depends on the true ACF $\rho_{\mathbf{xx}}[m]$, which is unknown in practice. Two simplifications turn it into a usable formula.

First, restrict the sum. Beyond a finite lag the true ACF is negligible, so the infinite sum is truncated at a chosen lag M. Under H_0 at lag $k > 0$, where $\rho_{\mathbf{xx}}[k] = 0$, the $\rho[k]$-dependent corrections in (23.10) vanish; using ACF symmetry $\rho[-m] = \rho[m]$ and $\rho_{\mathbf{xx}}[0] = 1$, the theoretical SE becomes

$$\mathrm{SE}[k] = \sqrt{\frac{1}{L}\left(1 + 2\sum_{j=1}^{M}\rho_{\mathbf{xx}}^2[j]\right)}. \tag{23.11}$$

Second, replace the unknown $\rho_{\mathbf{xx}}[j]$ by its sample estimate $R_{\mathbf{xx},norm}[j]$. This yields the plug-in (estimated) SE actually computed from data,

$$\widehat{\mathrm{SE}}[k] = \sqrt{\frac{1}{L}\left(1 + 2\sum_{j=1}^{M}R_{\mathbf{xx},norm}^2[j]\right)}, \tag{23.12}$$

so only one-sided sample ACF lags need to be evaluated.

Choosing M The truncation lag M should be large enough to include all non-negligible ACF mass (past the visible decay of $R_{\mathbf{xx},norm}[m]$), but small enough that the included terms are not dominated by sampling noise. Typical rules of thumb are $M \approx 10\log_{10} L$ or $M = \lfloor\sqrt{L}\rfloor$.

Choosing L (absolute sample size) The truncation lag M is dictated by the process (how fast its ACF decays); the record length L must then satisfy two separate constraints, both expressed as absolute floors on L:

- *CLT validity for $R_{\mathbf{xx},norm}[k]$.* The Gaussian approximation underlying $z_{\alpha/2}$ thresholds is reliable only once L is large in absolute terms, independently of M. A practical floor is $L \gtrsim 50$–100; below this the sampling distribution at small lags is visibly non-Gaussian and the bounds are unreliable regardless of how the variance is estimated.
- *Truncation validity, $M \ll L$.* Inverting the rule of thumb $M \approx \lfloor\sqrt{L}\rfloor$ gives the lower bound $L \gtrsim M^2$. A safer engineering rule is $L \gtrsim 10M$ to $20M$, which keeps each included sample ACF $R_{\mathbf{xx},norm}[j]$ ($j \le M$) accurate enough that the plug-in $\widehat{\mathrm{SE}}[k]$ approaches the theoretical $\mathrm{SE}[k]$. Recall that an individual sample ACF has SE of order $1/\sqrt{L-j}$, so each squared term in (23.12) contributes its own $O(1/L)$ noise.

Combining the two:

$$L \;\geq\; \max\Big(50\text{–}100,\; 10M \text{ to } 20M,\; M^2\Big).$$

Concrete examples: $M = 3$ requires $L \gtrsim 50$–100 (the CLT floor dominates; the squared rule alone would give only $L \geq 9$, far too few); $M = 5$ requires $L \gtrsim 100$; $M = 10$ requires $L \gtrsim 100$–200 (the $10M$ rule now dominates). Small M is a feature of a short-memory process, not a license to shrink L: even when the simple bound is nearly correct, L must still be large enough for both the sampling distribution to be Gaussian and the plug-in $\widehat{\mathrm{SE}}$ to converge to SE.

MA(q) cutoff special case

A particularly clean special case arises when the process is MA(q), so $\rho_{\mathbf{xx}}[m] = 0$ for $|m| > q$. The truncation lag in (23.12) then collapses to $M = q$, and the variance of the normalized sample ACF at lag $k > q$ is

$$\mathrm{Var}\big[R_{\mathbf{xx},norm}[k]\big] \approx \frac{1}{L}\left(1 + 2\sum_{m=1}^{q} \rho_{\mathbf{xx}}^2[m]\right), \quad k > q, \tag{23.13}$$

giving the theoretical standard error

$$\mathrm{SE}[k] = \sqrt{\frac{1}{L}\left(1 + 2\sum_{m=1}^{q} \rho_{\mathbf{xx}}^2[m]\right)}, \quad k > q. \tag{23.14}$$

Substituting the sample ACFs $R_{\mathbf{xx},norm}[m]$ for the unknown $\rho_{\mathbf{xx}}[m]$ yields the plug-in estimate

$$\widehat{\mathrm{SE}}[k] = \sqrt{\frac{1}{L}\left(1 + 2\sum_{m=1}^{q} R_{\mathbf{xx},norm}^2[m]\right)}, \quad k > q, \tag{23.15}$$

and the corresponding confidence bound is

$$c_\alpha[k] = z_{\alpha/2} \cdot \widehat{\mathrm{SE}}[k] = z_{\alpha/2}\sqrt{\frac{1}{L}\left(1 + 2\sum_{m=1}^{q} R_{\mathbf{xx},norm}^2[m]\right)}, \quad k > q. \tag{23.16}$$

For white noise ($q = 0$) the sum vanishes and $\widehat{\mathrm{SE}} = 1/\sqrt{L}$, recovering the simple bound $c_\alpha = z_{\alpha/2}/\sqrt{L}$. The bound widens with the strength of the autocorrelation at lower lags, so that higher lags are tested against a more conservative threshold.

Choosing q The order q is rarely known in advance and is typically selected from the data:

- **Cutoff inspection (Box-Jenkins).** Plot the ACF with the simple bound. Set q to the largest lag at which $\left|R_{\mathbf{xx},norm}[m]\right|$ exceeds $z_{\alpha/2}/\sqrt{L}$, then re-evaluate higher lags using the Bartlett bound.
- **Recursive widening.** Start with $q = 0$; if lag 1 is significant, set $q = 1$ and re-test lag 2 with the wider bound. Continue until the next lag falls within the bound.
- **Conservative upper bound.** If no prior information is available, pick a safe upper limit (e.g., $q = \lfloor\sqrt{L}\rfloor$). This sacrifices statistical power but avoids underestimating the variance.

Example 23.1 (Bartlett bound for an AR(1) signal): Consider an AR(1) signal of length $L = 200$ with normalized sample ACF $R_{\mathbf{xx},norm}[1] = 0.6$. Apply Bartlett's MA($q$) approximation with $q = 1$ (treating higher-lag autocorrelations as negligible) and compute the 95% confidence bound at lag $k = 2$ under (a) the simple white-noise assumption and (b) Bartlett's formula.

Solution: With $z_{\alpha/2} = 1.96$.

1. Simple bound:
$$c_{0.05} = \frac{1.96}{\sqrt{200}} \approx 0.139.$$

2. Bartlett bound, with $q = 1$ and $R^2_{\mathbf{xx},norm}[1] = 0.36$:
$$c_{0.05}[2] = 1.96\sqrt{\frac{1 + 2(0.36)}{200}} = 1.96\sqrt{\frac{1.72}{200}} \approx 0.182.$$

The Bartlett bound is roughly 31% wider, reflecting that the lag-1 correlation inflates the sampling variability of $R_{\mathbf{xx},norm}[k]$ at higher lags. An estimated $R_{\mathbf{xx},norm}[2] = 0.16$ would be flagged as significant by the simple bound, but is consistent with white-noise residual behavior under the Bartlett bound.

A simulated AR(1) realization with $\phi = 0.6$ and $L = 200$ is shown in Fig. 23.2: for $k > 1$ the Bartlett bound (blue) widens above the constant simple bound (red), so several mid-lag fluctuations that pierce the simple threshold remain within the Bartlett threshold and are correctly attributed to chance.

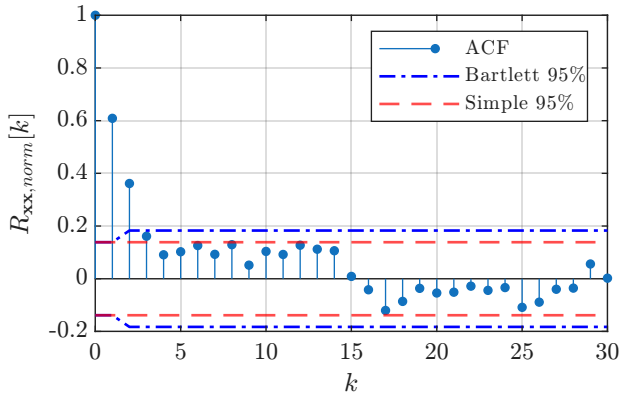


Figure 23.2.: Sample ACF of an AR(1) signal ($\phi = 0.6$, $L = 200$) with simple and Bartlett 95% confidence bounds. For $k > q = 1$ the Bartlett bound widens to account for the lag-1 autocorrelation, avoiding spurious detections that the simple bound would flag.

23.3. CCF

Goal: Determine whether the estimated CCF (Sec. 22.1) at lag k reflects a genuine relationship between $x[n]$ and $y[n]$, or is merely a result of random chance.

Practical summary

- At least one signal is white noise: compare $\left|R_{\mathbf{xy},norm}[k]\right|$ to the simple bound $z_{\alpha/2}/\sqrt{L-|k|}$ (from (23.18)).
- Both signals autocorrelated:
  - Pick a truncation lag M (rule of thumb $M \approx \lfloor\sqrt{L}\rfloor$ or $M \approx 10\log_{10} L$),
  - compute the plug-in Bartlett $\widehat{\mathrm{SE}} = \sqrt{\frac{1}{L}\left(1 + 2\sum_{j=1}^{M} R_{\mathbf{xx},norm}[j]\, R_{\mathbf{yy},norm}[j]\right)}$ from the one-sided sample ACFs of $\mathbf{x}$ and $\mathbf{y}$,
  - bound $c_\alpha[k] = z_{\alpha/2}\,\widehat{\mathrm{SE}}$ (eq. (23.24)).
- Conservative alternative: the decoupled bound (Sec. 23.3, "Conservative decoupled form"), which removes the joint ACF product at the cost of a wider threshold.
- Decision rule: reject independence H_0 at lag k iff $\left|R_{\mathbf{xy},norm}[k]\right| > c_\alpha[k]$.
- Common choice $\alpha = 0.05$, $z_{\alpha/2} = 1.96$.

Even for two independent signals, the *estimated* CCF will not be exactly zero due to finite sample effects. A confidence bound quantifies how large these random fluctuations can be. In practice, confidence intervals are computed for the normalized cross-correlation (bounded between -1 and 1), rather than the raw cross-covariance, which is scale-dependent. As with the ACF, the bounds below assume both series are stationary; on integrated data the spurious-regression effect produces large $R_{\mathbf{xy},norm}[k]$ even between independent inputs.

23.3.1. Simple Case: at Least One White-Noise Signal

Consider the null hypothesis H_0: the signals $x[n]$ and $y[n]$ are independent, i.e., $\rho_{\mathbf{xy}}[k] = 0$ for all k. If at least one of the signals is white noise (no autocorrelation), then the normalized sample CCF is approximately (analogous to the ACF case in Sec. 23.2)

$$R_{\mathbf{xy},norm}[k] \sim \mathcal{N}\left(0,\ \frac{1}{L-|k|}\right), \tag{23.17}$$

with standard error

$$\mathrm{SE}[k] = \frac{1}{\sqrt{L-|k|}}. \tag{23.18}$$

The confidence bound is $\pm c_\alpha[k]$, where

$$c_\alpha[k] = z_{\alpha/2} \cdot \mathrm{SE}[k] = \frac{z_{\alpha/2}}{\sqrt{L-|k|}}. \tag{23.19}$$

For large L, $L-|k| \approx L$, yielding the common approximation $\mathrm{SE} \approx 1/\sqrt{L}$ and $c_\alpha[k] \approx z_{\alpha/2}/\sqrt{L}$. If $\left|R_{\mathbf{xy},norm}[k]\right| > c_\alpha[k]$, the null hypothesis is rejected: the cross-correlation at lag k is statistically significant.

23.3.2. General Case: Bartlett's Formula

The simple $1/L$ variance assumes that at least one signal is white noise. When *both* $x[n]$ and $y[n]$ are autocorrelated (e.g., periodic), this assumption breaks down and can produce spurious correlations: falsely significant CCF values between two actually independent signals.

Bartlett's formula corrects for this by accounting for the internal autocorrelation structure of both signals, following the same theory→truncation→plug-in pattern as the ACF case in Sec. 23.2.2. Under H_0 (independence), the variance of the normalized sample CCF at lag k is

$$\operatorname{Var}\left[R_{\mathbf{xy},norm}[k]\right] \approx \frac{1}{L}\sum_{m=-\infty}^{\infty} \rho_{\mathbf{xx}}[m]\,\rho_{\mathbf{yy}}[m] \tag{23.20}$$

where $\rho_{\mathbf{xx}}[m]$ and $\rho_{\mathbf{yy}}[m]$ are the population autocorrelation coefficients of $x[n]$ and $y[n]$ at lag m (the $L \to \infty$ limit, Eq. (23.4)), respectively. The corresponding theoretical standard error is

$$\mathrm{SE} = \sqrt{\frac{1}{L}\sum_{m=-\infty}^{\infty} \rho_{\mathbf{xx}}[m]\,\rho_{\mathbf{yy}}[m]}. \tag{23.21}$$

If either signal is white noise (theoretically, $\rho_{\cdot}[m] = 0$ for $m \neq 0$), only the $m = 0$ term survives and $\mathrm{SE} = 1/\sqrt{L}$, recovering $c_\alpha[k] = z_{\alpha/2}/\sqrt{L}$.

Time-limited (truncated) variant

As in Sec. 23.2.2, the expression (23.21) is theoretical: it depends on the unknown true ACFs. Truncating the sum at a chosen lag M beyond which the ACFs are negligible gives the truncated theoretical SE,

$$\mathrm{SE} \approx \sqrt{\frac{1}{L}\sum_{m=-M}^{M} \rho_{\mathbf{xx}}[m]\,\rho_{\mathbf{yy}}[m]}. \tag{23.22}$$

Substituting the sample ACFs $R_{\mathbf{xx},norm}[m]$ and $R_{\mathbf{yy},norm}[m]$ for the unknown $\rho_{\mathbf{xx}}[m]$ and $\rho_{\mathbf{yy}}[m]$ yields the plug-in (estimated) SE

$$\widehat{\mathrm{SE}} \approx \sqrt{\frac{1}{L}\sum_{m=-M}^{M} R_{\mathbf{xx},norm}[m]\,R_{\mathbf{yy},norm}[m]}, \tag{23.23}$$

and using ACF symmetry $R[-m] = R[m]$ together with $R_{\mathbf{xx},norm}[0] = R_{\mathbf{yy},norm}[0] = 1$,

$$\widehat{\mathrm{SE}} \approx \sqrt{\frac{1}{L}\left(1 + 2\sum_{j=1}^{M} R_{\mathbf{xx},norm}[j]\,R_{\mathbf{yy},norm}[j]\right)}, \tag{23.24}$$

so only one-sided sample ACF lags need to be evaluated. The confidence bound is $c_\alpha[k] = z_{\alpha/2}\cdot\widehat{\mathrm{SE}}$.

Choosing M and L The same considerations as in the ACF case (Sec. 23.2.2, "Choosing M" and "Choosing L") apply, with M now required to exceed the visible ACF decay of *both* signals (e.g., Fig. 23.3a and 23.3b). The rules of thumb $M \approx \lfloor\sqrt{L}\rfloor$ or $M \approx 10\log_{10} L$ and the absolute floor $L \geq \max(50\text{–}100,\ 10M \text{ to } 20M,\ M^2)$ carry over unchanged.

Conservative decoupled form

A cleaner conservative form of the plug-in (estimated) SE is sometimes used in practice, since it removes the need to evaluate the joint ACF product. Starting from (23.23),

$$\widehat{\mathrm{SE}}^2 = \frac{1}{L} \sum_{m=-M}^{M} R_{\mathbf{xx},norm}[m]\, R_{\mathbf{yy},norm}[m],$$

apply the elementary inequality (valid lag-by-lag, in absolute value)

$$\left| R_{\mathbf{xx},norm}[m]\, R_{\mathbf{yy},norm}[m] \right| \le R_{\mathbf{xx},norm}[m]^2 + R_{\mathbf{yy},norm}[m]^2,$$

and use ACF symmetry $R[-m] = R[m]$ together with $R_{\mathbf{xx},norm}[0] = R_{\mathbf{yy},norm}[0] = 1$:

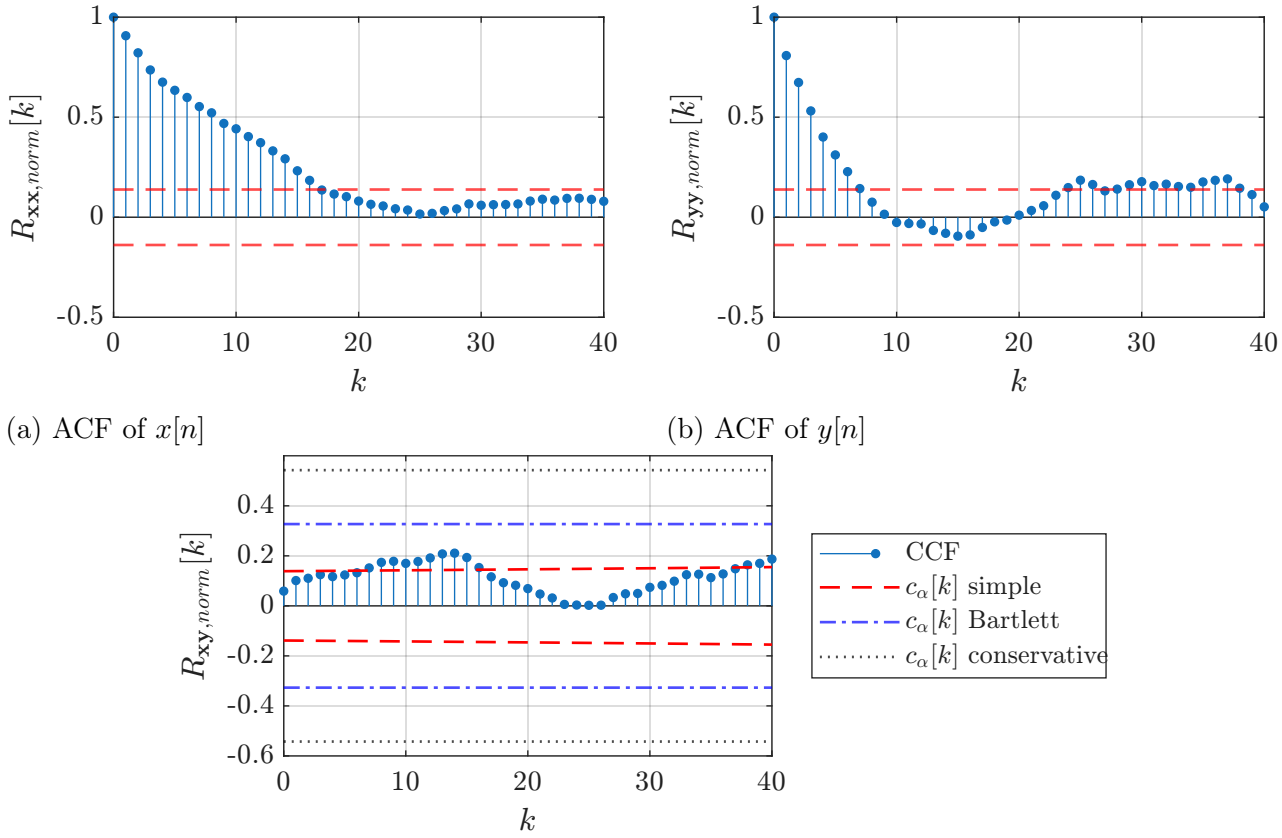
$$\sum_{m=-M}^{M} R_{\mathbf{xx},norm}[m]^2 = 1 + 2 \sum_{j=1}^{M} R_{\mathbf{xx},norm}[j]^2,$$

and likewise for $\mathbf{y}$. Combining the two terms,

$$\widehat{\mathrm{SE}} \le \frac{1}{\sqrt{L}} \sqrt{1 + 2 \sum_{j=1}^{M} R_{\mathbf{xx},norm}[j]^2 + 2 \sum_{j=1}^{M} R_{\mathbf{yy},norm}[j]^2}\,.$$

The bound decouples into two independent one-sided ACF sums, so each series can be evaluated in isolation.

Example 23.2: Fig. 23.3 illustrates the difference: two independent but autocorrelated signals are generated, and their CCF is plotted against the simple, Bartlett, and conservative one-sided confidence bounds. The simple bound (narrow) falsely flags several lags as significant; the Bartlett bound widens to account for the autocorrelation and avoids the spurious detections; the conservative one-sided bound is wider still, since it replaces the joint ACF product by the sum of squared single-series ACFs.



(a) ACF of $x[n]$

(b) ACF of $y[n]$

(c) CCF with simple, Bartlett, and conservative bounds

Figure 23.3.: Two *independent* AR(1) signals ($L = 200$): (a) and (b) show that each signal is strongly autocorrelated, violating the white-noise assumption of the simple bound; (c) the resulting CCF crosses the simple bound (red dashed, $\approx \pm 0.14$) at several lags (false positives), while the Bartlett bound (blue dash-dotted, $\approx \pm 0.33$) correctly accounts for the internal autocorrelation of both signals. The conservative one-sided bound (black dotted, $\approx \pm 0.55$) is even looser.

23.4. Cross-Coherence

Goal: Decide whether the estimated cross-coherence (Sec. 22.2.2) at frequency bin k reflects a stable linear coupling between $x[n]$ and $y[n]$, or is merely a result of finite-sample noise.

Practical summary

- Estimate $\hat{\gamma}^2_{\mathbf{xy}}[k]$ by averaging M quasi-independent spectral replicates (Welch segments or multitaper tapers); for $M = 1$ the estimate is identically 1 and uninformative.
- Compare $\hat{\gamma}^2_{\mathbf{xy}}[k]$ to the threshold $\gamma^2_{th} = 1 - \alpha^{1/(M-1)}$ from (23.27).
- Decision rule: reject independence H_0 at bin k iff $\hat{\gamma}^2_{\mathbf{xy}}[k] > \gamma^2_{th}$.
- For Welch with 50% Hann overlap, replace M by $M_{eff} \approx M/1.9$.

The true cross-coherence (Sec. 22.2.2) is a population quantity and must be estimated from a finite record. Both common estimators have the same form: average M quasi-independent replicates of

the cross- and auto-spectra and combine them as

$$\hat{\gamma}^2_{\mathbf{xy}}[k] = \frac{\left|\hat{S}_{\mathbf{xy}}[k]\right|^2}{\hat{S}_{\mathbf{xx}}[k]\,\hat{S}_{\mathbf{yy}}[k]}. \tag{23.25}$$

For $M = 1$ the Cauchy-Schwarz inequality saturates and $\hat{\gamma}^2_{\mathbf{xy}}[k] \equiv 1$ identically, so $M > 1$ is mandatory; the role of M is to inject independent realizations of the spectra into the ratio.

23.4.1. Hypothesis Test

The null hypothesis is the absence of linear coupling at bin k,

$$H_0 :\ \gamma^2_{\mathbf{xy}}[k] = 0, \qquad H_1 :\ \gamma^2_{\mathbf{xy}}[k] > 0. \tag{23.26}$$

For two independent Gaussian signals with M independent spectral replicates, the null distribution of $\hat{\gamma}^2_{\mathbf{xy}}[k]$ admits a closed-form upper quantile, and the $(1-\alpha)$ significance threshold is

$$\gamma^2_{th} = 1 - \alpha^{1/(M-1)}. \tag{23.27}$$

Reject H_0 at bin k iff $\hat{\gamma}^2_{\mathbf{xy}}[k] > \gamma^2_{th}$. The threshold decreases as M grows, so more averaging makes weaker couplings detectable; the cost is frequency resolution, which depends on the chosen estimator.

23.4.2. Estimators

Welch The classical estimator partitions the record into M overlapping segments of length N_{seg}, applies a window, and averages the resulting periodograms (Sec. 20.7.4). The frequency resolution is $\Delta f = f_s/N_{seg}$. With 50% Hann overlap the segments are not strictly independent: the standard correction is to replace M in (23.27) by an effective count $M_{eff} \approx M/1.9$, which avoids an optimistic threshold.

Multitaper The multitaper estimator keeps the record intact and multiplies it by K orthogonal Slepian (DPSS) tapers, averaging the resulting tapered cross- and auto-spectra,

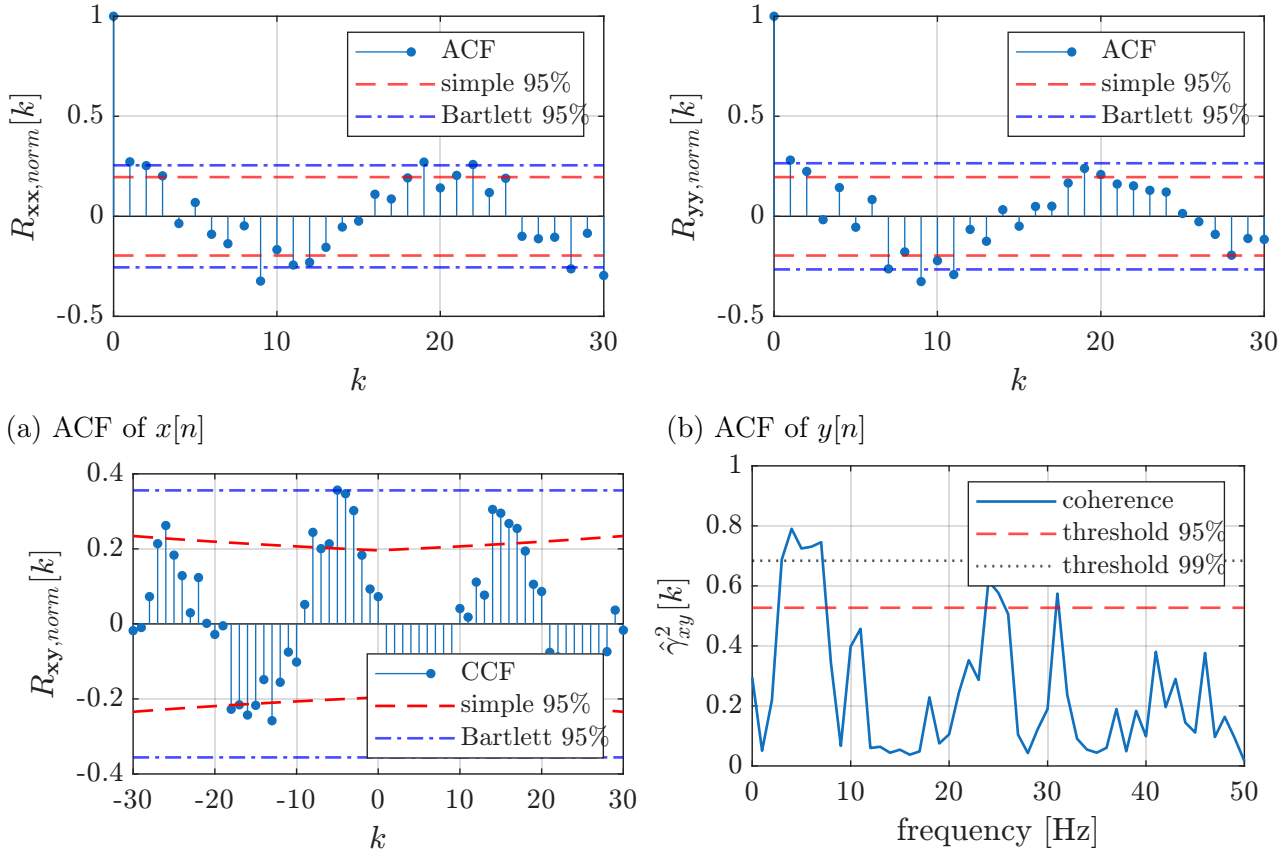
$$\hat{S}_{\mathbf{xy}}[k] = \frac{1}{K}\sum_{j=1}^{K} X^{(j)}[k]^*\, Y^{(j)}[k], \qquad \hat{S}_{\mathbf{xx}}[k] = \frac{1}{K}\sum_{j=1}^{K} \left|X^{(j)}[k]\right|^2, \tag{23.28}$$

where $X^{(j)}[k]$ is the DFT of $x[n]$ multiplied by the j-th DPSS taper of time-bandwidth product NW (typically $K = 2NW - 1$). The tapers are orthogonal by construction, so each acts as a quasi-independent realization and (23.27) applies with M replaced by K. Multitaper preserves the original frequency resolution (no segmentation) and is *preferred when the record is short.*

Example 23.3 (Multitaper cross-coherence of delay-coupled sinusoids): Reuse the noisy delay-coupled sinusoid pair of Fig. 22.3, with $f_0 = 5$ Hz, $f_s = 100$ Hz, $L = 100$, and $y[n]$ a copy of $x[n]$ delayed by $n_0 = 6$ samples plus independent noise. The cross-coherence is estimated with $K = 2NW - 1 = 5$ DPSS tapers ($NW = 3$), and the corresponding test bounds are

$$\gamma^2_{th}(\alpha) = 1 - \alpha^{1/(K-1)}, \quad \gamma^2_{th}(0.05) \approx 0.527, \quad \gamma^2_{th}(0.01) \approx 0.684.$$

Each individual signal is itself strongly autocorrelated (the cosine envelope dominates the noise), so the simple $1/\sqrt{L}$ ACF bound flags many spurious lags; the time-limited Bartlett bound with $M = \lfloor \sqrt{L} \rfloor = 10$ widens to absorb that structure, as shown in Fig. 23.4a and 23.4b. The cross-correlation Fig. 23.4c peaks near lag $k = n_0 = 6$ as expected from the delay; the simple bound flags many spurious lags, while the Bartlett joint bound absorbs the autocorrelation of both signals and leaves only the delay peak above threshold. In the frequency domain the same coupling shows up as a coherence peak: Fig. 23.4d shows that $\hat{\gamma}^2_{\mathbf{xy}}[k]$ pierces both the 95% and 99% thresholds at the sinusoid frequency $f_0 = 5$ Hz and stays below them elsewhere.



(a) ACF of $x[n]$

(b) ACF of $y[n]$

(c) CCF $R_{\mathbf{xy},norm}[k]$

(d) Multitaper cross-coherence

Figure 23.4.: Delay-coupled sinusoid pair of Fig. 22.3 ($L = 100$, $NW = 3$, $K = 5$): (a) and (b) sample ACFs with the simple $1/\sqrt{L}$ bound (red dashed) and the time-limited Bartlett bound with $M = 10$ (blue dash-dotted); (c) sample CCF with simple and Bartlett joint 95% bounds, peaking at $k = n_0 = 6$; (d) multitaper cross-coherence with the $\alpha = 0.05$ and $\alpha = 0.01$ test bounds, ≈ 0.527 and ≈ 0.684, peaking at $f_0 = 5$ Hz.

23.5. Granger Causality

Goal: Decide whether the past of one signal $x[n]$ improves the prediction of another signal $y[n]$ beyond y's own past: a directional, predictive notion of causality.

The null hypothesis states that the lagged-x coefficients are jointly zero,

$$\begin{aligned} H_0 &: \; b_1 = b_2 = \cdots = b_p = 0, \\ H_1 &: \; \exists\, i : \; b_i \neq 0. \end{aligned} \tag{23.29}$$

Assumption: both $x[n]$ and $y[n]$ are stationary.

23.5.1. Test statistic

The *restricted* model predicts $y[n]$ from its own past only, i.e. the AR(p) model of Sec. 21.3,

$$\hat{y}[n] = \sum_{i=1}^{p} a_i \, y[n-i], \tag{23.30}$$

with residual sum of squares SSE_r. The *unrestricted* model adds the past of $x[n]$,

$$\hat{y}[n] = \sum_{i=1}^{p} a_i \, y[n-i] + \sum_{i=1}^{p} b_i \, x[n-i], \tag{23.31}$$

with residual sum of squares SSE_u.

Because the unrestricted model nests the restricted one, adding the x-regressors cannot increase the residual, so $\mathrm{SSE}_u \leq \mathrm{SSE}_r$; the test asks whether this reduction is larger than sampling noise alone would produce.

Under H_0 the nested-model F-statistic is

$$F = \frac{(\mathrm{SSE}_r - \mathrm{SSE}_u)/p}{\mathrm{SSE}_u/(L-2p-1)} \tag{23.32}$$

Practical summary

- Fit the *restricted* model: AR(p) of $y[n]$ on its own p lags, giving residual sum of squares SSE_r.
- Fit the *unrestricted* model: add the p past lags of $x[n]$, giving SSE_u.
- Form the statistic
$$F = \frac{(\mathrm{SSE}_r - \mathrm{SSE}_u)/p}{\mathrm{SSE}_u/(L-2p-1)} \sim F_{p,\,L-2p-1}$$
- Reject "x does not Granger-cause y" iff $F > F_\alpha$.
- Run the test in both directions (Sec. 23.5.3).

Granger causality formalizes a *predictive* sense of causality: $x[n]$ is said to Granger-cause $y[n]$ if the past of x helps forecast y better than the past of y alone. It is a statement about forecast improvement, not about a physical mechanism.

23.5.2. Relation to VAR

The paired tests are exactly the coefficient tests of a bivariate VAR(p) (Sec. 22.9): stacking the two signals into $\mathbf{y}[n] = [x[n],\, y[n]]^\top$, channel j does not Granger-cause channel i iff $(\mathbf{A}_m)_{ij} = 0$ for all lags $m = 1, \ldots, p$.

23.5.3. Granger causality comparisons

Causality is directional, so the test is run twice with the roles of the two signals swapped:

- $x \to y$: restricted = AR(p) of y on its own past; unrestricted adds lagged x. Null $H_0^{x\to y}$: the lagged-x coefficients in the y equation are all zero.
- $y \to x$: the symmetric test on the x equation. Null $H_0^{y\to x}$: the lagged-y coefficients in the x equation are all zero.

The two outcomes combine into four mutually exclusive conclusions, Table 23.1.

Table 23.1.: Granger causality comparisons: the four directional conclusions from the paired tests.

$x \to y$	$y \to x$	Conclusion
reject	retain	unidirectional $x \to y$
retain	reject	unidirectional $y \to x$
reject	reject	bidirectional feedback $x \leftrightarrow y$
retain	retain	no Granger causality (independence)

Example 23.4 (Granger F-test): A record of length $L = 200$ is modeled with $p = 2$ lags. The restricted (own-past) fit gives $\mathrm{SSE}_r = 120$ and the unrestricted fit (adding the past of x) gives $\mathrm{SSE}_u = 100$. Then

$$F = \frac{(120 - 100)/2}{100/(200 - 2 \cdot 2 - 1)} = \frac{10}{100/195} \approx 19.5$$

Since $F_{0.05} \approx 3.04$, the statistic far exceeds the threshold and H_0 is rejected: x Granger-causes y. A simulated coupled pair is shown in Fig. 23.5.

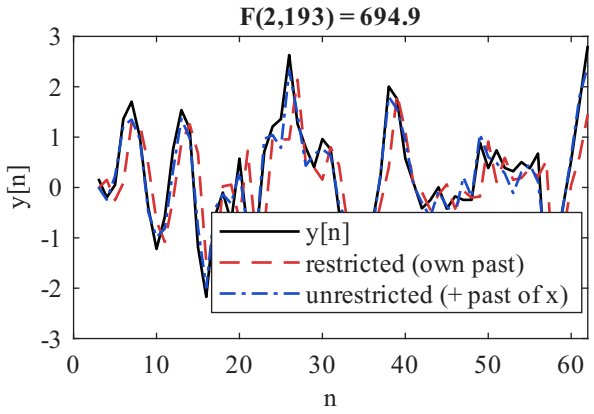


Figure 23.5.: Granger causality on a simulated pair where $x[n]$ drives $y[n]$ with a lag. Adding the past of x to the own-past AR model of y markedly reduces the prediction residual, and the resulting F-statistic exceeds the significance threshold.

- Granger causality is *predictive*, not mechanistic: it detects forecast improvement, not a physical cause.
- The bivariate test can be misled by a latent third driver that leads both signals; partial out the other channels with a conditional (multivariate) test on the full VAR.

- It presupposes stationarity; on trending or integrated data, difference first, otherwise the spurious-regression effect inflates significance.
- It is linear by construction and captures only *lagged* coupling, not instantaneous (same-sample) dependence.
- Conclusions depend on the lag order p; select it by AIC/BIC.

Further Reading

- Stationarity tests:
  - Unit Root Tests example, Matlab documentation (accessed 29-Apr-2026).
  - Augmented Dickey–Fuller (ADF) test
  - Kwiatkowski–Phillips–Schmidt–Shin (KPSS) test
  - Phillips–Perron (PP) test
- White noise test:
  - Ljung-Box test
- Granger Causality
  - Granger Causality, ApTech (accessed 31-May-2026).

24. Regression Metrics

Goal: Quantify prediction quality and compare models fairly across scales and datasets.

Given targets $y[n]$ and predictions $\hat{y}[n]$, $n = 0, \ldots, L-1$, the prediction error (residual) is

$$e[n] = y[n] - \hat{y}[n]. \tag{24.1}$$

Every metric below is a summary of these same L residuals.

24.1. Scale-Dependent Metrics

Scale-dependent metrics retain the original units of the data, making them appropriate when the size of the error, or the direction it leans, is meaningful in itself.

Mean squared error (MSE) and root mean squared error (RMSE)

$$\mathrm{MSE} = \frac{1}{L}\sum_{n} e^2[n], \qquad \mathrm{RMSE} = \sqrt{\mathrm{MSE}}. \tag{24.2}$$

Heavily penalizes large errors; sensitive to outliers. MSE is reported in squared units and RMSE returns to the units of the data. MSE is also commonly used as a training loss (differentiable, convex).

Mean absolute error (MAE) and median absolute error (MedAE)

$$\mathrm{MAE} = \frac{1}{L}\sum_{n} |e[n]|, \qquad \mathrm{MedAE} = \mathrm{median}_n\ |e[n]|. \tag{24.3}$$

Robust to outliers (especially MedAE). MAE is also used as a training loss (convex, but nondifferentiable at 0).

Mean error (ME)

$$\mathrm{ME} = \frac{1}{L}\sum_{n} e[n]. \tag{24.4}$$

Residuals of opposite sign cancel, so ME can sit at zero while the predictions are worthless; it is read next to MSE, never alone. The pair is tied together by

$$\mathrm{MSE} = \mathrm{ME}^2 + \frac{1}{L}\sum_{n}\Big(e[n] - \mathrm{ME}\Big)^2, \tag{24.5}$$

an exact identity splitting the MSE into a systematic offset and the spread around it, which is the sample form of the bias-variance split of Eq. (4.13).

Where ME says nothing

Two situations in which a zero ME carries no information:

- On the *training* residuals of a least-squares fit with an intercept, ME $= 0$ by construction (Sec. 3.3.3), whatever the quality of the fit. The metric is informative only on held-out data.
- ME is the bias of one predictor on one record, not the Bias$[\hat{y}]$ of Sec. 4.4, which averages over training sets at unseen points. A model can hold ME $= 0$ on every run and still be badly biased in that sense.

Example 24.1 (A six-sample record): A predictor is evaluated on $L = 6$ samples:

n	0	1	2	3	4	5
$y[n]$	10	12	9	11	13	10
$\hat{y}[n]$	9	14	9	10	14	5
$e[n]$	1	−2	0	1	−1	5

The residuals of the last row are the only quantities that need to be computed; each metric reuses them. Squaring and summing gives $\sum_n e^2[n] = 32$, so MSE $= 32/6 = 5.33$ and RMSE $= 2.31$. Summing the magnitudes gives $\sum_n |e[n]| = 10$, so MAE $= 10/6 = 1.67$. Sorting those magnitudes, $0, 1, 1, 1, 2, 5$, gives MedAE $= 1$.

The single bad sample at $n = 5$ contributes 25 of the 32 squared units, which is why RMSE lands 39% above MAE, while MedAE reports the typical sample and does not see that failure at all.

Example 24.2 (Two predictors, one target): Two predictors are compared on the same target signal (Fig. 24.1). Predictor A has intermediate, uniformly distributed errors. Predictor B is more accurate on most samples but contains three large outliers. The bar chart shows how different metrics rank the two: RMSE penalizes the outliers heavily (B is much worse), MAE is similar for both, and MedAE favors B (ignoring the outliers entirely). This illustrates why reporting more than one metric provides a more complete picture of prediction quality.

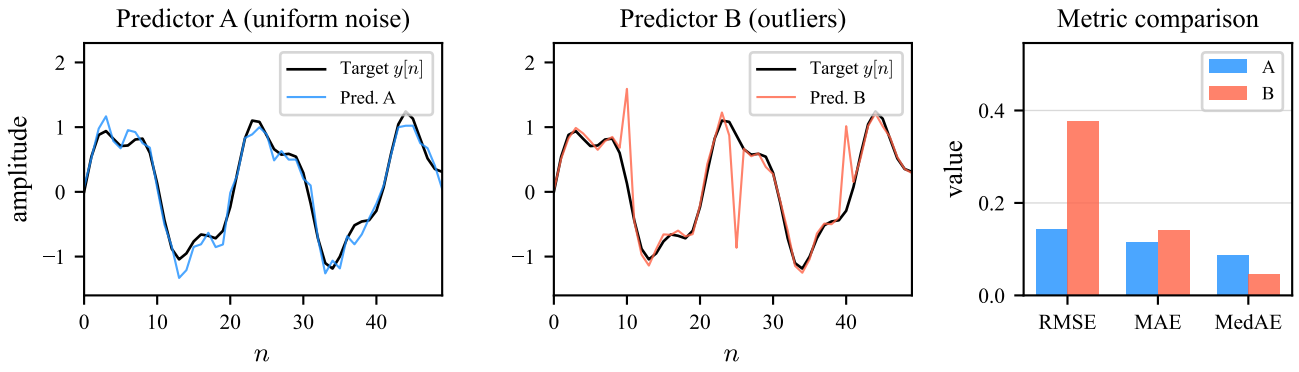


Figure 24.1.: Effect of outliers on metric values. Predictor A has uniform noise; Predictor B has sparse large errors. RMSE is sensitive to outliers, MAE is moderate, and MedAE is robust.

What these numbers say

All three metrics answer "how large is the error, in the units of the data":

- RMSE $\geq$ MAE always holds, and the size of that gap measures how unevenly the error is spread over the record.
- RMSE together with MedAE brackets the question of outliers.

24.2. Scale-Free Metrics

When comparing forecasts across series with different units or scales, scale-free metrics normalize the error so that results are comparable.

Mean absolute percentage error (MAPE) and symmetric MAPE (sMAPE)

$$\mathrm{MAPE} = \frac{100}{L}\sum_{n}\frac{|e[n]|}{|y[n]|+\varepsilon}, \qquad \mathrm{sMAPE} = \frac{100}{L}\sum_{n}\frac{2|e[n]|}{|y[n]|+|\hat{y}[n]|+\varepsilon}. \tag{24.6}$$

A small ε is added to avoid division by zero. Because MAPE divides by the target alone, the same absolute error is charged differently at different targets: an error of 40 is 40% against $y = 100$ but 300% against $y = 20$, as tabulated in Table 6.3. The charge is also one-sided, since a prediction cannot fall below zero: under-prediction saturates at 100%, while over-prediction is unbounded. sMAPE divides instead by the mean of target and prediction, which bounds it in $[0, 200]$%, but the bound is reached rather than avoided: as $y[n] \to 0$ the denominator collapses onto $|\hat{y}[n]|$ and the term saturates at 200% however small the absolute error, as the left panel of Fig. 24.2 shows.

Mean absolute scaled error (MASE)

MASE normalizes the MAE by the error of an in-sample naive forecast ($\hat{y}[n] = y[n-1]$):

$$\mathrm{MASE} = \frac{\frac{1}{L}\sum_{n}|e[n]|}{\frac{1}{L-1}\sum_{n=1}^{L-1}|y[n]-y[n-1]|}. \tag{24.7}$$

MASE < 1 means the model outperforms the naive baseline. The metric is robust across scales and can be extended to a seasonal naive denominator when seasonality is present. How demanding that baseline is depends on the signal: on a strongly correlated record the naive forecast is already close to optimal (Sec. 21.3).

Normalized RMSE (NRMSE)

NRMSE normalizes the RMSE by a measure of the target's spread:

$$\mathrm{NRMSE}_{\sigma} = \frac{\mathrm{RMSE}}{\sigma_y}, \qquad \mathrm{NRMSE}_{\mathrm{range}} = \frac{\mathrm{RMSE}}{\max y - \min y}. \tag{24.8}$$

Which denominator to trust

A scale-free metric is only as defensible as the quantity it divides by:

- MAPE divides by the target itself, which fails whenever the target passes near zero.
- sMAPE divides by the mean of target and prediction, which bounds the score without rescuing it: near a zero target the denominator follows the prediction down, and the term pins to 200% however small the absolute error.
- MASE divides by the naive one-step forecast, so it answers "better than repeating the last sample?".
- NRMSE_σ divides by the spread of the record, so it answers "better than predicting the mean?": the constant $\bar{y}$ scores exactly 1.
- $\text{NRMSE}_{\text{range}}$ divides by $\max y - \min y$, a figure two samples decide between them. It places the error on a convenient scale but carries no baseline, so no value of it passes or fails.

When a metric is quoted without its baseline, the number is not comparable across studies.

Example 24.3 (The same record, scale-free): The residuals of Example 24.1 are reused, so only the denominators change:

Metric	Divided by	Value
MAPE	$\lvert y[n]\rvert$, per sample	15.6%
sMAPE	mean of $\lvert y[n]\rvert$ and $\lvert \hat{y}[n]\rvert$, per sample	18.3%
MASE	naive step, $12/5 = 2.4$	0.69
NRMSE_σ	$\sigma_y = 1.34$	1.72
$\text{NRMSE}_{\text{range}}$	$\max y - \min y = 4$	0.58

The first two normalize each sample before averaging; the last three divide a single aggregate, MAE or RMSE, by one number.

Percentage errors. The sample at $n = 5$ alone accounts for more than half of MAPE, its ratio $|e|/|y| = 0.50$ against 0.17 or less everywhere else. sMAPE is the larger of the two here because that prediction lies below the target, so the symmetric denominator shrinks with it.

Naive baseline. MASE lands below 1, so the model beats the one-step naive forecast.

Spread. NRMSE_σ exceeds 1, meaning the errors are larger than the spread of the record itself, so predicting the constant $\bar{y}$ would have scored better, while $\text{NRMSE}_{\text{range}}$ reports a comfortable 0.58. Two baselines, two verdicts: which one is quoted decides whether this predictor looks good.

Rescaling. Multiplying both $y[n]$ and $\hat{y}[n]$ by 100, as if the record were reported in centimeters instead of meters, multiplies RMSE, MAE and MedAE by 100 and MSE by 100^2, and leaves MAPE, sMAPE, MASE and NRMSE untouched, as the right panel of Fig. 24.2 shows.

24.3. Information Criteria

Goal: Compare models of different complexity by penalizing the number of parameters, using the MSE as the goodness-of-fit term.

The metrics above rank predictions; information criteria rank *models*, charging each one for the parameters it spends. They were introduced in Sec. 6.5, together with Akaike's final prediction

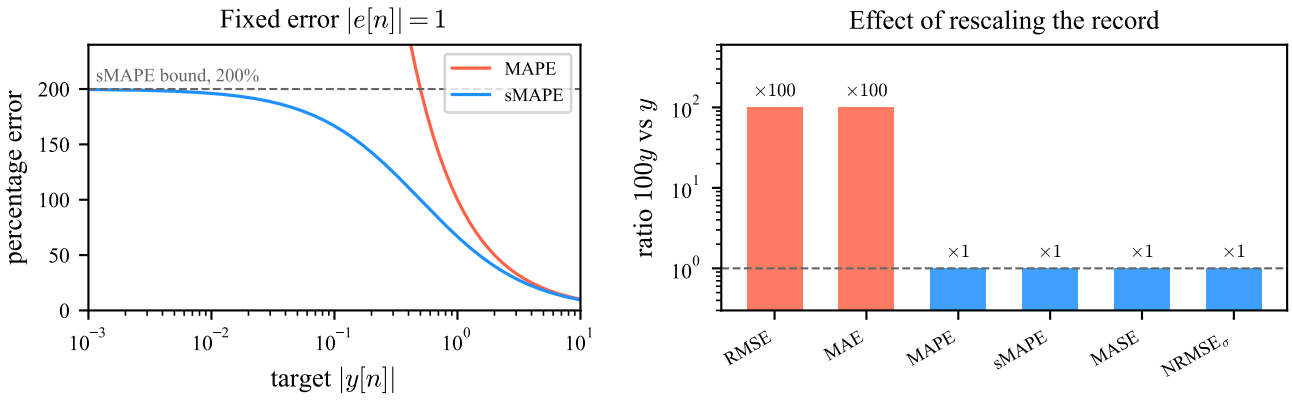


Figure 24.2.: Left: a fixed absolute error $|e[n]| = 1$ evaluated against a shrinking target, on a logarithmic target axis. MAPE diverges as $|y[n]| \to 0$ and leaves the frame, while sMAPE flattens onto its 200% bound. Right: the record of Example 24.1 rescaled by 100; each bar is the ratio of the metric on $100y$ to the metric on y.

error (FPE); this section applies them to the choice of a model order for a time series.

Adding parameters always improves the training MSE, which encourages overfitting. Information criteria replace the raw MSE with a penalized score that balances goodness-of-fit against the *number of parameters* k, given L samples:

$$\mathrm{AIC} = L\ln(\mathrm{MSE}) + 2k, \qquad \mathrm{AICc} = \mathrm{AIC} + \frac{2k(k+1)}{L-k-1}, \qquad \mathrm{BIC} = L\ln(\mathrm{MSE}) + k\ln L. \tag{24.9}$$

Lower values indicate a better model. The $L\ln(\mathrm{MSE})$ term rewards fit, and the second term charges for complexity: Akaike's information criterion (AIC) charges 2 per parameter; its corrected version (AICc) should be preferred when L/k is small; the Bayesian information criterion (BIC) charges $\ln L$ per parameter and therefore favors simpler models as the record grows. All three assume the residuals are approximately Gaussian.

Example 24.4 (Choosing an AR order): An auto-regressive (AR) process of order 2 (Sec. 21.3),

$$y[n] = 1.2\,y[n-1] - 0.6\,y[n-2] + \epsilon[n], \qquad \sigma_\epsilon = 1,$$

is simulated for 300 samples (Fig. 24.3, left). Candidate AR(p) models with $p = 1, \ldots, 10$ are fitted by least squares on the same $N = 290$ rows, so their residuals are comparable, and each model is charged $k = p + 1$ parameters: p coefficients plus the noise variance. Evaluating Eq. (24.9) with $L = N = 290$ gives the working forms $N\ln(\mathrm{MSE}) + 2k$ and $N\ln(\mathrm{MSE}) + k\ln N$.

The training MSE (dashed, right panel) falls from 1.55 at $p = 1$ to 0.93 at $p = 10$ and never marks a stopping point. Both criteria drop steeply from $p = 1$, where they reach 132 and 139 and leave the plotted range, to a minimum at $p = 2$, the true order, and rise afterwards: every additional coefficient buys less fit than the 2 (AIC) or $\ln N = 5.7$ (BIC) it costs. The BIC curve rises about three times faster, which is why on any record long enough that $\ln N > 2$ the BIC order cannot exceed the AIC order.

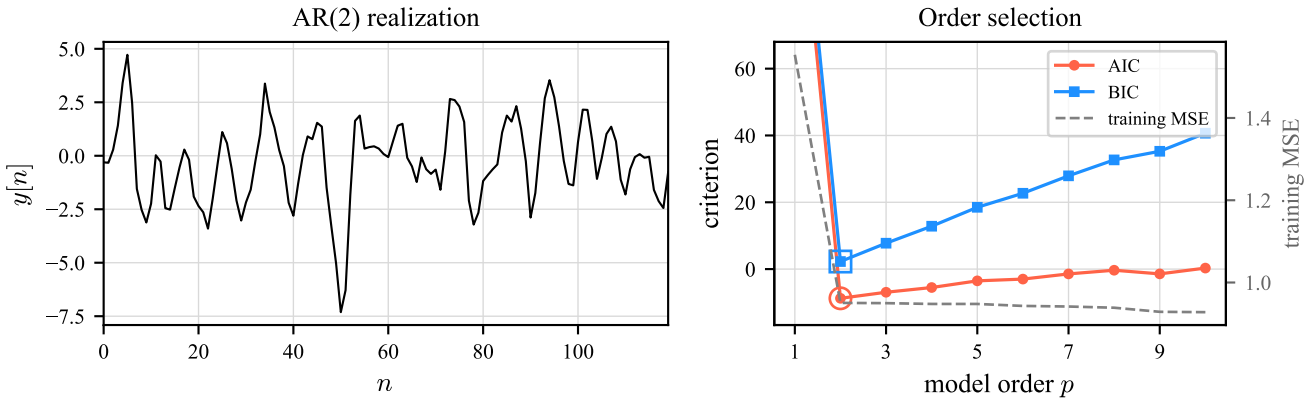


Figure 24.3.: Order selection for an AR(2) record. Left: the simulated signal. Right: AIC and BIC over candidate orders p, with the selected order circled, against the training MSE, which keeps falling. The $p = 1$ values lie above the plotted range.

Using the criteria

The criteria cost nothing once the model is fitted: they need only the achieved MSE and the parameter count, not a second fit or a held-out split. Their values are meaningless in isolation and comparable only between models fitted on the same samples, so a single number such as "AIC $= -8.8$" should never be reported alone. The Gaussian-residual assumption is their main limitation, and AIC and BIC can disagree, in which case the disagreement itself is the message: the data do not settle the order.

24.4. Summary

Table 24.1 collects the properties that decide which metric to report. The two questions it answers are whether a metric survives a change of units, and what a few bad samples do to it. A defensible report combines one metric in the units of the data with one scale-free metric, and states the baseline the second one uses.

Table 24.1.: Properties of the regression metrics of this chapter.

Metric	Eq.	Units	Scale-free	Outlier-robust	Also used as loss	Main pitfall
MSE	(24.2)	data^2	No	No	Yes	Dominated by the largest residual
RMSE	(24.2)	data	No	No	Via MSE	Same, in the units of the data
MAE	(24.3)	data	No	Partly	Yes	-
MedAE	(24.3)	data	No	Yes	No	Blind to the tail, even when the tail is what matters
ME	(24.4)	data	No	No	No	Cancels; a zero value does not mean small errors
MAPE	(24.6)	%	Yes	No	No	Diverges as $y[n] \to 0$; asymmetric
sMAPE	(24.6)	%	Yes	No	No	Saturates at 200% as $y[n] \to 0$, so the bound hides the failure
MASE	(24.7)	none	Yes	Partly	No	Baseline is easy to beat on strongly correlated records
NRMSE_σ	(24.8)	none	Yes	No	No	Verdict moves with the spread of the evaluated segment
$\text{NRMSE}_{\text{range}}$	(24.8)	none	Yes	No	No	Range is set by two samples, so one outlier rescales it
AIC, BIC	(24.9)	none	n/a	No	No	Compare models, not predictions; same data only

Part III.

Signal Classification

25. Signal Feature-Extraction Pipeline

Goal: Full FE-based signal classification pipeline.

Contents

25.1. Preface **388**
25.2. Workflow **388**
25.3. Signal transformation (optional) **390**
25.4. Windowing (optional) **390**
25.5. Train–test split in signals **392**
25.5.1. Classification 392
25.5.2. Prediction 392
25.6. Feature Extraction **393**
25.6.1. Motivation 393
25.6.2. Common Features 394
25.6.3. Interpretation 396
25.6.4. Dedicated libraries 397
25.7. Learned embeddings (representation learning) **398**
25.8. Summary **398**

25.1. Preface

Classical ML operates on fixed-length feature vectors, but signals are long and often of variable length. The feature-extraction (FE) pipeline bridges this gap by reducing each signal (or each L-sample window, when the signal is long or continuous) to a single N-dimensional feature vector ($N \ll L$), producing a tabular dataset that any regressor or classifier from earlier chapters can use. Here we focus on the surrounding pipeline choices.

25.2. Workflow

Goal: Map the end-to-end pipeline that turns a raw signal into an $M \times N$ feature matrix.

FE from signals is summarized in Fig. 25.1. Starting from a raw signal, the pipeline proceeds through the following stages, in the order shown. The stages fall into three groups:

Signal preprocessing (optional).

1. *Signal transformation* (Sec. 25.3): a point-wise mapping reshapes each signal's distribution (log, Box–Cox, etc.).
2. *Windowing* (Sec. 25.4): long or continuous signals are split into M windows of L samples each (non-overlapping or overlapping).

Dataset construction.

3. *Train/test split* (Sec. 25.5): signals (or windows) are partitioned into training and test sets *before* feature extraction, so that the leakage-sensitive downstream steps (feature selection, normalization) never see test data. Feature extraction itself is per-signal and does not leak; the split is placed here only to set up those later steps.
4. *Feature extraction* (Sec. 25.6): each window (or whole signal) is mapped to an N-dimensional feature vector via $f : \mathbb{R}^L \to \mathbb{R}^N$. Stacking the M feature vectors row-wise yields the $M \times N$ tabular dataset.

Downstream ML (covered in earlier chapters of the book).

5. *Feature selection* (Sec. 9.4): retain the subset of the N features most informative for the task.
6. *Normalization* (Sec. 4.7): rescale the retained features (e.g. standardization) with parameters fit on the training set only.
7. *ML model*: fit a regressor or classifier on the normalized training features and evaluate on the held-out test set, producing the final outputs (predictions, class labels, performance metrics).

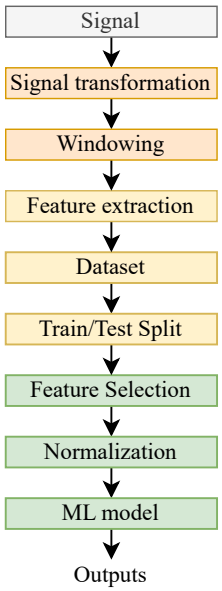


Figure 25.1.: Feature extraction from signals. The resulting dataset is used for downstream ML tasks such as regression or classification.

Some classifiers to be introduced in Ch. 26 already have several of these pipeline stages built in (e.g. internal feature selection, normalization, or end-to-end feature learning). When such a classifier is used, the corresponding stand-alone stages may be redundant and can be skipped.

25.3. Signal transformation (optional)

Goal: Apply a point-wise functional mapping that reshapes a signal's distribution to improve downstream prediction or classification, while preserving invertibility when the transform is applied to a regression target.

A signal transformation can produce either:

- a univariate signal from a univariate input (basic case);
- a multivariate set of signals from a univariate input, for further multivariate processing (advanced case).

Notes:

- When the transformation is applied to a regression *target* (univariate to univariate), it must be invertible so predictions can be mapped back to the original scale. For feature-generating transforms (univariate to multivariate) and for classification, invertibility is not required.
- Some transformations are restricted to positive signals, $y[n] > 0 \, \forall n$.

Common transformations:

- Logarithmic transformation,

$$\tilde{y}[n] = \log\big(y[n] + 1\big), \qquad y[n] \geq 0. \tag{25.1}$$

  The $+1$ shift avoids $\log 0$; the transform requires non-negative input.
- Square root transformation,

$$\tilde{y}[n] = \sqrt{y[n]} \tag{25.2}$$

- Exponential transformation,

$$\tilde{y}[n] = y^m[n] \tag{25.3}$$

Example 25.1: A sensor signal $y[n]$ with values in [0, 10,000] is highly right-skewed (Fig. 25.2). Applying $\tilde{y}[n] = \log\big(y[n] + 1\big)$ compresses the dynamic range to $[0, \approx 9.2]$, making the distribution more symmetric and improving downstream classifier performance.

In Python: `sklearn.preprocessing.PowerTransformer` (Box–Cox, Yeo–Johnson).

25.4. Windowing (optional)

Goal: When the input is a long or continuous signal, segment it into fixed-length windows; choose the window length L and overlap without leaking information between train and test.

This stage is skipped when each input signal is already a short, fixed-length record. It is required when the input is a long recording or a continuous stream that must be segmented before FE.

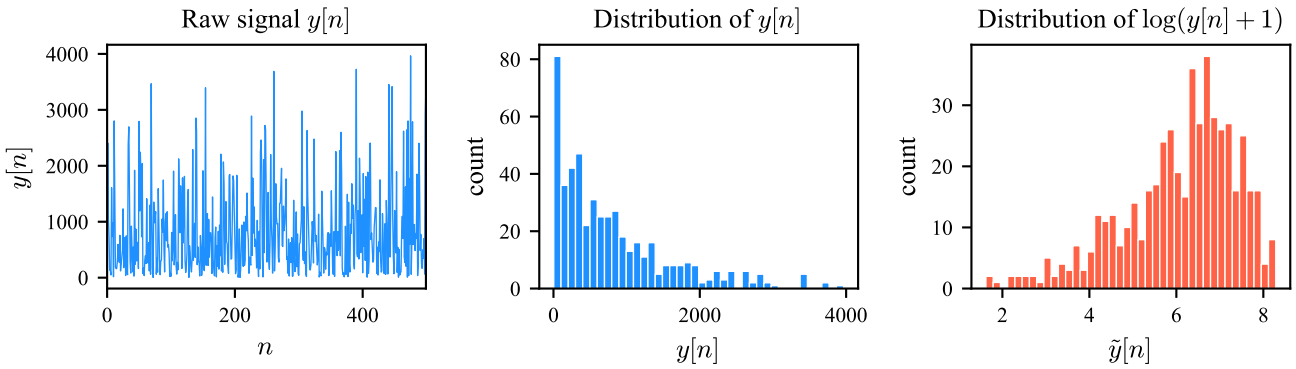
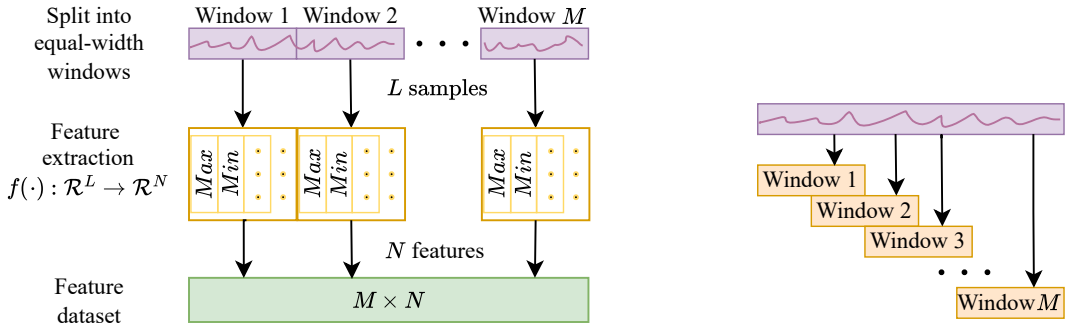


Figure 25.2.: Log transformation of a right-skewed signal.



(a) Non-overlapping windows

(b) Overlapping windows

Figure 25.3.: Split signal into windows.

Non-overlapping windows (Fig. 25.3a) are the standard choice. Overlapping windows (Fig. 25.3b) are less recommended since they may reduce performance: the resulting feature vectors become too similar.

For overlapping windows, the overlap is controlled by the step size S (number of samples between consecutive window starts). The overlap ratio is

$$r = \frac{L - S}{L}, \tag{25.4}$$

where $r = 0$ corresponds to non-overlapping windows and $r = 0.5$ to 50% overlap.

Feature redundancy

Overlapping windows produce highly correlated feature vectors: adjacent windows share most of their samples, so each new vector adds little information. The same overlap also opens a leakage path at the train/test split (Sec. 25.5).

Each window contains an equal number of samples, L. The value of L is:

- Field-related, e.g. 20–40 msec in acoustic and speech processing.
- Hand-picked by visual analysis.
- Hyper-parameter (least preferred: no prior, pure search).

25.5. Train–test split in signals

Goal: Partition a signal dataset so that evaluation estimates generalisation, not memorisation: respect source identity for classification and temporal order for prediction.

Possible data leakage

Random train-test splitting, which is standard for i.i.d. tabular data, is invalid for time-series signals. Adjacent windows share temporal context (and samples, if overlapping), so random splitting causes data leakage and over-optimistic performance estimates. Train and test windows must never be adjacent.

25.5.1. Classification

When the goal is to classify signals (e.g. fault detection, speaker identification), the split must be done by source rather than by individual windows. Each source (e.g. speaker, device, or recording session) appears entirely in either the train or the test set, never both.

This is known as *grouped CV* (Sec. 4.5). It ensures the model is evaluated on truly unseen sources, not on different windows from a source it has already learned.

Example 25.2: In a speaker identification task with 20 speakers, group splitting assigns 16 speakers to training and 4 to testing. All windows from a given speaker belong to the same set. A random window-level split would leak speaker-specific patterns into training, inflating accuracy.

25.5.2. Prediction

A temporal split is used: all data before a cutoff time t_c is used for training, and data after t_c for testing. The test set must always lie in the future relative to the training set.

Time-series cross-validation Standard k-fold CV shuffles the data, violating temporal order. Instead, time-series CV (Fig. 25.4) uses an expanding (or sliding) window:

1. Start with a minimal training set of the earliest samples.
2. Train the model and evaluate on the next time step (or window).
3. Expand the training set to include the previous test point, and repeat.

Each fold preserves the temporal ordering: training data always precedes test data (reference; example in Python).

Optionally, a gap of g samples can be inserted between the training and test sets. This prevents leakage from auto-correlated signals where adjacent windows carry similar information. The gap size g is a hyper-parameter that depends on the correlation length of the signal.

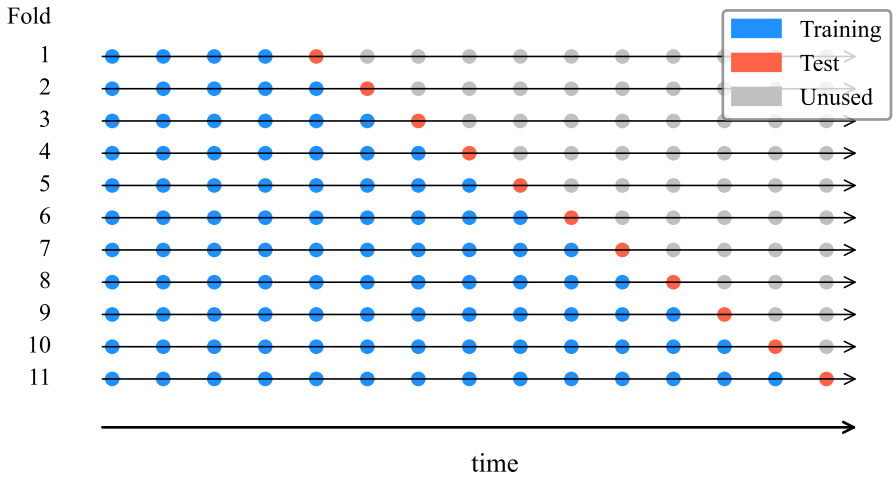


Figure 25.4.: Time-series cross-validation with expanding window. Each row is one fold: training samples (blue) always precede the test sample (orange); future samples (gray) are unused (inspired by source).

25.6. Feature Extraction

Goal: Summarize a raw signal by a small set of informative numbers, so a standard ML classifier can be applied without overfitting on the raw samples.

25.6.1. Motivation

An alternative to operating on raw samples is to first *describe* each signal by a handful of interpretable quantities (its mean, its dominant frequency, its peak rate) and then hand the resulting vector to any tabular classifier (logistic regression, SVM, random forest). This decouples the signal-processing step from the learning step and, by collapsing L raw samples down to $N \ll L$ features, directly mitigates the overfitting issue.

Feature: A feature is a numerical function of the signal that captures a characteristic of interest. Formally, feature extraction is a mapping

$$f : \mathbb{R}^L \to \mathbb{R}^N, \tag{25.5}$$

that converts a signal of L samples into a single scalar ($N = 1$) or a vector of N values.

Dimensions

- Signal length L: from hundreds to tens of thousands of samples.
- Feature count N: from a few to thousands.

The strong reduction $L \to N$ is precisely what makes feature-based classification tractable on small labeled datasets.

25.6.2. Common Features

Features are commonly grouped by the domain they summarize: time, frequency, joint time–frequency, and field-specific. The remainder of this section walks through these families.

Statistical (time-domain) features

Descriptive statistics computed directly on the signal samples $y[1], \ldots, y[L]$:

- Mean:
$$\bar{y} = \frac{1}{L} \sum_{n=1}^{L} y[n] \tag{25.6}$$
- Variance:
$$s_y^2 = \frac{1}{L} \sum_{n=1}^{L} \Big(y[n] - \bar{y} \Big)^2 \tag{25.7}$$
- Root mean square (RMS):
$$y_{\text{rms}} = \sqrt{\frac{1}{L} \sum_{n=1}^{L} y^2[n]} \tag{25.8}$$
- Zero-crossing rate (ZCR):
$$\text{ZCR} = \frac{1}{L-1} \sum_{n=2}^{L} \mathbf{1}\Big[y[n] \cdot y[n-1] < 0 \Big] \tag{25.9}$$
where $\mathbf{1}[\cdot]$ is the indicator function.
- Additional: maximum, minimum, median, skewness, kurtosis, energy ($\sum y^2[n]$), number of peaks.

Spectral (frequency-domain) features

Features derived from the magnitude spectrum $|Y[k]|$ obtained via the Fourier transform of the signal. They summarize *where* energy sits in frequency rather than *when* it occurs in time.

- Spectral centroid, the "center of mass" of the spectrum:
$$\text{SC} = \frac{\sum_{k=1}^{K} f_k \, |Y[k]|^2}{\sum_{k=1}^{K} |Y[k]|^2}, \tag{25.10}$$
where f_k is the frequency of the k-th bin. A high SC indicates a signal dominated by high-frequency content.
- Spectral bandwidth, spectral rolloff, spectral flatness, and the energy in pre-defined frequency bands.

Auto-correlation function based features

Quantities derived from the ACF of the signal. They capture periodicity and temporal structure (note the relation between ACF and PSD (Sec. 21.2)).

Model-based (parametric) features

Coefficients of a parametric time-series model fitted to the signal. Fitting an AR(p) model (Sec. 21.3), or more general ARMA model (Sec. 21.7), yields a compact set of coefficients that summarizes the signal's spectral shape.

The AR (linear-prediction) coefficients $h_1, \ldots, h_p$ (LPC) are obtainable from the ACF via the Yule-Walker equations (Sec. 21.3.1) and form a p-dimensional feature vector; the residual (prediction-error) variance is a useful scalar companion feature. LPC are a standard features in speech and audio processing.

Mixed-domain features

Features built on joint time–frequency representations, such as the short-time Fourier transform (STFT, Sec. 20.7) or the wavelet transform. These are appropriate when the discriminative information is non-stationary, i.e. when the spectral content changes over the duration of the signal.

A more elaborate example is the *wavelet scattering transform*, a translation-invariant, deep-network-like representation built by cascading wavelet transforms with modulus (and averaging) operations. It yields stable features without any learning and is well suited to small datasets; a ready-made implementation is given in the libraries below.

Field-tailored features

Quantities developed within a specific application domain to encode domain knowledge. A canonical example is the Mel-frequency cepstral coefficients (MFCC) used in speech and audio processing. A close relative is the gammatone cepstral coefficients (GTCC), which replace the mel filterbank with a gammatone filterbank that models the cochlea; GTCC are sometimes more robust than MFCC in the presence of noise.

Inherent windowing

Some of the FE methods, such as STFT or MFCC, have inherent windowing.

For ready-made implementations of dozens of features in each of the families above, see, for example, the tsfel feature list.

Computational cost Some features, especially spectral or wavelet-based ones, are non-trivial to compute for large L. Two practical remedies:

- *Parallel computation*: signals are processed independently, so the per-signal feature pipeline parallelizes trivially across CPU or GPU cores.
- *Feature selection* (Ch. 9): keep only a compact, discriminative subset so the heaviest features are not computed unless they pay for themselves.

General Note: Feature Normalization

Features from different domains may have very different scales (a centroid in Hertz vs. a unit-less skewness). Apply feature normalization before stacking them into a single vector, otherwise distance- and gradient-based classifiers will be dominated by the largest-magnitude feature.

Output of the pipeline Each input signal of L samples is mapped to a single N-dimensional feature vector. Stacking the vectors of all training signals row-wise yields a tabular dataset of shape (M, N) on which any standard classifier from earlier chapters can be trained directly.

25.6.3. Interpretation

A single feature is rarely tied to one branch of the theory. To illustrate, take the root-mean-square (RMS) of a signal,

$$\mathrm{RMS}(\mathbf{y}) = \sqrt{\frac{1}{L}\sum_{n=1}^{L} y^2[n]} = \frac{1}{\sqrt{L}}\|\mathbf{y}\|, \tag{25.11}$$

and read it through five complementary approaches. Each approach points to neighbouring features in the catalogue above.

Statistical interpretation Under the common zero-mean assumption $\bar{y} \approx 0$, Eq. (25.11) reduces to the (biased) sample standard deviation of $\{y[1], \ldots, y[L]\}$ (Sec. 1.1). The natural extensions are higher-order sample moments such as kurtosis.

Distance interpretation Eq. (25.11) is, up to the factor $1/\sqrt{L}$, the Euclidean (L_2) distance from the sample vector $\mathbf{y} = (y[1], \ldots, y[L])$ to the origin. Replacing the L_2 norm by another L_p or Minkowski distance (Sec. 10.2) yields a family of analogous "size" features, of which RMS is the $p = 2$ representative.

Signal characterization Squaring RMS recovers the signal power $P_{\mathbf{y}} = \frac{1}{L}\|\mathbf{y}\|^2 = \mathrm{RMS}(\mathbf{y})^2$, and the corresponding energy is $E_{\mathbf{y}} = L \cdot P_{\mathbf{y}}$ (Sec. 20.1.2). RMS is therefore the simplest amplitude-based characterization of the signal.

Spectral interpretation By Parseval's theorem (Eq. (20.69)), the same energy can be read off the magnitude spectrum $|Y[k]|$, so RMS is the zeroth-order summary of the spectrum. Higher-order summaries of the same spectrum, the spectral centroid, bandwidth, rolloff, and band energies sit naturally beside it.

ACF interpretation The energy identity $E_{\mathbf{y}} = R_{\mathbf{y}}[0]$ gives $\mathrm{RMS}(\mathbf{y}) = \sqrt{R_{\mathbf{y}}[0]/L}$, i.e. RMS is the value of the auto-correlation function at lag zero (Sec. 21.1). Other ACF-related features, such as the lag and height of the first non-trivial peak of $R_{\mathbf{y}}[k]$ at $k \neq 0$ extend the same family and capture periodicity rather than amplitude.

25.6.4. Dedicated libraries

Goal: Use ready-made toolboxes to compute large feature banks and run statistical feature selection without hand-coding each feature.

Several open-source libraries provide large collections of pre-implemented signal features, eliminating the need to code them from scratch.

`tsfresh` (**T**ime **S**eries **F**resh)

- Python-based, 150+ features covering statistical, spectral, and non-linear domains.
- Built-in statistical feature selection (relevance testing via hypothesis tests).
- Seamless `scikit-learn` integration (transformers, pipelines).

`tsfel` (**T**ime **S**eries **F**eature **E**xtraction **L**ibrary)

- Python-based, 60+ features organized by domain (statistical, temporal, spectral).

`hctsa` (**h**ighly **c**omparative **t**ime-**s**eries **a**nalysis)

- Matlab-based, 7,700+ features, the largest available feature library.
- Designed for exploratory comparison across many time-series datasets.

`catch22` (**CA**nonical **T**ime-series **CH**aracteristics)

- A curated subset of 22 features selected from 4,791 `hctsa` features on 93 publicly available datasets.
- Provides strong baseline performance on a broad variety of signals.
- Applied on internally normalized signals; optional mean and std features.
- Fast `C` implementation with Python, R, and Matlab wrappers.

`kymatio` Unlike the general-purpose feature banks above, this is a specialized library that computes a single family of features, the wavelet scattering transform.

- Python-based, GPU-enabled (`PyTorch`, `TensorFlow`, `JAX` backends).
- Scattering transforms for 1D signals, 2D images, and 3D volumes.
- See the kymatio documentation.

`scikit-learn`

- Does not include signal-level feature extraction, but provides classifiers, regressors, and feature selection methods.
- Feature selection can be embedded in a `Pipeline` for end-to-end workflows.

Complementary: `mlxtend` supplies sequential feature selection (forward/backward) and model-evaluation utilities on top of `scikit-learn`.

25.7. Learned embeddings (representation learning)

Instead of manually designing signal features based on domain knowledge, representation learning uses deep neural networks to automatically learn informative representations (embeddings) directly from raw signals or time-frequency representations (e.g., spectrograms).

These learned features can be obtained through several paradigms:

- *Supervised pre-training*: A deep neural network (e.g., a 1D CNN, ResNet, or Transformer) is trained on a large labeled signal classification dataset. Once trained, the final classification layer is discarded, and the activations of the penultimate layer are extracted as a fixed-size feature vector (embedding) $\mathbf{z} \in \mathbb{R}^d$ for downstream tasks.
- *Self-supervised learning (SSL)*: Representations are learned from unlabeled data by solving a "pretext" task. Popular SSL approaches for signals include:
  - *Autoencoders (AE)*: An encoder network maps the signal to a low-dimensional bottleneck state, and a decoder network reconstructs the original signal. The bottleneck state serves as the learned embedding.
  - *Contrastive learning*: The network is trained to maximize similarity between different augmented views (e.g., adding noise, cropping, jittering) of the same signal window (positive pairs) while minimizing similarity with views from other signals (negative pairs). This is the 1-D counterpart of the Siamese setup of Sec. 19.1, with augmented windows in place of image pairs. Contrastive Predictive Coding (CPC) uses temporal forecasting as a pretext task to learn representations.
  - *Masked autoencoders (MAE)*: Portions of the signal are masked (zeroed out), and the network is trained to predict or reconstruct the missing segments. This is the image recipe of Sec. 19.2 with patches replaced by time segments, including the rule that the loss is evaluated on the masked positions only.

Pros and Cons

- **Advantages**: Reduces the need for hand-crafted feature engineering, captures complex non-linear patterns, and scales well with large datasets.
- **Disadvantages**: Requires large volumes of data to learn generalizable representations, lacks direct physical interpretability (black-box features), and has high training and inference computational costs compared to basic statistical features.

25.8. Summary

- Pipeline order (Fig. 25.1): (optional) signal transformation → (optional) windowing → train/test split → feature extraction, producing an $M \times N$ tabular dataset via $f : \mathbb{R}^L \to \mathbb{R}^N$.
- Signal transformations (log, Box–Cox, etc.) reshape distributions to improve downstream performance; they must be invertible when applied to a regression target.
- Windowing applies only to long or continuous signals; short fixed-length signals skip directly to the split. When used, non-overlapping windows are preferred and the overlap ratio $r = (L - S)/L$ controls redundancy and leakage risk.
- Train–test splitting happens *before* feature extraction and must respect structure: group-split for classification, temporal/expanding-window CV for prediction, never random.

26. Signal Classifiers

Goal: Assign a whole signal $y[n]$ to one of C discrete classes.

Contents

26.1. Preface **399**
26.2. Distance Metrics **400**
26.2.1. Dynamic Time Warping (DTW) 400
26.2.2. Shapelets 402
26.3. Pseudo-Random-Features-Based Classifiers **404**
26.3.1. Time-series forest 404
26.3.2. Random Interval Spectral Ensemble (RISE) 405
26.4. Symbolic aggregate approximation (SAX) **405**
26.5. Random Convolutions **406**
26.5.1. ROCKET 406
26.5.2. MINIROCKET 407
26.6. Ensemble Classifiers **407**
26.6.1. HIVE-COTE 407
26.7. Summary **407**

26.1. Preface

Given a training set of labeled signals, the task is to assign a whole signal $y[n]$, $n = 0, \ldots, L-1$ to one of C discrete classes. Unlike the regression setting of earlier chapters, the input is a sequence of inter-related samples rather than a fixed-length feature vector.

Dataset Signals in a dataset may be

- univariate (single channel $y[n]$) or multivariate (K channels $y_k[n]$, e.g. ECG leads or accelerometer axes);
- equal-length or variable-length (length L may differ between examples);
- scarce: labeled time-series datasets are typically small relative to L.

Overfitting challenge Using each sample $y[n]$ as a feature yields an L-dimensional input from only a handful of training examples, so a raw-sample classifier overfits easily. The methods below differ mainly in how they summarize, transform, or compare signals before classification in order to control this.

Main categories of methods

- *Distance-based*: define an elastic distance between raw signals and apply k-NN (Dynamic Time Warping).
- *Feature-based*: extract interpretable summaries and classify on them (time-series forest / RISE, shapelets, SAX and SAX-VSM). The general feature-extraction pipeline these methods specialize is covered in Sec. 25.6.
- *Specially adapted transforms*: apply a large bank of random or fixed transforms followed by a linear classifier (ROCKET, MINIROCKET), or combine several domain-specific classifiers in a meta-ensemble (HIVE-COTE).
- *Deep learning* (out of scope here, pointer only): 1D CNNs, RNNs, and pretrained time-series foundation models. These are contrasted with classical methods in the closing table.

26.2. Distance Metrics

The standard distances are merely applicable

The standard distance metrics of Sec. 10.2 treat a signal as a plain vector in $\mathbb{R}^L$ and pair samples index-by-index. For signals this is rarely what we want, for two reasons:

- *Sample-to-sample comparison*: Distances, such as L_p and cosine, are sensitive to any misalignment along the time axis, such as a shift, a stretch, a local speed change, or an additional/missing event, even when the two signals share the same underlying characteristics. They are also restricted to equal-length inputs.
- *Curse of dimensionality*: treating each of the L samples as a separate feature places the classifier in the ultra-high-dimensional regime (Sec. 10.4).

The distance below are particularly fit to signals of the certain kinds.

26.2.1. Dynamic Time Warping (DTW)

Goal: "Elastic" distance metric between two signals.

- Time-domain, non-linear metric that can stretch and compress the time axis to find the best possible match between sequences.
- Applied for k-NN and other classifiers.
- Historically, developed to compare same/different words spoken with different speed.

Definition

Given two signals [9]

$$\begin{aligned} X &= (x_1, \ldots, x_m, \ldots, x_M)\,, \\ Y &= (y_1, \ldots, y_n, \ldots, y_N)\,, \end{aligned} \tag{26.1}$$

the DTW algorithm takes as input a pointwise dissimilarity function

$$d_{mn} = f(x_m, y_n) \geq 0. \tag{26.2}$$

A common choice is the 1D Euclidean distance, $d_{mn} = |x_m - y_n|$.

A *warping path* is a sequence of index pairs $\pi = \{(m_k, n_k)\}_{k=1}^{K}$ that satisfies three constraints:

- *Boundary*: $(m_1, n_1) = (1, 1)$ and $(m_K, n_K) = (M, N)$.
- *Monotonicity*: $m_{k+1} \geq m_k$ and $n_{k+1} \geq n_k$ (no turns back).
- *Step size*: each step is vertical $(m{+}1, n)$, horizontal $(m, n{+}1)$, or diagonal $(m{+}1, n{+}1)$ (no skips).

The DTW distance is the minimum cumulative cost along any admissible path,

$$\mathrm{DTW}(X, Y) = \min_{\pi} \sum_{k=1}^{K} d_{m_k n_k}, \tag{26.3}$$

and is obtained by dynamic programming over the cost matrix $[d_{mn}]$ in $O(MN)$ time. The resulting path serves as an elastic distance between the two signals, allowing non-linear stretching of the time axis.

Unlike Euclidean distance, which pairs samples index-by-index and therefore penalizes any time shift or stretch, DTW is free to re-align the time axes and then measures the residual mismatch under the best alignment. The distance is therefore small whenever X and Y share the same *shape*, even if one is faster, slower, or shifted in time.

- DTW is a *dissimilarity measure*, not a true metric; the triangle inequality does not hold in general.
- The raw sum depends on path length. To compare pairs with different M, N, it is common to normalize by the number of steps K (mean cost per step).

Example 26.1: Two signals of different lengths: X is a chirp of $M = 1000$ samples and Y is a sinusoid of $N = 399$ samples (Fig. 26.1). The cost matrix $d_{mn} = |x_m - y_n|$ and the optimal warping path are shown in Fig. 26.2. After alignment along the optimal path, the two signals match closely (Fig. 26.3), demonstrating how DTW stretches the time axis to find the best correspondence.

Multivariate DTW

The multivariate case is identical to the univariate one, except that the pointwise dissimilarity is defined between vectors rather than scalars. For K-dimensional (possibly complex-valued) samples $\mathbf{x}_m, \mathbf{y}_n \in \mathbb{C}^K$, a natural choice is the Euclidean norm of the difference,

$$d_{mn} = \|\mathbf{x}_m - \mathbf{y}_n\|_2 = \sqrt{\sum_{k=1}^{K} \left(x_{k,m} - y_{k,n}\right) \left(x_{k,m} - y_{k,n}\right)^*}. \tag{26.4}$$

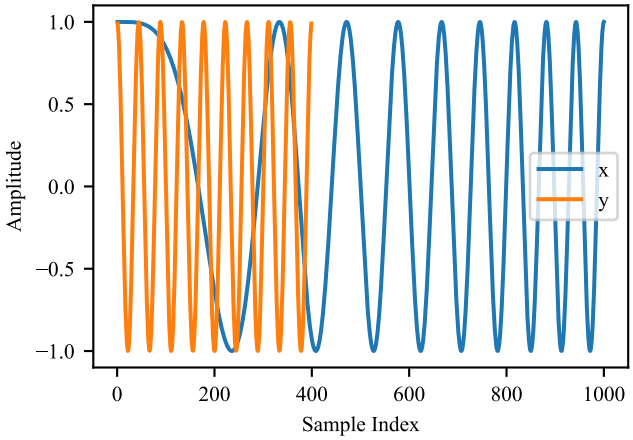


Figure 26.1.: Signal examples.

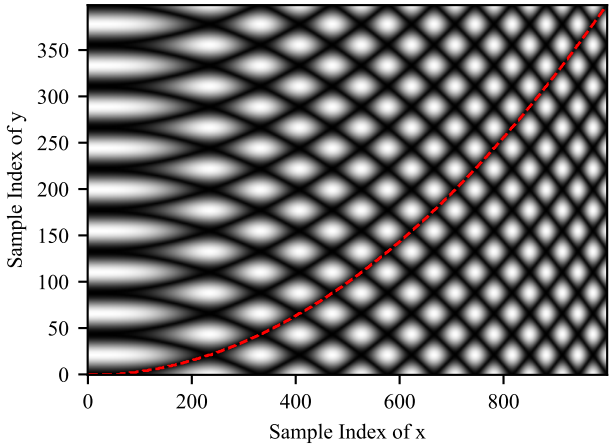


Figure 26.2.: d_{mn} with the optimal path.

k-NN with DTW

Because DTW provides a pairwise distance between signals of possibly different lengths, it plugs directly into any distance-based classifier. The canonical choice is k-NN (Section 10): to classify a new signal, compute its DTW distance to every training signal and take a majority vote among the k nearest. With $k = 1$, 1-NN + DTW is a strong and frequently cited baseline for time-series classification.

26.2.2. Shapelets

Goal: Classify time series by identifying short, discriminative sub-sequences (shapelets) that characterize each class.

A shapelet is a sub-sequence $S = (s_1, \ldots, s_l)$ of length l that is "maximally representative of a class" [18].

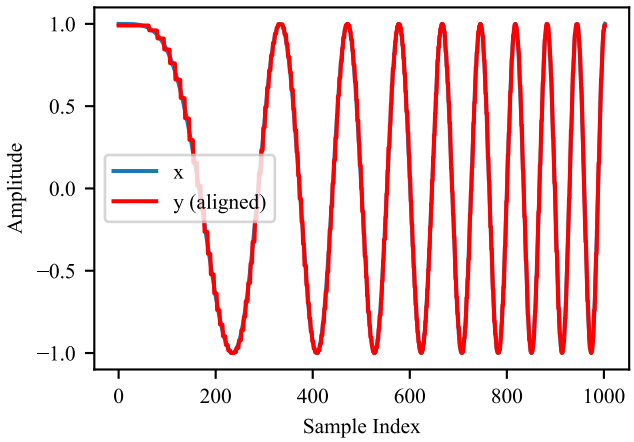


Figure 26.3.: Aligned signals.

Shapelet distance

Given a shapelet S of length l and a signal $X = (x_1, \ldots, x_M)$, the shapelet distance is the minimum distance over all possible alignments:

$$d(S, X) = \min_{i=1,\ldots,M-l+1} \sqrt{\frac{1}{l} \sum_{j=1}^{l} (s_j - x_{i+j-1})^2} \tag{26.5}$$

This sliding-window RMSE measures how well the shapelet matches anywhere in the signal (Fig. 26.4).

Classification

1. Extract shapelet candidates from the training set (sub-sequences of varying lengths).
2. For each candidate, compute its distance to every training signal.
3. Select the shapelet(s) whose distances best separate the classes (e.g. by information gain).
4. Use the shapelet distances as features for a classifier (e.g. decision tree, SVM).

Properties

- A signal may contain more than one discriminative shapelet; multiple shapelets can be combined as a feature vector.
- Shapelets are interpretable — they correspond to physically meaningful signal patterns.
- Optimized for time-domain analysis; less effective for signals whose discriminative information lies in repeated patterns or the frequency domain.
- Shapelet discovery can be computationally expensive ($O(M^2 \cdot l)$ per candidate); approximate and learned methods exist to reduce cost.

In Python: `tslearn` (shapelet learning), `sktime` (shapelet transform classifier).

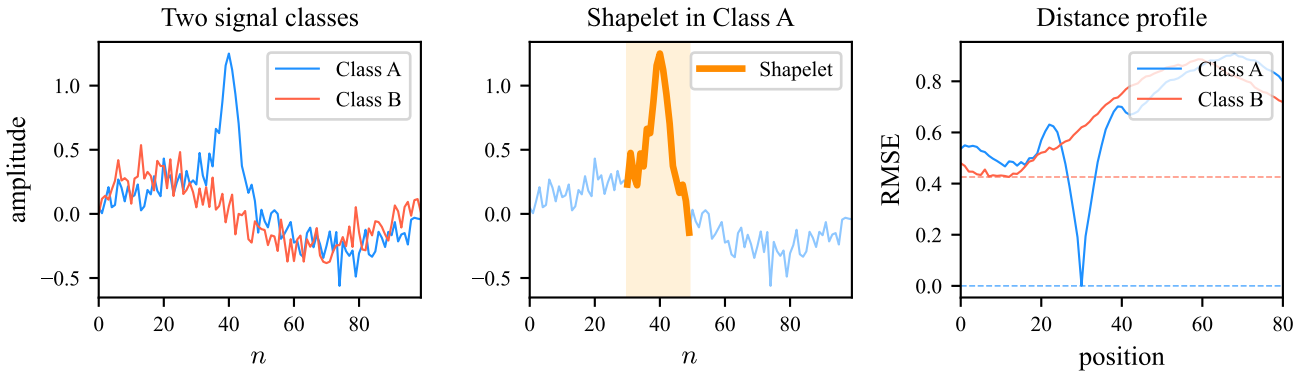


Figure 26.4.: Shapelet-based classification. Left: two signal classes (Class A contains a characteristic bump). Center: the shapelet (orange) extracted from Class A. Right: distance profile — Class A has a near-zero minimum distance, while Class B remains high, enabling discrimination.

26.3. Pseudo-Random-Features-Based Classifiers

26.3.1. Time-series forest

Goal: Classify time series using random forests built on summary features extracted from random intervals.

Time-series forest (TSF) is an interval-based method. Instead of operating on raw samples or shapelets, it extracts simple statistics from randomly chosen sub-intervals of the signal.

Algorithm

1. For each tree in the forest, randomly select $\sqrt{M}$ intervals $[a_i, b_i] \subset \{1, \ldots, M\}$.
2. From each interval, extract three summary features: mean, standard deviation, and slope.
3. Train a decision tree on the resulting feature vector (of length $3\sqrt{M}$).
4. Repeat for all trees and aggregate predictions by majority vote.

Properties

- Somewhat interpretable: each split corresponds to a statistic computed over a specific time interval.
- Computationally efficient compared to shapelet-based methods.
- Captures interval-level (phase) information rather than point-level or subsequence-level patterns.

26.3.2. Random Interval Spectral Ensemble (RISE)

RISE replaces the time-domain summary statistics with spectral features (e.g. periodogram coefficients), making it effective for signals whose discriminative information lies in the frequency domain.

In Python: `sktime` (interval-based classifiers including TSF and RISE).

26.4. Symbolic aggregate approximation (SAX)

Goal: Convert a real-valued time series into a discrete symbolic string, enabling the use of text-mining techniques for classification.

Algorithm

1. **Piecewise Aggregate Approximation (PAA):** divide the signal of length M into w equal-length segments and replace each segment by its mean, producing a reduced representation of length w.
2. **Symbolization:** map each mean value to a symbol from an alphabet of size α using breakpoints derived from the standard normal distribution (assuming z-normalized data). The result is a string of length w over the alphabet $\{a, b, \ldots\}$.

Example 26.2: A signal of $M = 128$ samples is converted to SAX with $w = 8$ segments and alphabet size $\alpha = 4$. The PAA reduces the signal to 8 mean values (Fig. 26.5, left). Each mean is mapped to a symbol based on the breakpoints of the standard normal distribution (center). The resulting SAX string is shown on the right.

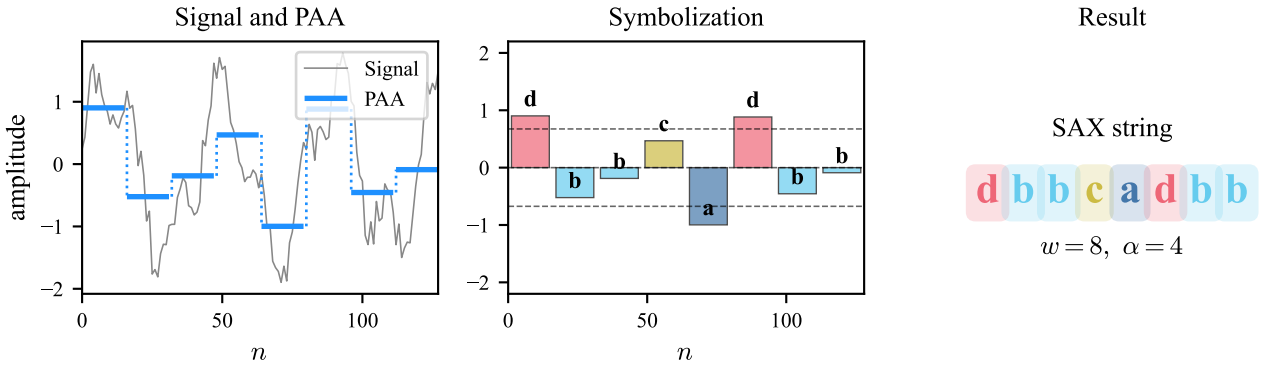


Figure 26.5.: SAX visualization: original signal with PAA overlay (left), symbolization with breakpoints (center), and resulting SAX string (right).

SAX-VSM

SAX combined with the Vector Space Model (SAX-VSM) [16] applies text-classification ideas to time-series classification:

1. Convert each training signal to a SAX string using a sliding window, producing a "bag of words."
2. Build a tf-idf weighted term-frequency vector for each class.
3. Classify a new signal by computing the cosine similarity of its bag-of-words vector to each class vector.

Properties

- Interpretable: discriminative patterns correspond to readable symbolic words.
- Dimensionality reduction is controlled by two parameters: number of segments w and alphabet size α.
- Sensitive to the choice of w and α; typically selected by cross-validation.

In Python: `tslearn` (PAA, SAX), `sktime` (SAX-based classifiers).

26.5. Random Convolutions

Goal: Classify time series by transforming them with a large number of random convolutional kernels, then training a linear classifier on the resulting features.

26.5.1. ROCKET

RandOm Convolutional KErnel Transform (ROCKET) [5].

Algorithm

1. Generate a large number (e.g. 10,000) of random convolutional kernels with random length, weights, bias, dilation, and padding.
2. Convolve each kernel with the input signal.
3. From each convolution output, extract two features: the maximum value and the proportion of positive values (ppv).
4. Train a linear classifier (e.g. ridge regression) on the resulting feature vector.

The key insight is that a sufficiently large collection of random kernels captures diverse patterns (trends, spikes, oscillations at various scales) without any learning or search over the kernel space.

Properties

- May be slow: training scales linearly with the number of signals and kernels.
- Accuracy comparable to deep learning and ensemble methods (e.g. HIVE-COTE).
- Not interpretable: random kernels have no direct physical meaning.

26.5.2. MINIROCKET

MINIROCKET [6] is a faster, near-deterministic variant. The main differences are summarized below.

	ROCKET	MINIROCKET
Kernel weights	Random (continuous)	Fixed $\{-1, 2\}$
Features	Max value + ppv	ppv only
Determinism	Stochastic	Near-deterministic
Speed	Fast	Up to 75× faster

In Python: `sktime` (`RocketClassifier`, `MiniRocketClassifier`).

26.6. Ensemble Classifiers

Goal: Achieve state-of-the-art accuracy by combining classifiers that operate on different signal representations.

26.6.1. HIVE-COTE

The Hierarchical Vote Collective of Transformation-based Ensembles (HIVE-COTE) [14] is a meta-ensemble that fuses the outputs of several diverse classifiers, each operating on a different signal domain:

- Time domain (e.g. shapelet transform, TSF).
- Frequency domain (e.g. RISE).
- Dictionary-based representation (e.g. SAX-based methods).
- Interval-based and whole-series distance-based classifiers.

Each component classifier produces a probability distribution over the classes. HIVE-COTE combines them using a weighted vote, where the weights reflect each component's estimated accuracy.

Properties

- Among the most accurate non-DL classifiers on standard benchmarks.
- Computationally expensive: trains multiple full classifiers.
- Modular: individual components can be replaced or upgraded independently.

26.7. Summary

Method comparison

The methods presented in this chapter are compared below. The "domain" column indicates the signal representation used by each method. Accuracy and speed are qualitative rankings on standard benchmarks.

Method	Domain	Accuracy	Speed	Interpretable
DTW + k-NN	Time (distance)	Moderate	Slow	No
Shapelets	Time (sub-sequence)	Moderate	Slow	Yes
TSF / RISE	Time / Frequency (interval)	Moderate	Fast	Partially
SAX-VSM	Symbolic	Moderate	Fast	Yes
ROCKET	Time (random convolution)	High	Fast	No
HIVE-COTE	Multi-domain (ensemble)	Highest	Very slow	No

Classical vs. DL approaches

	Classical (non-DL)	DL
Signal modeling	Not required	Not required
Multivariate signals	Less effective	Naturally supported
Interpretability	Limited (except shapelets, SAX)	Limited
Generalization	Low	High
Complexity	High vs. model-based	High
Feature engineering	Required (hand-crafted)	Learned automatically
Data requirements	Moderate	Large (orders of magnitude more)
Hyper-parameters	Few	Many; optimization required
Non-linear signals	Supported	Supported

27. Exploratory Analysis of Univariate Signal Samples

Goal: Screen a labeled set of signals with exploratory data analysis (EDA).

The signal-classification chapters so far assume that each input sample is a clean, equally-sampled univariate record of comparable length and energy. In practice, raw acquisitions may fail any of these assumptions. This chapter is a partial diagnostic checklist applied *before* signal classification.

Contents

27.1. Visual inspection for artifacts **409**
27.2. Detecting saturated samples **410**
27.3. Signal boundaries: onset and offset **411**
27.4. Sample-rate adequacy **412**
27.5. Energy and power across classes **414**
27.6. Effective signal length **415**
27.7. Fundamental frequency **416**
27.8. Data-integrity permutation tests **417**

27.1. Visual inspection for artifacts

Goal: Before running any quantitative diagnostic, look at a representative sample of recordings per class.

Every quantitative test below answers a pre-specified question. Visual inspection answers the question you did not think to ask. A few minutes spent plotting raw samples typically exposes problems that would otherwise survive into the model: a class with a constant DC offset, a sensor that dropped to zero for a fraction of a second, a mains-hum line at 50/60 Hz, isolated spikes from electrical interference, or stretches of missing samples filled with zeros or NaN.

Minimum protocol

- Plot at least 5 randomly drawn samples per class, with the same y-axis range across classes.
- For each class plot one sample's amplitude histogram and one sample's spectrogram (Sec. 20.7) or PSD (Sec. 20.4.2).
- Report the count of samples containing `NaN`, $\pm\infty$, or runs of identical-valued samples.

Common artifacts

- *DC offset*:Is it per-class or inter-class consistent?
- *Mains hum*: narrow spectral lines at 50 Hz (Europe) or 60 Hz (US) and their harmonics.
- *Saturation*: flat tops at the A/D rails, treated quantitatively in Sec. 27.2.
- *Spikes and glitches*: isolated single-sample excursions far outside the local envelope. They inflate P_i (Sec. 27.5) and bias high-order spectral features.
- *Dropouts*: stretches of repeated values (often exactly zero) where the sensor lost lock or the buffer was empty. Indistinguishable in features from genuine silence unless explicitly flagged.
- *Phase or polarity flip*: a class consistently inverted relative to the other.

Visual inspection does not scale, and that is fine

On a dataset of 10^5 samples no human inspects them all. Inspect a stratified sample, e.g. 20 per class, plus the longest, shortest, highest power, and lowest-power (Sec. 20.1.2) recording per class. The goal is not full coverage; it is to discover failure modes.

Re-run on any change

Re-run the inspection plots after any change to the acquisition pipeline. Every protocol change is an opportunity for a new artifact.

27.2. Detecting saturated samples

Goal: Identify samples whose amplitude was hard-limited by the A/D converter.

A saturated sample exceeds the converter's full-scale range $\pm A_{\max}$. The signature appears in three views:

- *Time domain*: flat-top runs of consecutive points pinned at $\pm A_{\max}$.
- *Amplitude histogram*: an isolated spike of mass at the rails, separated from the interior distribution.

Three per-sample diagnostics make this quantitative:

1. *Rail fraction*: $\rho_i = \frac{1}{N_i} \sum_{n=0}^{N_i-1} \mathbb{1}\left[\left|x_i[n]\right| \geq (1-\epsilon)\, A_{\max}\right]$ for a small tolerance ϵ (e.g. 10^{-3}).
2. *Longest rail run*: $\ell_i = \max\{k : x_i[n], \ldots, x_i[n+k-1]$ all satisfy $|\cdot| \geq (1-\epsilon)A_{\max}\}$.
3. *Histogram inspection*: visible spike of mass at $\pm A_{\max}$, separated from the interior distribution.

Fig. 27.1 contrasts a clean and a saturated sinusoid across the three views.

Handle explicitly

When saturation is detected, choose explicitly haw to handle it: discard the sample, re-acquire with adjusted gain, or record saturation status as an explicit feature.

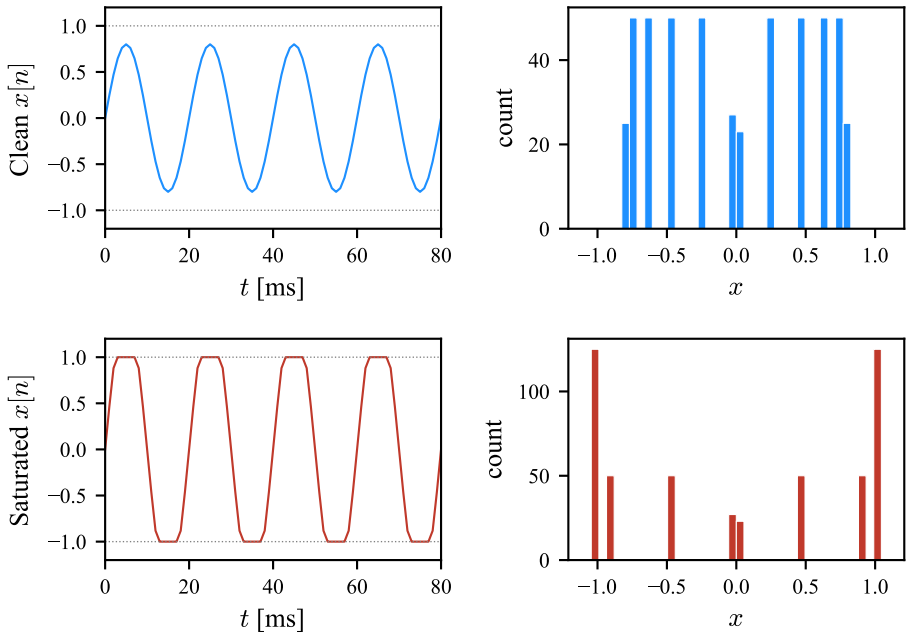


Figure 27.1.: Detecting saturation. Top: clean sinusoid. Bottom: the same sinusoid driven into A/D saturation. Left: time domain (flat-top runs visible). Right: amplitude histogram (rail spikes).

27.3. Signal boundaries: onset and offset

Goal: Locate the actual signal interval $[n_{\text{on}}, n_{\text{off}}]$ inside each sample and/or characterize its decay.

The nominal duration $T_i = N_i/f_s$ counts every recorded sample, including pre-event silence and post-event decay. The signal of interest sometimes occupies only a sub-interval $[n_{\text{on}}, n_{\text{off}}]$ with *effective duration*

$$T_i^{\text{eff}} = (n_{\text{off}} - n_{\text{on}})/f_s. \tag{27.1}$$

A small ratio T_i^{eff}/T_i is itself a quality metric: most of the recording is not signal.

Onset and offset

Given a per-sample log envelope $20\log_{10} e_i[n]$ (e.g. short-time RMSE over a window of a few fundamental periods; the factor 20 is the amplitude-to-dB convention, replace it with 10 for a squared/power envelope), locate the peak and define

$$n_{\text{on}} = \min\{n : 20\log_{10} e_i[n] \geq \text{peak}_{\text{dB}} - \theta\}, \quad n_{\text{off}} = \max\{n : 20\log_{10} e_i[n] \geq \text{peak}_{\text{dB}} - \theta\}, \tag{27.2}$$

with some θ threshold (measured in dB, .e.g. $\theta = 20$) and a hangover of L_h samples that keeps the segment still marked active to overcome an "occasional drop" in amplitude.

Noise floor θ in per-class setting may become class-discriminative feature.

Threshold may result leakage

Test set should not be used for thresholding derivation.

Exponential decay

When the signal has the form

$$x_i[n] \approx a\, e^{-(n-n_{\text{on}})/\tau} \cos\!\big(2\pi f_0 n / f_s + \phi\big), \tag{27.3}$$

the log envelope is linear in n between onset and the late noise floor. A direct linear fit of $\log e_i[n]$ on $[n_{\text{on}},\, n_{\text{off}}]$ returns $\hat{\tau}$. The per-class distribution of $\hat{\tau}$ is sometimes discriminative.

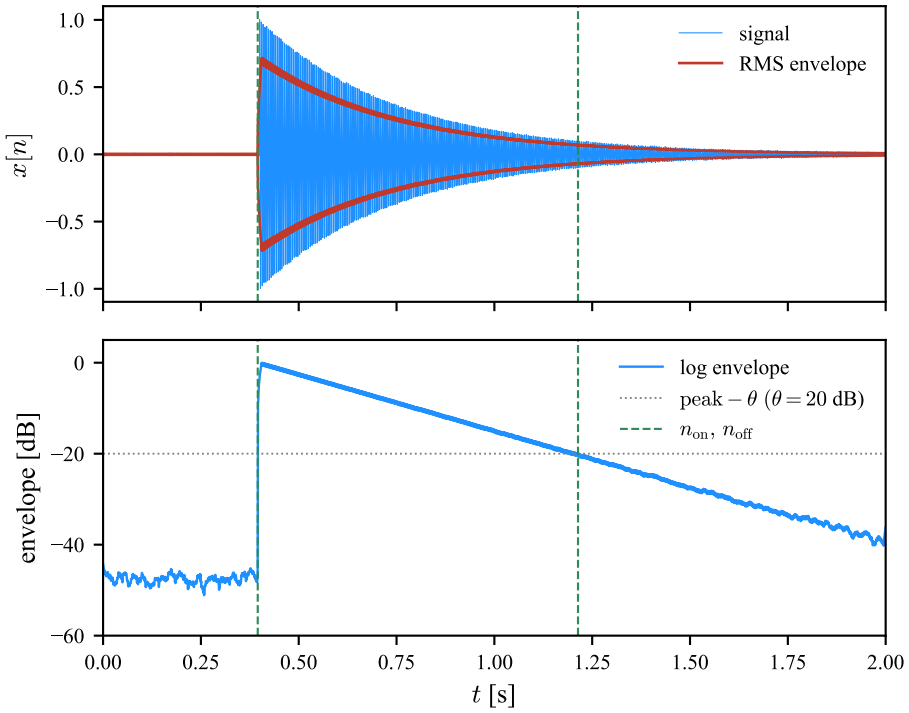


Figure 27.2.: Onset / offset detection on a synthetic damped sinusoid. Top: signal $x[n]$ with the RMS envelope (red) overlaid; vertical dashed lines mark the detected n_{on}, n_{off}. Bottom: log envelope in dB with the peak $-\,\theta$ threshold (dotted) and the same boundary lines.

For decay-dominated signals, $\hat{\tau}$ becomes a per-class feature in its own right.

Takeaway

Trim samples to active interval $[n_{\text{on}},\, n_{\text{off}}]$ before any downstream diagnostic.

27.4. Sample-rate adequacy

Goal: Confirm f_s is high enough to resolve the informative band, and check whether it is wastefully too high.

The Nyquist criterion $f_s > 2f_{\max}$ (see the signals chapter) is the lower bound. The classification literature obsesses over the under-sampling case; the over-sampling case is rarely discussed but just as common in practice.

The too-high case

When f_s is many times the informative bandwidth, every per-sample vector $\mathbf{x}_i$ is artificially long. The concrete costs:

- Feature-extraction computation costs.
- Some features have higher dimensions or are less informative.
- Curse of dimensionality on a representation whose intrinsic dimension is unchanged.

Spectral-occupancy diagnostic

Estimate the PSD (of a representative subset) of samples and locate the frequency f_{occ} above which a chosen fraction (e.g. 99%) of the in-band power lies. Set the working sample rate to $f_s' \gtrsim 2.5\, f_{\text{occ}}$ via decimation (with an anti-alias prefilter). Fig. 27.3 shows that PSDs at $2.5\, f_{\max}$ and $25\, f_{\max}$ contain the same informative band.

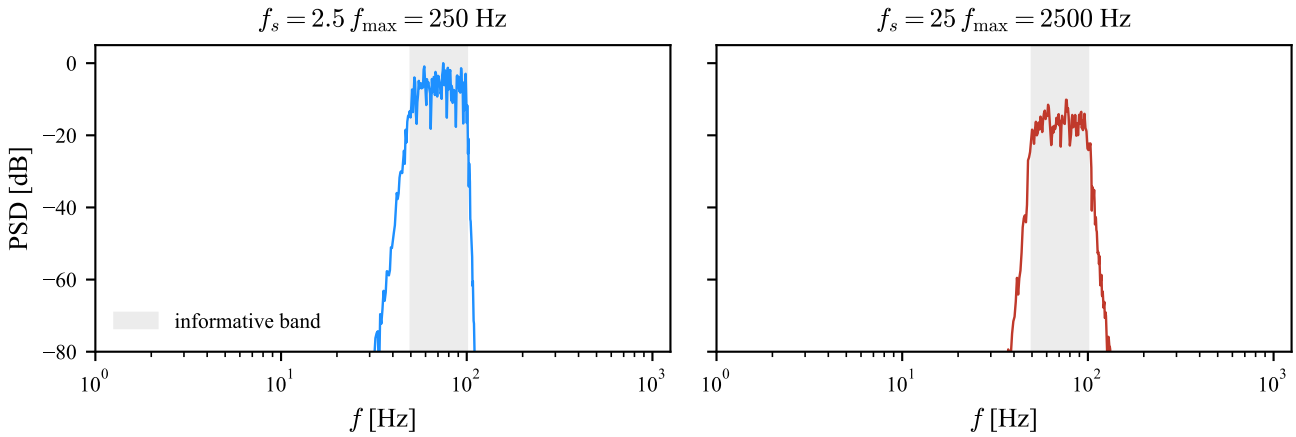


Figure 27.3.: Power spectrum of a narrowband signal sampled near Nyquist ($f_s = 2.5\, f_{\max}$) and oversampled by 10×. The informative band is identical; the extra Nyquist budget contains only noise.

Set f_s from spectral occupancy

The acquisition sample rate is usually a sensor default, not a modeling choice. Estimate the spectral occupancy of representative recordings and decimate to a working $f_s' \gtrsim 2.5\, f_{\text{occ}}$ before feature extraction.

Terminology: decimation vs. undersampling

Undersampling keeps every k-th sample as-is. *Decimation* first applies a low-pass anti-alias filter at the new Nyquist rate $f_s'/2$, then keeps every k-th sample. Use decimation. Prefer a linear-phase (FIR) anti-alias filter: a non-linear-phase one (e.g. IIR Butterworth) introduces frequency-dependent group delay that distorts onset positions and decay envelopes used downstream (Sec. 27.3).

Re-check with the classifier

The spectral-occupancy cut f_s' is necessary but not sufficient. After decimating the training set to f_s' (identical decimation applied to the test set, never with parameters fit on test), refit the chosen classifier and compare its held-out accuracy to a baseline fit at the original f_s. Equal accuracy confirms the discarded frequencies carried no class-discriminative information.

Sweep recipe

Decimate to a small grid of working rates (e.g. 1×, 0.5×, 0.25× of the original f_s) and plot held-out performance (e.g., accuracy) vs. f_s'.

Leakage warning

Decimation parameters (cutoff, filter order) are derived from training-set statistics only. This is the same end-to-end check pattern used in the permutation diagnostics of Sec. 27.8.

27.5. Energy and power across classes

Goal: Check whether per-sample amplitude alone discriminates the classes, and decide whether that discrimination reflects the phenomenon or the acquisition setup.

For each sample define

$$E_i = \sum_{n=0}^{N_i-1} x_i[n]^2, \qquad P_i = \frac{E_i}{N_i}. \tag{27.4}$$

When sample lengths N_i differ across the dataset, E_i is misleading and the average power P_i is the fair cross-class quantity. When the recordings include silence or decay tails, restrict the sum to the signal interval $[n_{\text{on}}, n_{\text{off}}]$ recovered in Sec. 27.3; otherwise silence padding deflates P_i and a class with a longer recorded tail looks quieter than it is.

A per-class boxplot of P_i (Fig. 27.4) answers two questions at once: is amplitude class-discriminative, and is the discrimination credible? Two distinct interpretations apply:

- *Phenomenon-driven*: classes genuinely differ in radiated power (e.g. loud vs. quiet sound source). Amplitude is a valid feature.
- *Acquisition-driven*: classes were collected on different hardware, gains, or distances. Amplitude is a confound and the classifier will exploit it instead of the phenomenon.

Disambiguation recipe

1. Compare P_i distributions before and after per-sample amplitude normalization (e.g. divide by $\sqrt{P_i}$ or scale to $\max|x_i| = 1$).
2. Refit the classifier on the normalized samples. If accuracy collapses to chance, amplitude was the only working signal; whether that is acceptable depends on the answer to the phenomenon-vs-acquisition question above.

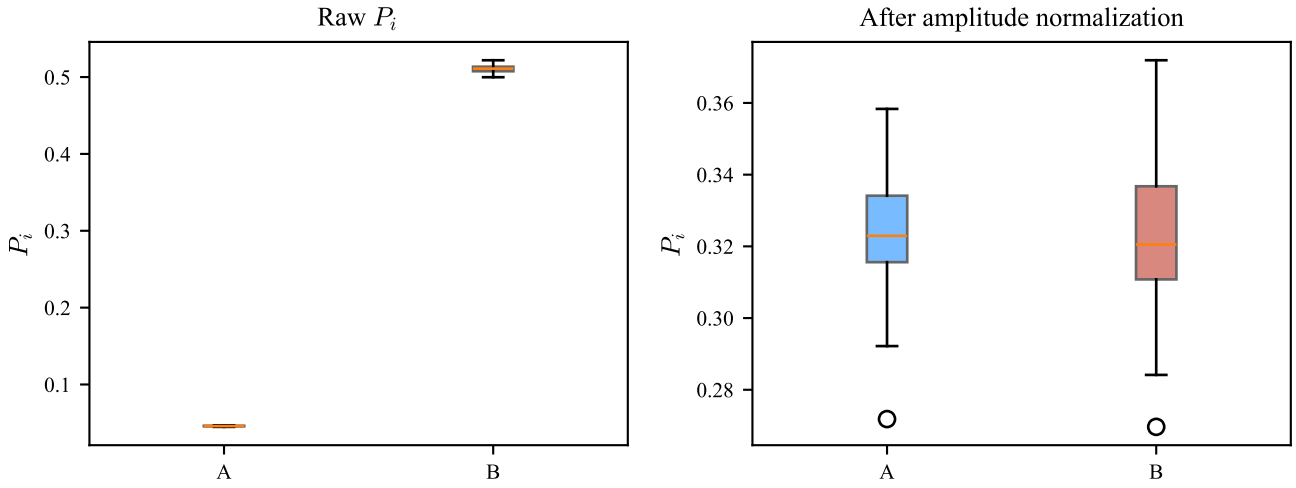


Figure 27.4.: Per-class power P_i before (left) and after (right) per-sample amplitude normalization. Class A and B are constructed to differ *only* in gain.

Interaction with saturation

A class whose samples are saturated while the other's are not has its P_i artificially pinned toward $A^2_{\max}$.

Acquisition-driven amplitude is leakage

If the amplitude difference between classes traces to a hardware or protocol difference at acquisition time rather than to the phenomenon, the classifier is learning the acquisition setup.
The leak is invisible to a held-out test set drawn from the same acquisition and is exposed only when the model is deployed on a new setup.
Normalize amplitude away or rerun acquisition under a single protocol.

27.6. Effective signal length

Goal: Match the recording duration $T_i = N_i/f_s$ to the time scale of the phenomenon being classified.

Two failure modes appear at the ends of the T_i scale.

Too short

A sample that contains only a fraction of a fundamental period cannot yield a reliable $\hat{f}_0$ (Sec. 27.7), and its spectral-resolution floor is $\Delta f = 1/T_i$. As a practical lower bound, target

$$T_i \cdot \hat{f}_0 \gtrsim 5, \tag{27.5}$$

i.e. at least five fundamental periods per sample. Plot the distribution of $T_i\hat{f}_0$ per class; an excess of samples below this threshold flags a recording protocol that needs to be lengthened.

Too long

A long recording is often a mixture of regimes (onset transient, steady state, decay) that should be classified separately. The diagnostic is intra-sample non-stationarity: split each $\mathbf{x}_i$ into Q contiguous sub-windows of equal length.

Significant spread within a single sample means either the sample should be re-windowed (Sec. 25.4) or the class label refers to a regime that occupies only part of the recording.

Spectral-resolution floor

No spectral feature can resolve detail finer than $\Delta f = 1/T_i$. A classifier asked to discriminate two classes whose spectra differ on a finer scale will fail regardless of the model; the fix is longer recordings, not a richer model.

Class-conditional duration histogram

A per-class histogram of T_i frequently reveals acquisition mismatch: one class consistently recorded longer than another (Fig. 27.5).

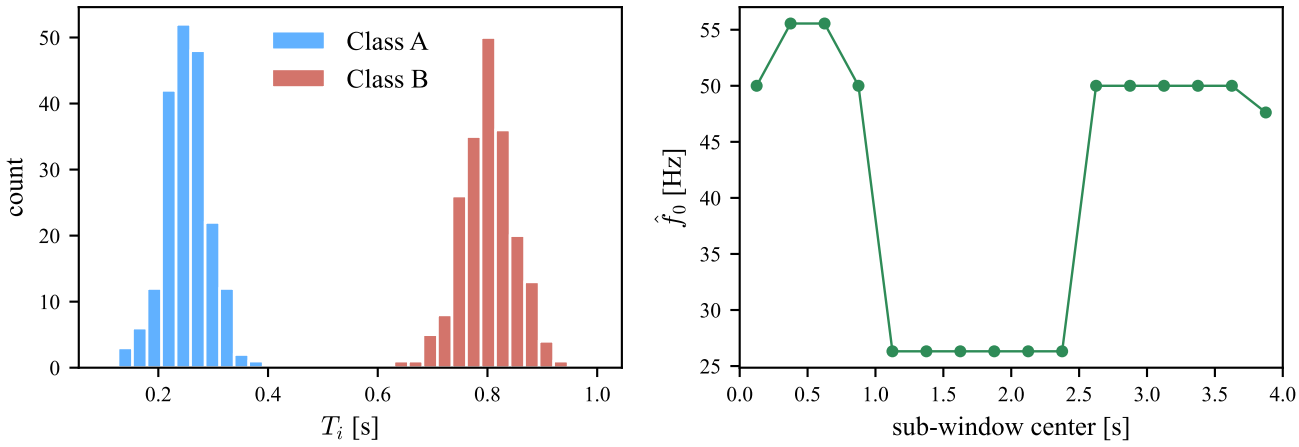


Figure 27.5.: Per-class histogram of T_i; class A is systematically shorter.

Acquisition-driven time-length is leakage

T_i itself can leak into duration-sensitive features.

27.7. Fundamental frequency

Goal: Estimate $\hat{f}_0$ per sample and per class, and check that its distribution agrees with the physical phenomenon.

For periodic or quasi-periodic signals, the fundamental frequency f_0 is often the single most informative scalar summary. The standard estimators are:

- *Autocorrelation peak* (Sec. 21.2.2): $\hat{f}_0 = f_s / \arg\max_{\tau \geq \tau_{\min}} r_{xx}[\tau]$, where r_{xx} is the biased autocorrelation and $\tau_{\min}$ excludes the trivial peak at $\tau = 0$.
- *Periodogram peak*: $\hat{f}_0 = \arg\max_f \left|X(f)\right|^2$, restricted to the expected band (Sec. 20.5).

A per-class histogram of $\hat{f}_0$ (Fig. 27.6) often reveals class separability via f_0 alone, and an empty overlap region is itself an important finding: a one-dimensional decision rule on $\hat{f}_0$ may already saturate accuracy and a complex classifier is unnecessary.

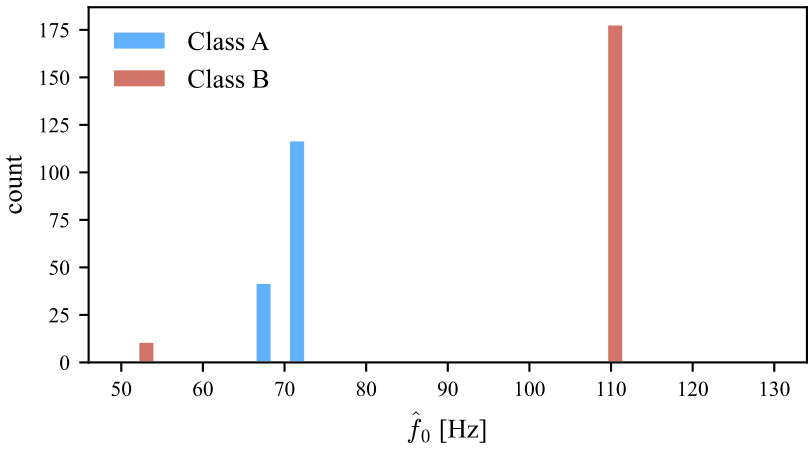


Figure 27.6.: Per-class histogram of $\hat{f}_0$ for two synthetic classes. Class A: narrowband around f_A. Class B: narrowband around f_B.

Interaction with saturation (Sec. 27.2). Saturation inflates higher harmonics but does *not* shift the fundamental, so $\hat{f}_0$ from autocorrelation or cepstrum remains usable for mildly saturated samples. A periodogram-based estimator restricted to the fundamental band is also robust; only an estimator that searches the full spectrum can be misled into reporting a harmonic.

f_0 ambiguity at short records

When the sample length T_i contains fewer than ≈ 3 fundamental periods, $\hat{f}_0$ from any of the three estimators has high variance and bias. This is a recording-length issue, not an estimator issue (Sec. 27.6).

27.8. Data-integrity permutation tests

Goal: Use targeted shuffles of labels and class membership to expose chance-level pipelines and per-sample leakage cues.

Two of the diagnostics in Ch. 13 are particularly relevant once a signal-classification pipeline is in place: the label-permutation null baseline (Sec. 13.3.1), which establishes the dataset- and pipeline-specific chance accuracy, and the cross-class sample permutation (Sec. 13.3.5), which exposes per-sample leakage cues (filename, timestamp, recording-session id, residual saturation signature, duration) that survive a swap between class buckets. Run both before trusting any held-out score.

28. Multivariate Signal Classification

Goal: Extend the univariate feature-extraction pipeline of the previous chapter to signals with $C > 1$ channels, where cross-channel structure carries information the per-channel pipeline discards.

Contents

28.1. Preface **419**

28.2. Channel-wise feature extraction **420**

28.2.1. Data fusion (early fusion) 420

28.2.2. Feature fusion (early fusion) 420

28.2.3. Late fusion 420

28.3. Cross-channel features **421**

28.3.1. Multivariate ACF 421

28.3.2. VAR coefficients 422

28.3.3. Coherence 422

28.3.4. Phase-locking value 422

28.3.5. Common spatial patterns (CSP) 423

28.3.6. Learned representations 423

28.4. Dimensionality reduction across channels **423**

28.5. Multivariate decomposition (*) **424**

28.5.1. Empirical mode decomposition (EMD) 424

28.5.2. Variational mode decomposition (VMD) 425

28.5.3. Dynamic mode decomposition (DMD) 426

28.5.4. Multichannel eigendecomposition (MSSA) 427

28.5.5. Multivariate variants and mode alignment 428

28.6. Cross-channel model transfer and linear alignment **429**

28.6.1. Setup: from vectors to channels 429

28.6.2. Why direct transfer fails 430

28.6.3. Linear alignment 431

28.6.4. Summary and practice 434

28.1. Preface

A multivariate signal is a collection of C synchronously sampled channels, organized as a matrix

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_C^\top \end{bmatrix} \in \mathbb{R}^{C \times L}, \qquad \mathbf{x}_c \in \mathbb{R}^L, \tag{28.1}$$

where row c is the L-sample record of channel c and column n is the instantaneous snapshot across all channels at sample n (Fig. 28.1).

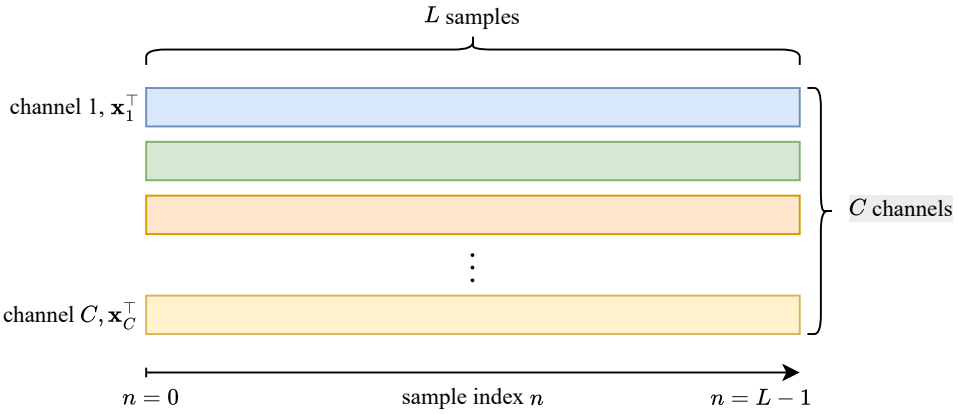


Figure 28.1.: A multivariate signal as a $C \times L$ matrix: C synchronously sampled channels along the rows and L time samples along the columns.

Typical sources:

- 3-axis IMU: accelerometer or gyroscope with $C = 3$ orthogonal axes.
- Multi-lead ECG ($C = 2, 3, 12$) and multi-channel EEG ($C = 8$–256).
- Microphone arrays for speech or acoustic event detection.

Applicability note

This chapter does not explicitly cover multi-rate sensor measurement, such as in multi-sensor industrial monitoring (vibration, temperature, current on the same machine).

Applying the univariate pipeline channel-by-channel is a valid baseline but discards the inter-channel structure, such as correlation, relative phase, shared latent sources, which often carry the discriminative information (e.g. an IMU gesture is defined by how the three axes move *together*, not by any one axis alone).

The pipeline reuses every stage from the previous chapter (Ch. 26) except feature extraction, which now produces two concatenated blocks:

- *Per-channel features* (Sec. 28.2): the univariate $f : \mathbb{R}^L \to \mathbb{R}^N$ applied independently to each of the C channels, yielding $C \cdot N$ features.
- *Cross-channel features* (Sec. 28.3): values that quantify inter-channel structure, yielding N_{cc} features.

Stacking gives a row of $N' = C \cdot N + N_{\text{cc}}$ features (per window).

28.2. Channel-wise feature extraction

Goal: Treat the C channels as C separate sources and combine them with the fusion strategies of Ch. 15.

The C channels are C synchronously sampled sources describing the same sample, so the fusion taxonomy of Ch. 15 applies directly, with each channel playing the role of a source. Three strategies result.

28.2.1. Data fusion (early fusion)

Stack the raw channel records into a single length-$C \cdot L$ vector and apply one univariate feature map $f : \mathbb{R}^{CL} \to \mathbb{R}^N$ (data concatenation, Sec. 15.1). This is the simplest option, but it loses channel identity and implicitly assumes the channels share a common representation (units, scale, sampling rate).

28.2.2. Feature fusion (early fusion)

Apply a univariate feature map $f : \mathbb{R}^L \to \mathbb{R}^N$ (time, frequency, or wavelet features as in Ch. 26) to each channel independently and concatenate the per-channel feature vectors (feature concatenation, Sec. 15.2):

$$\mathbf{z}_{\text{pc}}(\mathbf{X}) = \Big[f(\mathbf{x}_1)^\top,\ f(\mathbf{x}_2)^\top,\ \ldots,\ f(\mathbf{x}_C)^\top \Big]^\top \in \mathbb{R}^{CN}. \tag{28.2}$$

This is the trivial extension of the univariate pipeline and a reasonable baseline. Its weakness is that every inter-channel statistic is discarded: two datasets that differ only in how channels move *relative to each other* produce identical $\mathbf{z}_{\text{pc}}$.

Example 28.1: On a 3-axis accelerometer ($C = 3$) with $N = 10$ univariate features per channel, $\mathbf{z}_{\text{pc}} \in \mathbb{R}^{30}$. Walking and waving can produce near-identical per-axis statistics (mean, variance, dominant frequency) while differing sharply in axis-to-axis correlation, which this baseline cannot see.

28.2.3. Late fusion

Process each channel with its own feature-extraction and classification pipeline, then combine the C per-channel predictions with a fusion rule (classifier fusion, Sec. 15.3). This is attractive when the channels are heterogeneous or come from different modalities, since each pipeline can be specialized. Like feature fusion, however, it does not model the joint cross-channel structure, since each channel is classified in isolation. Capturing that structure is the role of the cross-channel features of Sec. 28.3.

Time synchronization

For an effective application of cross-channel features, all channels must share a common time base. Random jitter-misalignment can effectively destroys some of these features.

28.3. Cross-channel features

Goal: Summarize the inter-channel structure of $\mathbf{X}$ into a small vector of scalar features that complements the per-channel block.

28.3.1. Multivariate ACF

Viewing each channel as a signal, the auto-correlation (Sec. 21.1,Eq. (21.11)) and the cross-correlation (Sec. 22.1,Eq. (22.6)) merge into a single matrix-valued, lag-indexed object:

$$\begin{aligned} \mathbf{R}_{\mathbf{XX}}[\tau] &\in \mathbb{R}^{C\times C}, \\ \Big(\mathbf{R}_{\mathbf{XX}}[\tau]\Big)_{ij} &= R_{ij}[k] = \sum_n \Big(x_i[n]-\bar{x}_i\Big)\Big(x_j[n+\tau]-\bar{x}_j\Big). \end{aligned} \tag{28.3}$$

The following special cases recover the constructs of Part II:

- $i = j$: the auto-correlation $R_{\mathbf{xx}}[\tau]$ of channel i (Sec. 21.1).
- $i \neq j$: the cross-correlation between channels i and j (Sec. 22.1).
- $\tau = 0$, biased normalization (Eq. (21.14)): the covariance matrix used below.

The biased, unbiased, and normalized variants of Sec. 21.1.2 and Sec. 22.1 apply entry-wise to $\mathbf{R}_{\mathbf{XX}}[\tau]$.

Covariance and correlation

At $\tau = 0$, the biased form of the multivariate ACF (Sec. 28.3.1, Eq. (21.14)) is the covariance matrix

$$\mathbf{\Sigma} = \frac{1}{L}\mathbf{R}_{\mathbf{XX}}[0] \in \mathbb{R}^{C\times C}, \qquad \mathbf{\Sigma}_{ij} = \mathrm{Cov}\big[\mathbf{x}_i, \mathbf{x}_j\big] = R_{ij,biased}[k], \tag{28.4}$$

or its normalized counterpart, the Pearson correlation matrix

$$\mathbf{R}_{ij} = \frac{\mathbf{\Sigma}_{ij}}{\sqrt{\mathbf{\Sigma}_{ii}\,\mathbf{\Sigma}_{jj}}} \in [-1, 1]. \tag{28.5}$$

$\mathbf{R}$ is symmetric with unit diagonal, so the $C(C-1)/2$ strictly upper-triangular entries are used as features (Fig. 28.2). For $C = 3$ this adds 3 features; for $C = 16$, 120.

Lagged cross-correlation

For $\tau \neq 0$, the off-diagonal entries of the multivariate ACF $\mathbf{R}_{\mathbf{XX}}[\tau]$ ((28.3)) encode delayed coupling between channels and the timing of that coupling. Writing $r_{ij}[\tau] \triangleq (\mathbf{R}_{\mathbf{XX}}[\tau])_{ij}$ for the (i,j)-entry, two scalar features per channel pair are typical:

- the peak value $\max\limits_{|\tau|\le\tau_{\max}} r_{ij}[\tau]$, $i \neq j$, and
- the argmax lag $\operatorname{argmax}\limits_{|\tau|\le\tau_{\max}} r_{ij}[\tau]$, $i \neq j$.

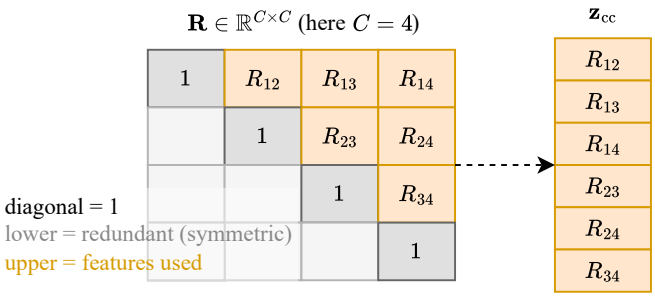


Figure 28.2.: Cross-channel features from the correlation matrix $\mathbf{R}$: diagonal entries are always 1 and the lower triangle is redundant by symmetry, so only the $C(C-1)/2$ upper-triangular entries (highlighted) are stacked into the cross-channel feature vector $\mathbf{z}_{\text{cc}}$.

28.3.2. VAR coefficients

Fitting a vector AR (VAR(p), Sec. 22.9) model to the C-channel window estimates the coefficient matrices $\mathbf{A}_1, \ldots, \mathbf{A}_p \in \mathbb{R}^{C\times C}$. Their off-diagonal entries $(\mathbf{A}_m)_{ij}$, $i \neq j$, quantify how the past of channel j predicts the present of channel i: a *directed*, lagged, conditional measure of coupling[1], in contrast to the symmetric, undirected correlation of Sec. 28.3.1. The $pC(C-1)$ off-diagonal entries across all lags are stacked as features; the pC diagonal entries are per-channel self-prediction and may be assigned to the per-channel block instead.

28.3.3. Coherence

For a pair of channels, the *cross-coherence* (Sec. 22.2.2) $\gamma_{ij}[k]$ between $\mathbf{x}_i$ and $\mathbf{x}_j$ can be evaluated (e.g., by Welch estimate). Features are band-averaged coherences

$$\bar{\gamma}_{ij,b} = \frac{1}{|B_b|} \sum_{k\in B_b} \gamma_{ij}^2[k] \tag{28.6}$$

over frequency bands B_b of interest (e.g. α, β, γ bands in EEG).

28.3.4. Phase-locking value

For EEG-style phase coupling, let $\phi_c[n]$ be the instantaneous phase of channel c (from the Hilbert transform or a band-limited analytic signal). The phase-locking value (PLV) between channels i, j is

$$\text{PLV}_{ij} = \left| \frac{1}{L} \sum_{n=0}^{L-1} \mathrm{e}^{\,j\left(\phi_i[n]-\phi_j[n]\right)} \right| \in [0,1]. \tag{28.7}$$

PLV$=1$ indicates perfect phase synchrony; PLV$=0$ indicates uniform phase difference.

[1] The significance of these off-diagonal coefficients is tested by Granger causality (Sec. 23.5).

28.3.5. Common spatial patterns (CSP)

For binary classification, CSP learns K spatial filters $\mathbf{w}_k \in \mathbb{R}^C$ that maximize the variance ratio between the two classes:

$$\mathbf{w}_k = \operatorname*{argmax}_{\mathbf{w}} \frac{\mathbf{w}^\top \boldsymbol{\Sigma}^{(1)} \mathbf{w}}{\mathbf{w}^\top \boldsymbol{\Sigma}^{(2)} \mathbf{w}}, \tag{28.8}$$

where $\boldsymbol{\Sigma}^{(c)}$ is the average covariance matrix of class c. Each filter $\mathbf{w}_k$ projects the window onto a latent channel $\mathbf{w}_k^\top \mathbf{X} \in \mathbb{R}^L$; its variance is large for one class and small for the other, so the K log-variances

$$z_k = \log \operatorname{Var}[\mathbf{w}_k^\top \mathbf{X}], \qquad k = 1, \ldots, K, \tag{28.9}$$

form the K-dimensional CSP feature vector for that window (the log makes the features more nearly Gaussian for the downstream classifier). CSP is standard in brain–computer interfaces; like any supervised feature, $\mathbf{w}_k$ must be fitted on the training set only.

28.3.6. Learned representations

End-to-end deep models (1D-CNN, RNN, transformer) operate directly on $\mathbf{X}$ and subsume both per-channel and cross-channel feature extraction as learned representations. They trade interpretability and small-data robustness for raw performance on large datasets.

High number of cross-features

Feature selection (Sec. 9.4) is strongly recommended: number of cross-channel features grows quickly with C, and of these features are redundant.

28.4. Dimensionality reduction across channels

Goal: When C is large, project onto $C' \ll C$ latent channels before per-channel feature extraction.

Beyond $C \gtrsim 10$ the cross-channel feature count grows as $O(C^2)$ and the per-channel block becomes unwieldy. A remedy is to first reduce the C channels to $C' \ll C$ latent channels with a global *linear projection*, fixed once for the whole record, then run the per-channel pipeline of Sec. 28.2 on the reduced set.

Linear projection

Let $\mathbf{X} \in \mathbb{R}^{C \times L}$ be a window. Fit a linear projection $\mathbf{W} \in \mathbb{R}^{C' \times C}$ and form

$$\mathbf{X}' = \mathbf{W}\mathbf{X} \in \mathbb{R}^{C' \times L}, \tag{28.10}$$

then apply the univariate feature map to each of the C' rows of $\mathbf{X}'$. Common choices for $\mathbf{W}$:

- *PCA* (Sec. 11.2): rows of $\mathbf{W}$ are the top-C' eigenvectors of the training covariance matrix $\boldsymbol{\Sigma}_{\text{train}}$. Captures maximum variance; label-agnostic.

- *Independent component analysis (ICA)*: rows of $\mathbf{W}$ are spatial filters that unmix the channels into statistically independent sources. Where PCA only decorrelates the channels (a second-order property), ICA seeks full statistical independence, so it can separate physically distinct generators that PCA leaves mixed. Unsupervised; widely used for artifact removal in EEG, where ocular and muscle components are isolated and dropped before reconstruction.
- *CSP* (Sec. 28.3.5): a supervised filter for binary classification. Its rows maximize the variance of the projection for one class while minimizing it for the other, obtained from the generalized eigendecomposition of the two class covariance matrices. The leading and trailing filters give the most class-discriminative latent channels, but, being supervised, $\mathbf{W}$ must be fitted on the training set only.

28.5. Multivariate decomposition (*)

Goal: Opposite to the linear projection of Sec. 28.4: rather than *reduce* the C channels to fewer latent ones, re-represent each channel as its own set of adaptive, narrowband *modes*, so the feature map sees cleaner components and a narrowband signature lands in a single mode.

The linear projection of Sec. 28.4 applies one global matrix $\mathbf{W}$, fitted across windows, to *reduce* the channels. An adaptive decomposition goes the other way: it lets each signal dictate, per window, a small set of constituent *modes*, each isolating a single oscillation, system dynamic, or fault signature. This *expands* one channel into several mode-channels rather than shrinking C; the univariate feature map of Ch. 26 is then applied mode-by-mode rather than channel-by-channel, which concentrates a narrowband fault signature in one mode instead of spreading it across the full-band channel record. Dimensionality is reduced only if a few informative modes are then kept and the rest discarded. These methods are common in vibration-based fault diagnosis and in biomedical signals. The four methods detailed below differ in what fixes the modes: the data alone (EMD), a prescribed mode budget (VMD), a linear dynamical model (DMD), or an eigenbasis of the signal's own delayed copies (MSSA). They are illustrated throughout on a common test signal (Fig. 28.3), a slow tone plus a short high-frequency burst,

$$x(t) = \underbrace{\sin(2\pi f_1 t)}_{\text{slow tone}} + \underbrace{a\,\mathrm{e}^{-(t-t_0)^2/2\tau^2}\,\sin(2\pi f_2 t)}_{\text{HF burst}} + \varepsilon(t), \tag{28.11}$$

with slow frequency $f_1 = 3$ Hz, burst frequency $f_2 = 18$ Hz, burst amplitude $a = 0.8$, center $t_0 = 1$ s, width $\tau = 0.18$ s, and a small noise term $\varepsilon(t)$. The decompositions react differently: EMD adapts the number of modes to the data and tends to spread the burst across the first two of them; VMD splits the signal into a prescribed number K of band-limited modes that isolate the slow tone and the burst; DMD fits a handful of modes, each a single sinusoid with its own frequency and growth rate; and MSSA recovers each oscillation as a pair of data-adaptive components. Each subsection below defines a method, illustrates it on this signal, and explains how its number of modes is set.

28.5.1. Empirical mode decomposition (EMD)

EMD is fully data-driven and fixes no basis in advance. It expands the signal into *intrinsic mode functions* (IMFs): components that are oscillatory enough to have a meaningful instantaneous frequency. Concretely, an IMF must satisfy two conditions: (i) its number of local extrema and zero-crossings are equal or differ by one (no riding waves), and (ii) at every instant the mean of the

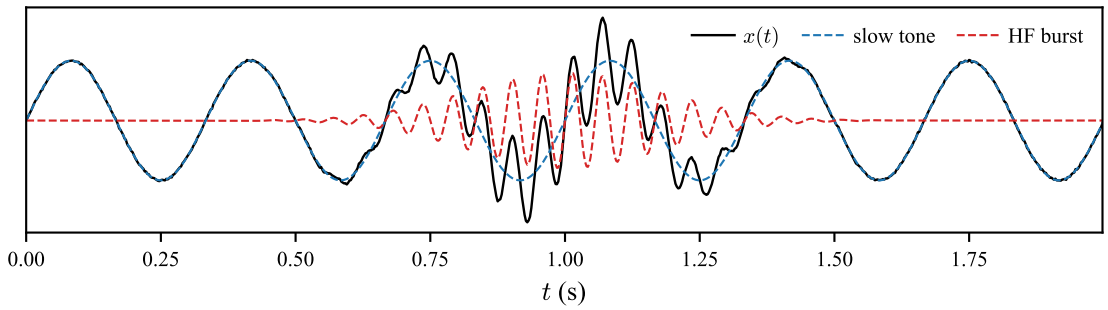


Figure 28.3.: Common test signal for the three decomposition methods: a slow 3 Hz tone plus a Gaussian-windowed 18 Hz burst (Eq. (28.11)), shown with its two true components dashed.

upper envelope (through the maxima) and the lower envelope (through the minima) is zero, so the waveform is locally symmetric about zero.

An IMF is extracted by *sifting*, the iterative inner loop of EMD. One sift spline-interpolates the local maxima into an upper envelope and the local minima into a lower envelope, and subtracts their mean from the signal; the result is sifted again, and again, until the two IMF conditions hold. That IMF is then subtracted from the signal and the whole sifting loop restarts on the residual to yield the next IMF, peeling components from highest to lowest frequency until only a monotonic trend remains:

$$x[n] = \sum_{k=1}^{K_{\mathrm{EMD}}} d_k[n] + r[n], \tag{28.12}$$

with IMFs d_k and residual trend r.

Choosing k. There is nothing to choose: the count K_{EMD} is decided by the data (and the sift stopping rule), and is of order $\log_2 L$ for a broadband window. The band occupied by IMF k is signal-dependent, so k carries no fixed physical meaning and can denote different content from one window or channel to the next. EMD is also prone to *mode mixing*, where one scale spreads across adjacent IMFs (the burst in Fig. 28.4) or one IMF mixes disparate scales; the ensemble variants EEMD and CEEMDAN average sifts over many added-noise realizations to suppress it.

28.5.2. Variational mode decomposition (VMD)

VMD splits the signal into a prescribed number K of modes that add back to it,

$$x(t) = \sum_{k=1}^{K} u_k(t), \tag{28.13}$$

with each mode u_k a narrowband oscillation concentrated around its own center frequency ω_k. The modes are found jointly, not peeled off one at a time as in EMD, and their number K is fixed in advance, which makes VMD markedly more robust to noise and to mode mixing. A bandwidth parameter α controls how tightly each mode is held to its band.

Choosing k. Here K is a hyperparameter fixed in advance, the number of modes the user expects. Too small a K *under-bins* the signal, forcing distinct components to share a mode; at $K = 1$ in

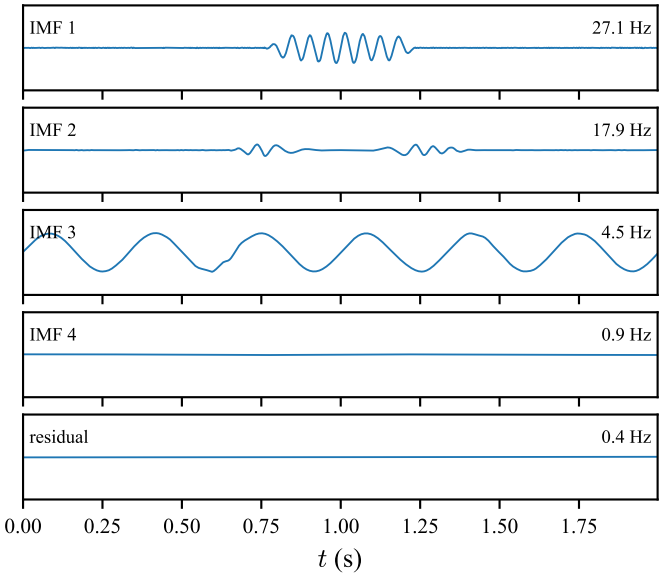


Figure 28.4.: EMD of the test signal: the data-driven set of intrinsic mode functions plus the residual trend, each labeled with its spectral-centroid center frequency. The number of IMFs is set by the data, not chosen, and the 18 Hz burst is split between IMF 1 and IMF 2 (mode mixing). All panels share a common amplitude scale.

Fig. 28.5 the burst is lost into the dominant tone. Too large a K *over-splits* it, so surplus modes latch onto noise (the extra modes at $K = 4$). The matched $K = 2$ isolates the slow tone and the burst. In practice K is set from the number of narrowband components expected in the application, or by increasing it until the last mode carries only residual energy or two center frequencies ω_k collide. Because K and the band structure are prescribed, the ω_k are stable across windows and channels, which is what makes VMD modes alignable where EMD modes are not.

28.5.3. Dynamic mode decomposition (DMD)

DMD models the signal as evolving under a single linear rule, $\mathbf{x}_{n+1} \approx \mathbf{A}\mathbf{x}_n$, and returns *spatio-temporal* modes: each mode is a fixed spatial pattern $\boldsymbol{\phi}_k$ (the *where*, constant over time) that evolves as a single complex exponential λ_k^n (the *when*). From its eigenvalue λ_k each mode carries an explicit frequency and growth or decay rate,

$$f_k = \frac{|\angle \lambda_k|}{2\pi\,\Delta t}, \qquad \sigma_k = \frac{\ln|\lambda_k|}{\Delta t}, \tag{28.14}$$

with $\sigma_k < 0$ a decaying transient, $\sigma_k > 0$ a growing one, and $\sigma_k \approx 0$ a sustained oscillation. "Explicit growth rate" means σ_k is a number the method reports for each mode, unlike the EMD and VMD modes, which are plain time series with no such tag. A single channel carries no spatial dimension, so it is first *time-delay (Hankel) embedded* into a stack of shifted copies before DMD is applied.

Choosing k. The number of modes is the rank r at which the dynamics are truncated. Real signals give complex-conjugate eigenvalue pairs, so r modes describe about $r/2$ oscillations, each with its own (f_k, σ_k). Figure 28.6 sweeps r: too small ($r = 2$) captures only the 3 Hz tone and misses the burst; the matched $r = 4$ recovers both the 3 and 18 Hz oscillations as sustained modes ($\sigma \approx 0$);

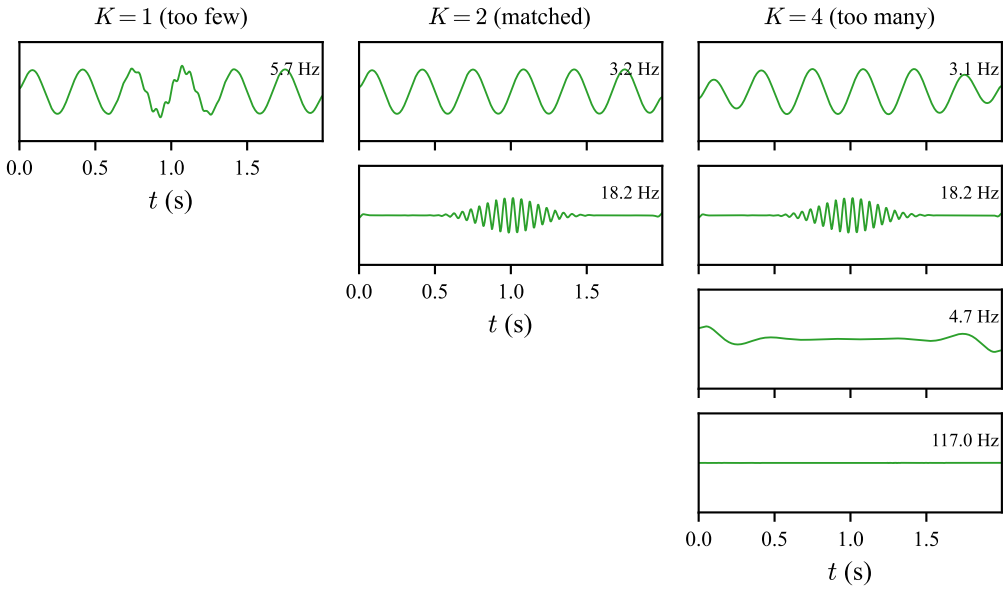


Figure 28.5.: Influence of the mode budget K in VMD. $K = 1$ under-bins the signal into one blurred mode that loses the burst; the matched $K = 2$ cleanly isolates the 3 Hz tone and the 18 Hz burst; $K = 4$ over-splits, with two surplus modes latching onto a near-tone artifact and broadband noise. Each mode is labeled with its center frequency, and all panels share a common amplitude scale.

too large ($r = 8$) adds spurious high-frequency, fast-decaying modes that are then discarded. The rank is chosen by keeping the dominant singular values of the data.

28.5.4. Multichannel eigendecomposition (MSSA)

Like DMD, singular spectrum analysis (SSA) starts from the *time-delay (Hankel) embedding* of the signal: the length-L record is cut into overlapping length-L_w segments stacked as the columns of a trajectory matrix $\mathbf{X}_{\text{traj}} \in \mathbb{R}^{L_w \times (L - L_w + 1)}$. SSA then eigendecomposes this matrix (an SVD of $\mathbf{X}_{\text{traj}}$, equivalently an eigendecomposition of its covariance), giving a set of *eigentriples*, each a data-adaptive temporal pattern with a weight. Grouping the eigentriples and averaging each group back along the anti-diagonals reconstructs additive components,

$$x[n] = \sum_{g} c_g[n], \tag{28.15}$$

so the modes are an orthogonal basis learned from the signal's own delayed copies rather than a fixed or dynamical one.

Eigentriple pairing. A pure oscillation does not map to one eigentriple but to a near-equal *pair* (its sine and cosine phases), so a clean tone occupies two adjacent components; a slow trend takes the leading triple, and broadband noise spreads over the long tail of small eigentriples. The paired plateaus in the scree plot of Fig. 28.7 make this structure visible and tell where the informative components end and the noise tail begins.

Multichannel (MSSA). Where this section's EMD, VMD, and DMD act on one channel at a time, the multichannel form stacks the per-channel trajectory matrices into one block-Hankel matrix and

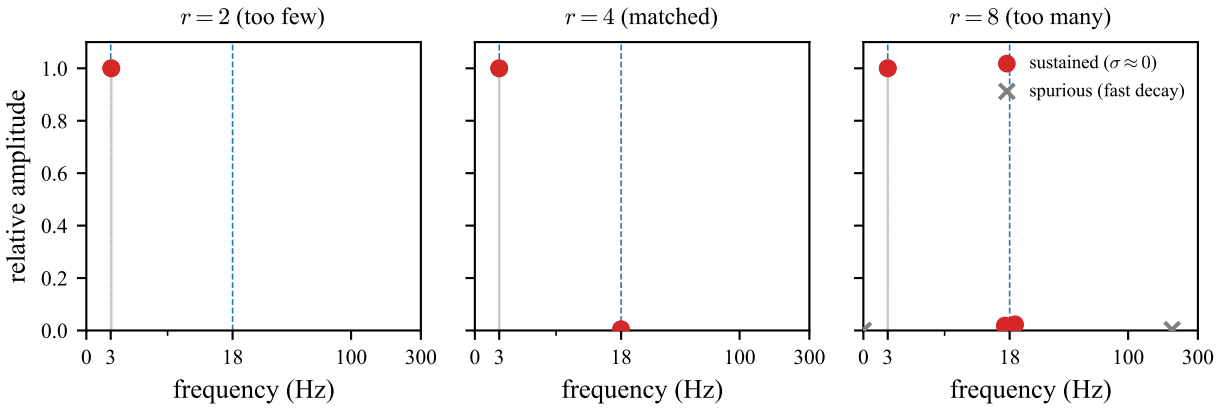


Figure 28.6.: Influence of the truncation rank r in DMD, shown as the recovered modes in the frequency–amplitude plane (relative amplitude versus mode frequency; the true 3 and 18 Hz bands dashed). $r = 2$ recovers only the dominant tone; the matched $r = 4$ adds the burst, both as sustained modes ($\sigma \approx 0$); $r = 8$ introduces spurious, fast-decaying modes (crosses) that contribute little and are dropped.

eigendecomposes them *jointly*, yielding a *common* set of temporal components shared by all C channels. Mode k is therefore the same band in every channel by construction, an eigendecomposition route to the cross-channel alignment that MEMD and MVMD achieve, so MSSA needs no separate alignment step.

Choosing k. Two knobs set the result: the window length L_w (the embedding dimension, which fixes the frequency resolution, finer as L_w grows toward $L/2$) and the number of leading eigentriples kept and how they are grouped. Because oscillations come in pairs, the retained rank is about $2 \times$ (number of tones) plus one for a trend; the scree's paired plateaus (Fig. 28.7) guide the cut, with the small-eigenvalue tail discarded as noise.

28.5.5. Multivariate variants and mode alignment

The EMD and VMD above are *univariate*: run independently per channel, they break a property the cross-channel pipeline needs, *mode alignment*, that the k-th mode mean the same thing across channels before its features can be compared or fused. The mode count and the center frequency of mode k drift from channel to channel. Their *multivariate* counterparts, multivariate EMD (MEMD) and multivariate VMD (MVMD), fix this by construction, extracting a *common* set of modes jointly across all C channels so that mode k occupies the same frequency band in every channel.

Example 28.2 (Aligning a resonance burst across axes): A 3-axis accelerometer ($C = 3$) on a rotating machine records, per window, a slow imbalance tone plus a short high-frequency burst from an incipient bearing fault that excites the three axes with different amplitudes and arrival times. A fixed band split (per-channel Fourier or wavelet features) smears the burst across several bins, and per-channel EMD returns a different number of IMFs per axis, so "IMF 2" of the x-axis need not match "IMF 2" of the y-axis. Running MVMD with, say, $K = 4$ common modes instead yields one burst-dominated mode shared by all three axes. The univariate map then produces aligned per-mode features, for example the log-energy $\log \mathrm{Var}[\cdot]$ of the burst mode on each axis, that are directly comparable across channels and feed the cross-channel block of Sec. 28.3 (the same energy-of-a-latent-channel idea as the CSP features of Eq. (28.9)).

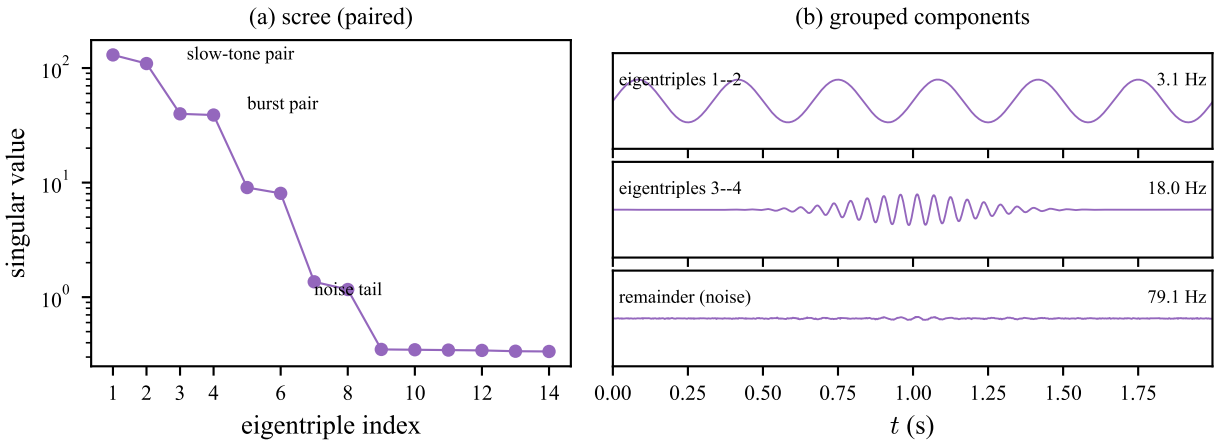


Figure 28.7.: SSA of the test signal. (a) Scree plot of the trajectory-matrix singular values: they fall in near-equal *pairs*, one pair per oscillation (slow tone, burst), above a flat noise tail. (b) Grouping each pair and reconstructing recovers the 3 Hz tone (eigentriples 1–2) and the localized 18 Hz burst (eigentriples 3–4); the remaining small eigentriples form a low-energy noise component. Each component is labeled with its spectral-centroid center frequency, and the panels share a common amplitude scale.

Figure 28.8 makes this alignment problem concrete on the three-axis signal: per-channel EMD returns a different number of IMFs per axis and the center frequency of mode k drifts from axis to axis, whereas per-channel VMD with a common K locks the modes onto shared frequency bands; MVMD enforces this equality exactly.

Possible data leakage

The linear projection $\mathbf{W}$ (PCA, ICA, CSP) is fitted across samples and must be estimated on the *training* set only, then applied unchanged to the test set (Sec. 4.7). Fitting it on the combined train+test set leaks information and inflates accuracy. The decompositions above (EMD, VMD, DMD, MSSA) are computed independently per window and carry no such cross-sample fit.

28.6. Cross-channel model transfer and linear alignment

Goal: The per-channel pipelines of Sec. 28.2 require training C separate models. Test whether a classifier fitted on one *source* channel can be applied to the remaining *target* channels without retraining.

28.6.1. Setup: from vectors to channels

In vector terms, a fitted classifier is a fixed function on the feature space, and applying it to samples drawn from a shifted distribution is exactly the dataset-shift situation of Sec. 13.2.2. The multivariate signal restates this generic vector setting with two additional structural properties.

Let channel s be the source and channel t a target. Applying the same per-channel feature map

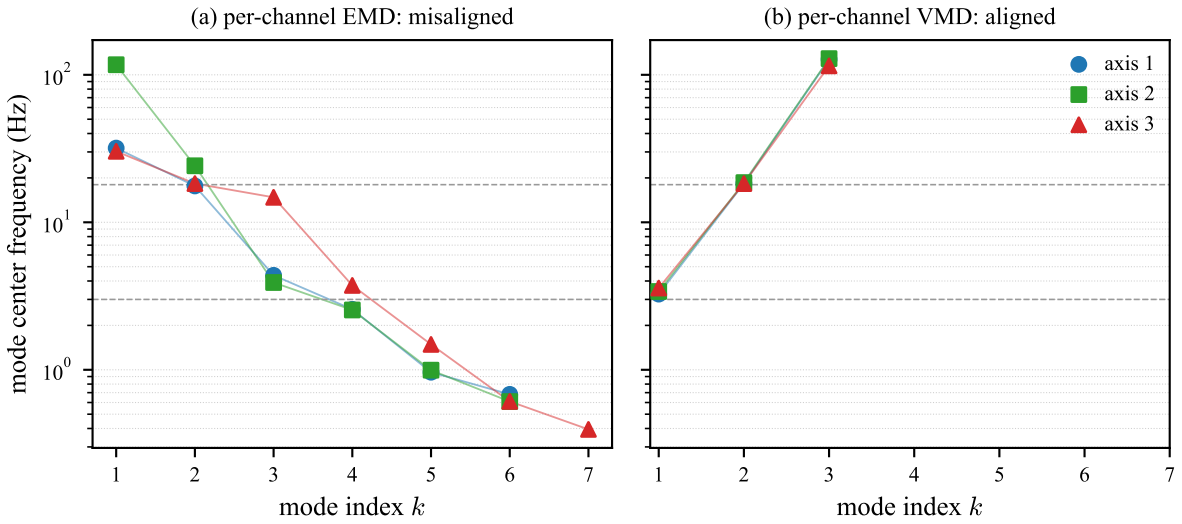


Figure 28.8.: Cross-channel mode alignment on the three-axis resonance-burst signal: mode center frequency (spectral centroid) versus mode index k. (a) Per-channel *EMD* returns a different number of IMFs per axis and the frequency of mode k drifts across axes, so "IMF k" does not denote the same band on every axis. (b) Per-channel *VMD* with a common $K = 3$ locks modes 1 and 2 onto the shared 3 and 18 Hz bands (dashed) on every axis. MVMD enforces a common mode set exactly by construction.

$f : \mathbb{R}^L \to \mathbb{R}^N$ (Sec. 28.2) to the M training windows of each channel gives two training feature matrices

$$\mathbf{Z}_s,\ \mathbf{Z}_t \in \mathbb{R}^{M \times N}, \tag{28.16}$$

with the following properties:

- *Shared feature space.* The same feature map is applied to every channel, so the columns of $\mathbf{Z}_s$ and $\mathbf{Z}_t$ have identical dimension and meaning. Generic multimodal sources (e.g. EEG combined with EMG, Ch. 15) lack this property.
- *Paired rows.* The channels are synchronously sampled (chapter preface), so row i of $\mathbf{Z}_s$ and row i of $\mathbf{Z}_t$ describe the *same* window of the same physical event. The two matrices are paired row by row, and the pairing requires no labels. Generic tabular domains rarely offer this.

The *transfer experiment*: train the full pipeline (feature extraction, normalization, classifier) on the training windows of the source channel and apply it, unchanged, to the test windows of every channel, with the same grouped train/test split (Sec. 4.5) used for all channels. Collecting the accuracies yields a $C \times C$ transfer matrix whose diagonal holds each channel's native within-channel accuracy; the off-diagonal entries are directly comparable to it.

28.6.2. Why direct transfer fails

Each channel observes the same underlying event through its own measurement chain. In feature space this appears as a differently calibrated coordinate system: per-feature gain and offset (sensor sensitivity, distance to the event) and feature mixing (frequency response, mechanical or electrical coupling). The decision boundary learned in the source coordinates then cuts the target clouds in the wrong place, even when the target channel, with its *own* trained model, is just as accurate as

the source.

Example 28.3 (Gain and offset mismatch): A single feature z separates two classes on the source channel: class A training values $\{-2, -1\}$, class B training values $\{1, 2\}$, and the fitted rule predicts B when $z > 0$. The target channel observes the same four events with half the gain and an offset of $+2$:

$$z' = \tfrac{1}{2}z + 2 \quad \Rightarrow \quad \text{A} \to \{1,\ 1.5\}, \qquad \text{B} \to \{2.5,\ 3\}.$$

All four target values are positive, so the source rule predicts B everywhere: accuracy drops from 100% to 50%, chance level, although the target values separate the classes perfectly.

Equivalent sensors, inconsistent domains

Channels can be equivalent *as sensors* (each reaches the same accuracy with its own model) and still be inconsistent *as domains*: the classifier two-sample test of Sec. 13.2.2 easily distinguishes their feature distributions. Direct transfer failure is a dataset-shift event, not evidence that the target channel is less informative.

28.6.3. Linear alignment

An *alignment* is a map, learned on training data only, that transports target features into the source coordinate system before the source classifier reads them. The class labels are never used. The three standard linear alignments below are previously presented vector tools re-purposed; they are ordered by the statistics they match, and therefore by the distortions they can undo.

Moment matching

Each feature is standardized with the target's training statistics and de-standardized with the source's, i.e. the standardization of Sec. 4.7 applied twice:

$$z'_j = \frac{z_j - \mu_{t,j}}{\sigma_{t,j}}\,\sigma_{s,j} + \mu_{s,j}, \qquad j = 1, \ldots, N, \tag{28.17}$$

where $(\mu_{t,j}, \sigma_{t,j})$ and $(\mu_{s,j}, \sigma_{s,j})$ are the per-feature training means and standard deviations of the target and source channels. Moment matching undoes any per-feature gain and offset, as in Example 28.3, where (28.17) recovers the source values exactly. It cannot undo feature mixing, since each feature is corrected in isolation.

Covariance alignment (CORAL)

Correlation alignment (CORAL) matches the full second-order structure: the target features are whitened with the target training covariance and re-colored with the source covariance,

$$\mathbf{z}' = \boldsymbol{\Sigma}_s^{1/2}\,\boldsymbol{\Sigma}_t^{-1/2}\,(\mathbf{z} - \boldsymbol{\mu}_t) + \boldsymbol{\mu}_s, \tag{28.18}$$

where $\boldsymbol{\Sigma}_s, \boldsymbol{\Sigma}_t \in \mathbb{R}^{N\times N}$ are the training covariance matrices (the $\tau = 0$ object of Sec. 28.3.1) and the matrix square roots are computed from their eigendecompositions, as in PCA (Sec. 11.2).

Distribution matching is blind to rotations

Any transform that *preserves* the feature distribution is invisible to an alignment that only matches that distribution. A rotation or reflection of the feature space can leave all means and covariances unchanged, so neither moment matching (28.17) nor CORAL (28.18) can detect it, let alone undo it. Worse, the re-coloring in (28.18) is only determined up to such a rotation, and the arbitrary choice may rotate exactly the directions the source classifier relies on; CORAL can then underperform plain moment matching.

Paired linear map

The strongest linear alignment uses the property the previous two ignore: the rows of $\mathbf{Z}_s$ and $\mathbf{Z}_t$ are paired. Center each matrix with its own training means, $\tilde{\mathbf{Z}}_s = \mathbf{Z}_s - \mathbf{1}_M \boldsymbol{\mu}_s^\top$ and $\tilde{\mathbf{Z}}_t = \mathbf{Z}_t - \mathbf{1}_M \boldsymbol{\mu}_t^\top$, and fit a regularized linear map from the target space to the source space,

$$\mathbf{W} = \underset{\mathbf{W}\in\mathbb{R}^{N\times N}}{\arg\min} \left\| \tilde{\mathbf{Z}}_s - \tilde{\mathbf{Z}}_t \mathbf{W} \right\|_F^2 + \lambda \|\mathbf{W}\|_F^2 = \left(\tilde{\mathbf{Z}}_t^\top \tilde{\mathbf{Z}}_t + \lambda \mathbf{I} \right)^{-1} \tilde{\mathbf{Z}}_t^\top \tilde{\mathbf{Z}}_s, \tag{28.19}$$

where $\|\cdot\|_F$ is the Frobenius norm (the square root of the sum of squared entries). At test time the alignment is $\mathbf{z}' = \mathbf{W}^\top(\mathbf{z} - \boldsymbol{\mu}_t) + \boldsymbol{\mu}_s$. Eq. (28.19) is ridge regression (Sec. 4.9) with N outputs, in which the *source features* play the role of the regression targets; no class labels appear anywhere. Distribution matching can never decide which target direction corresponds to which source direction, but the row pairing resolves the correspondence by construction, so the paired map recovers arbitrary linear mixing, including the rotations that defeat moment matching and CORAL.

Example 28.4 (Feature swap): Four paired training windows are recorded by both channels. On the source channel the class is carried by z_1 alone:

$$\mathbf{Z}_s = \begin{pmatrix} -1 & 1 \\ -1 & -1 \\ 1 & 1 \\ 1 & -1 \end{pmatrix} \begin{matrix} \text{A} \\ \text{A} \\ \text{B} \\ \text{B} \end{matrix} \qquad \mathbf{Z}_t = \begin{pmatrix} 1 & -1 \\ -1 & -1 \\ 1 & 1 \\ -1 & 1 \end{pmatrix},$$

with the fitted source rule: predict B when $z_1 > 0$. The target channel swaps the two features, $\mathbf{z}' = (z_2, z_1)$. Compare the three alignments.

Solution: *Direct transfer.* The target first feature equals the source z_2, which is independent of the class: accuracy 50%.

Moment matching. In both channels every feature has training mean 0 and standard deviation 1, so (28.17) is the identity map: still 50%.

CORAL. Both training covariances equal $\mathbf{I}$ (the features are uncorrelated with unit variance), so (28.18) is also the identity map: still 50%. The swap preserves the feature distribution, so no distribution-matching alignment can see it.

Paired map. The means are zero, so centering changes nothing, and

$$\mathbf{Z}_t^\top \mathbf{Z}_t = 4\,\mathbf{I}, \qquad \mathbf{Z}_t^\top \mathbf{Z}_s = 4 \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} = 4\,\mathbf{P},$$

where $\mathbf{P}$ is the swap (permutation) matrix. With $\lambda \to 0$, Eq. (28.19) gives $\mathbf{W} = \mathbf{P}$ exactly; with $\lambda = 1$, $\mathbf{W} = \frac{4}{5}\mathbf{P}$, which shrinks the features but produces the same decisions. The pairing identifies which target column corresponds to which source column, the swap is undone, and the accuracy returns to 100%.

Figure 28.9 illustrates the same hierarchy on Gaussian feature clouds whose target version is rotated, scaled, and shifted: direct transfer collapses to chance, moment matching recovers the per-feature scale and offset but not the rotation, and the paired map restores essentially the native accuracy.

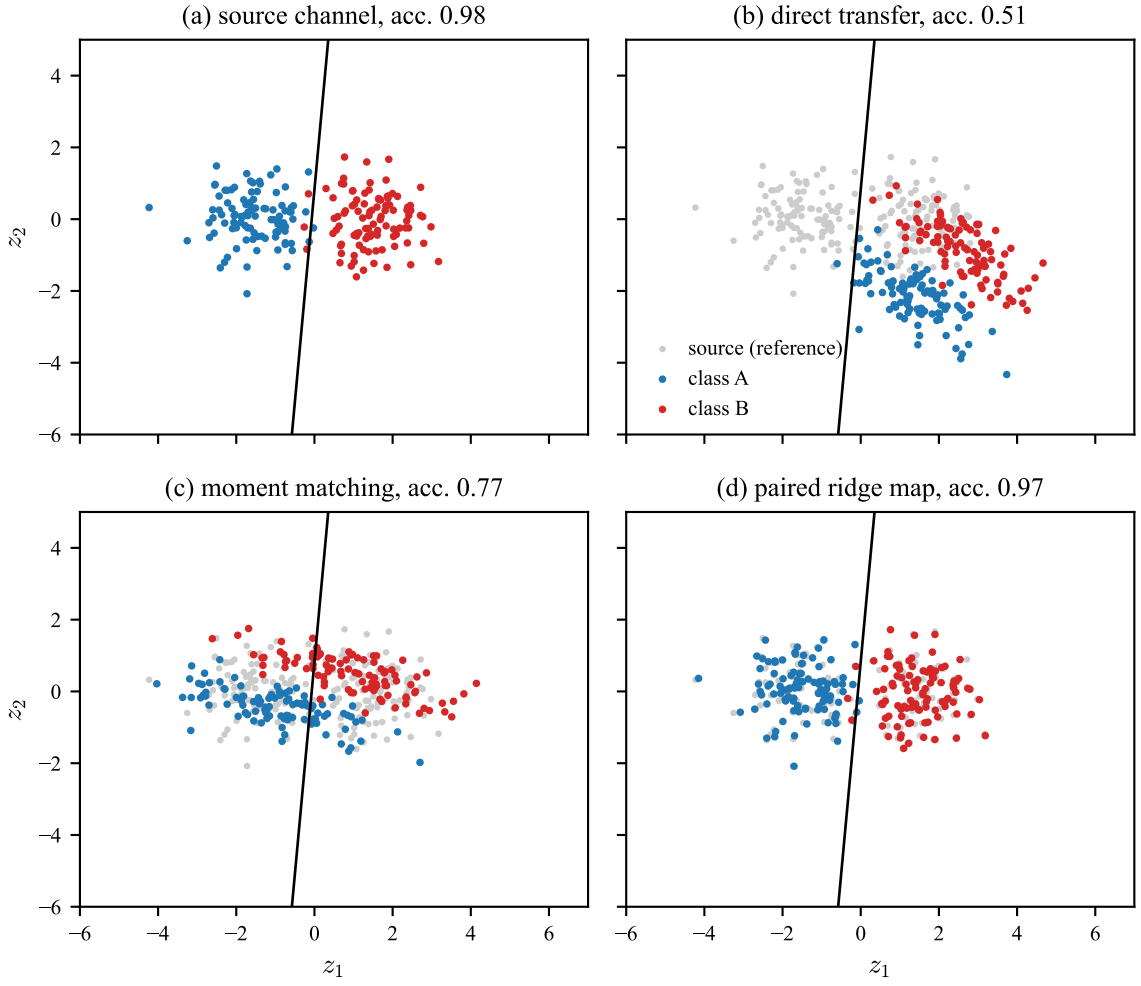


Figure 28.9.: Cross-channel model transfer in a two-feature toy problem. The linear decision boundary (black line) is fitted once, on the source channel, and is identical in all panels; gray points are the source-channel reference clouds. (a) Native performance on the source channel. (b) The target channel reports the same events through a rotation, anisotropic gain, and offset, and direct transfer collapses to chance. (c) Moment matching (28.17) restores each feature's offset and scale, but the rotation remains. (d) The paired ridge map (28.19) also undoes the rotation and restores essentially the native accuracy.

28.6.4. Summary and practice

Method	Statistics matched	Paired rows	Distortions undone
Moment matching (28.17)	per-feature mean and std	not needed	per-feature gain and offset
CORAL (28.18)	mean and full covariance	not needed	correlation structure, up to a rotation ambiguity
Paired map (28.19)	row correspondence (ridge fit)	required	any linear mixing

Table 28.1.: Linear alignment methods for cross-channel model transfer.

Alignment is fitted, so it can leak

All alignment parameters ($\boldsymbol{\mu}$, σ_j, $\boldsymbol{\Sigma}$, $\mathbf{W}$) are data-dependent transformations. Like normalization (Sec. 4.7) and PCA, they must be estimated on the training partition only and applied unchanged to the test partition; estimating them on the full dataset inflates the transfer accuracy.

Calibration transfer

The paired map gives a practical recipe for commissioning an additional channel: record paired, *unlabeled* data with the already-deployed channel, fit (28.19), and reuse the existing classifier, instead of collecting a labeled training set per channel. Native per-channel training remains the accuracy ceiling, so alignment is a deployment shortcut, not a replacement for it.

Beyond signals

Nothing in (28.17)–(28.19) is signal-specific: the methods apply to any tabular problem in which source and target share one feature space, such as the same instrument type at two sites or two scanners measuring the same specimens. The multichannel signal setting is special only in that synchronous sampling supplies the paired rows for free.

Whether an aligned transfer still falls short of the target channel's native model is a paired comparison of two classifiers on the same test windows, assessed with the tools of Ch. 14, in particular McNemar's test (Sec. 14.1.4).

Part IV.

Appendix

A. Notation

Numbers and indexing

a	Scalar
$\mathbf{a}$	Vector
a_i	Element i of a vector a, indexing starting at 1
$\mathbf{A}$	Matrix
a_{ij}	Element i, j of a matrix $\mathbf{A}$, indexing starting at 1
$\mathbb{R}$	Real numbers domain
$\mathbb{R}^D$	D-dimensional vector
$\mathbb{R}^{D_1 \times D_2}$	matrix of a dimension $D_1 \times D_2$
$\mathbf{I}$	Identity matrix
$\mathbf{1}$	Vector/matrix of ones
$\mathbf{0}$	Vector/matrix of zeros
$\mathbb{1}$	Indicator function (Sec. B.2)

Datasets

L	Model complexity
N	Number of features
M	Number of entries in the dataset
K	Number of classes
Δ^{K-1}	Probability simplex: $\{\mathbf{p} \in \mathbb{R}^K : p_i \geq 0,\ \sum_i p_i = 1\}$
$\mathbf{w}$ or w_i	Model parameters (vector form)
$f(\cdot; \mathbf{w})$	Model
$h(\mathbf{x})$ or $h(x)$	True unknown function
x_{ij}	Single data value
$\mathbf{x}_i$	Single data vector (sample i); $\mathbf{x}_i^T$ is the i-th row of $\mathbf{X}$
$\tilde{\mathbf{x}}_j$	j-th column (feature) of $\mathbf{X}$
$\mathbf{X}$	Data matrix
$\mathbf{y}$	Target vector for the data in $\mathbf{X}$
$\hat{\mathbf{y}}$	Prediction vector of $\mathbf{y}$
y_i	Target value
$\hat{y}_i$	Predicted target value
$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$ or $\mathcal{L}(y_i, \hat{y}_i)$	Loss function
λ	Regularization parameter
$\mathbf{a}^{[k]}$	Activation of layer k
$\mathbf{z}^{[k]}$	Output of layer k
$g_k(\cdot)$	Activation function of layer k
$\boldsymbol{\theta}$ or θ_i	Model parameters (general form)
$\boldsymbol{\alpha}$	Kernel/dual coefficients vector
C	Number of clusters
$\mathcal{C}_c$	Cluster c; $\lvert\mathcal{C}_c\rvert$ is its size
$\boldsymbol{\mu}_c$	Center (centroid) of cluster c
$d(\mathbf{x}, \mathbf{x}')$	Distance between two samples
$\boldsymbol{\gamma}_i$ or γ_{ic}	Soft cluster assignment (responsibility) of sample i
$\mathbf{e}$	Error/residual vector
$\boldsymbol{\epsilon}$ or ϵ_i	Noise vector/term
$\mathbf{n}$	Noise vector (signal processing)
$\mathbf{h}$	Impulse response / filter coefficients
$\mathbf{P}$	Projection matrix
$\mathbf{K}$	Kernel matrix
$\mathbf{R}$	Autocorrelation matrix
$\phi(\cdot)$	Feature mapping / basis function
α	Learning rate (gradient descent step size)

Statistics

x	Sample set
$\bar{x}$	Sample mean
s_x^2	Sample variance (biased or unbiased)
s_x	Sample std (biased or unbiased)
s_{xy}	Sample covariance (biased or unbiased)
r_{xy}	Sample correlation coefficient
μ	Population mean
σ^2	Population variance
σ	Population standard deviation
$\mathbb{E}[\cdot]$	Expectation operator
$\mathrm{Var}[\cdot]$	Variance operator
$\mathrm{Cov}[\cdot]$	Covariance operator
$\mathrm{SE}(\cdot)$	Standard error of an estimate

Signals

ω	Angular frequency (discrete)
θ	Phase angle
A	Amplitude
F	Frequency [Hz]
F_s	Sampling frequency
T	Period [sec]
x_{rms}	Root mean square of a signal
$\mathrm{CF}_{\mathbf{x}}$	Crest factor

B. Essential formulas

B.1. Linear algebra

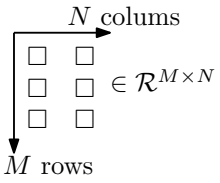


Figure B.1.: Vector and matrix dimensions notation.

B.1.1. Basic operations

For a column vector $\mathbf{z} = [z_1, z_2, \ldots, z_N]^T \in \mathbb{R}^N$, the squared Euclidean norm is

$$\|\mathbf{z}\|^2 = \mathbf{z}^T\mathbf{z} = z_1^2 + z_2^2 + \cdots + z_N^2. \tag{B.1}$$

For matrices $\mathbf{A}$ and $\mathbf{B}$ of compatible sizes, the transpose of a product is

$$(\mathbf{AB})^T = \mathbf{B}^T\mathbf{A}^T. \tag{B.2}$$

B.1.2. Dot product

For two vectors $\mathbf{a}, \mathbf{b} \in \mathbb{R}^N$,

$$\mathbf{a}\cdot\mathbf{b} = \mathbf{a}^T\mathbf{b} = \mathbf{b}^T\mathbf{a} = \|\mathbf{a}\|\,\|\mathbf{b}\|\cos\theta, \tag{B.3}$$

where θ is the angle between them. In particular, $\mathbf{a} \perp \mathbf{b} \;\Leftrightarrow\; \theta = 90^\circ \;\Leftrightarrow\; \mathbf{a}^T\mathbf{b} = 0$.

B.1.3. Pseudo-inverse

When $\mathbf{X} \in \mathbb{R}^{M\times N}$ has full column rank ($M \geq N$ and the columns are linearly independent), the matrix $\mathbf{X}^T\mathbf{X}$ is invertible and the (left, Moore-Penrose) *pseudo-inverse* is

$$\mathbf{X}^+ = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T, \tag{B.4}$$

which satisfies $\mathbf{X}^+\mathbf{X} = \mathbf{I}$ and provides the least-squares solution to overdetermined linear systems.

B.1.4. Least squares

The least-squares loss for a linear model $\mathbf{X}\mathbf{w} \approx \mathbf{Y}$ is

$$\mathcal{L}(\mathbf{w}) = \|\mathbf{Y} - \mathbf{X}\mathbf{w}\|^2 = (\mathbf{Y} - \mathbf{X}\mathbf{w})^T(\mathbf{Y} - \mathbf{X}\mathbf{w}) \tag{B.5}$$

$$= \mathbf{Y}^T\mathbf{Y} - 2\mathbf{w}^T\mathbf{X}^T\mathbf{Y} + \mathbf{w}^T\mathbf{X}^T\mathbf{X}\mathbf{w}. \tag{B.6}$$

Setting the gradient to zero,

$$\frac{\partial \mathcal{L}(\mathbf{w})}{\partial \mathbf{w}} = -2\mathbf{X}^T\mathbf{Y} + 2\mathbf{X}^T\mathbf{X}\mathbf{w} = \mathbf{0}, \tag{B.7}$$

gives the *normal equations* $\mathbf{X}^T\mathbf{X}\mathbf{w} = \mathbf{X}^T\mathbf{Y}$ and the closed-form solution

$$\mathbf{w}^\star = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{Y} = \mathbf{X}^+\mathbf{Y}. \tag{B.8}$$

B.1.5. Eigenvalues and eigenvectors

Definition. Let $\mathbf{A} \in \mathbb{R}^{n\times n}$. A non-zero vector $\mathbf{v} \in \mathbb{R}^n$ is an *eigenvector* of $\mathbf{A}$ with associated *eigenvalue* $\lambda \in \mathbb{R}$ if

$$\mathbf{A}\mathbf{v} = \lambda\mathbf{v}. \tag{B.9}$$

Geometrically, $\mathbf{A}$ acts on any vector along the direction of $\mathbf{v}$ by scaling it by the factor λ, without rotation.

The eigenvalues of $\mathbf{A}$ satisfy the identities

$$\operatorname{tr}(\mathbf{A}) = \sum_{i=1}^{n} a_{ii} = \sum_{i=1}^{n} \lambda_i, \tag{B.10}$$

$$\det(\mathbf{A}) = \prod_{i=1}^{n} \lambda_i. \tag{B.11}$$

Eigendecomposition

Assume $\mathbf{A}$ has n linearly independent eigenvectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_n$ (this assumption, called *diagonalisability*, is non-trivial; not every square matrix satisfies it). Stack them as the columns of

$$\mathbf{Q} = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \end{bmatrix}. \tag{B.12}$$

Applying $\mathbf{A}$ to each column scales it by the corresponding eigenvalue, so column-by-column

$$\mathbf{A}\mathbf{Q} = \begin{bmatrix} \lambda_1\mathbf{v}_1 & \lambda_2\mathbf{v}_2 & \cdots & \lambda_n\mathbf{v}_n \end{bmatrix} = \mathbf{Q}\mathbf{\Lambda}, \tag{B.13}$$

where $\mathbf{\Lambda} = \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$. Multiplying on the right by $\mathbf{Q}^{-1}$ gives the *eigendecomposition* of $\mathbf{A}$,

$$\mathbf{A} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^{-1}, \tag{B.14}$$

$$\mathbf{Q}^{-1}\mathbf{A}\mathbf{Q} = \mathbf{\Lambda}. \tag{B.15}$$

Symmetric case (spectral theorem)

When $\mathbf{A}$ is real symmetric, $\mathbf{A} = \mathbf{A}^T$, the eigendecomposition always exists and has stronger properties: the eigenvalues λ_i are real, and the eigenvectors $\mathbf{v}_i$ can be chosen orthonormal. In that case $\mathbf{Q}$ is an orthogonal matrix, $\mathbf{Q}^{-1} = \mathbf{Q}^T$, and

$$\mathbf{A} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T. \tag{B.16}$$

A sample covariance matrix is symmetric and positive semi-definite, so its eigendecomposition has real non-negative eigenvalues and orthonormal eigenvectors.

Positive (semi-)definite matrices

A real symmetric matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ is called

- *positive definite* (PD), written $\mathbf{A} \succ 0$, if $\mathbf{x}^T\mathbf{A}\mathbf{x} > 0$ for every non-zero $\mathbf{x} \in \mathbb{R}^n$;
- *positive semi-definite* (PSD), written $\mathbf{A} \succeq 0$, if $\mathbf{x}^T\mathbf{A}\mathbf{x} \geq 0$ for every $\mathbf{x} \in \mathbb{R}^n$ (equality is allowed for non-zero $\mathbf{x}$).

Equivalent characterisations in terms of the eigenvalues $\lambda_1, \ldots, \lambda_n$ (which are real because $\mathbf{A}$ is symmetric):

- $\mathbf{A} \succ 0 \;\Leftrightarrow\; \lambda_i > 0$ for all i;
- $\mathbf{A} \succeq 0 \;\Leftrightarrow\; \lambda_i \geq 0$ for all i.

A PD matrix is invertible (none of its eigenvalues is zero); a PSD matrix may be singular. Common examples:

- For any matrix $\mathbf{X}$, the Gram matrix $\mathbf{X}^T\mathbf{X}$ is PSD; it is PD when $\mathbf{X}$ has full column rank.
- A sample covariance matrix is PSD; it is PD when the data spans the full feature space.
- The Hessian of a strictly convex function is PD at every point; the Hessian of a convex function is PSD.

Negative definite ($\mathbf{A} \prec 0$) and negative semi-definite ($\mathbf{A} \preceq 0$) are defined analogously, with all eigenvalues strictly negative or non-positive.

B.1.6. Projections

Vector projection

The vector projection of $\mathbf{a}$ onto $\mathbf{b}$ is the orthogonal projection of $\mathbf{a}$ onto the direction of $\mathbf{b}$ (Fig. B.2). With $\hat{\mathbf{b}} = \mathbf{b}/\|\mathbf{b}\|$ the unit vector in the direction of $\mathbf{b}$,

$$\mathbf{a}_1 = (\mathbf{a} \cdot \hat{\mathbf{b}})\,\hat{\mathbf{b}} = \frac{\mathbf{a}^T\mathbf{b}}{\|\mathbf{b}\|^2}\,\mathbf{b}. \tag{B.17}$$

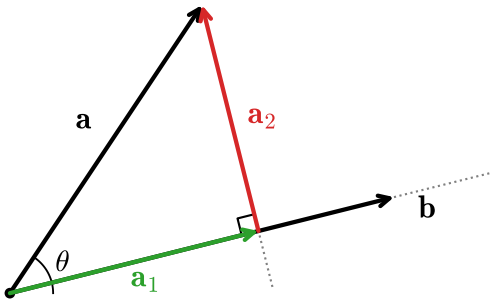


Figure B.2.: $\mathbf{a}_1$ is the projection of $\mathbf{a}$ onto $\mathbf{b}$.

Subspace projection

The projection of a vector $\mathbf{a} \in \mathbb{R}^N$ onto a subspace spanned by the columns of $\mathbf{B} = [\mathbf{b}_1, \mathbf{b}_2, \ldots, \mathbf{b}_k] \in \mathbb{R}^{N \times k}$ (Fig. B.3) is

$$\mathbf{a}_1 = \mathbf{P}\mathbf{a}, \qquad \mathbf{P} = \mathbf{B}(\mathbf{B}^T\mathbf{B})^{-1}\mathbf{B}^T, \tag{B.18}$$

where $\mathbf{P}$ is the orthogonal projection matrix onto the column space of $\mathbf{B}$. It satisfies $\mathbf{P}^2 = \mathbf{P}$ and $\mathbf{P}^T = \mathbf{P}$. The single-vector formula above is the special case $k = 1$.

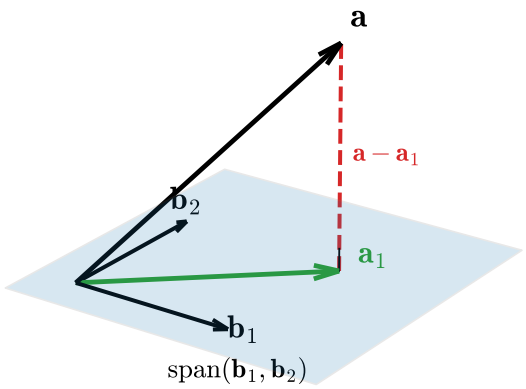


Figure B.3.: Orthogonal projection of $\mathbf{a}$ onto the subspace spanned by $\mathbf{b}_1, \mathbf{b}_2$.

Condition number

The *condition number* of a matrix measures how much small perturbations of the input can be amplified in the output of a linear system. For a non-singular matrix $\mathbf{A}$,

$$\kappa(\mathbf{A}) = \frac{\sigma_{\max}(\mathbf{A})}{\sigma_{\min}(\mathbf{A})}, \tag{B.19}$$

$$\kappa(\mathbf{A}^T\mathbf{A}) = \kappa(\mathbf{A})^2, \tag{B.20}$$

where $\sigma_{\max}$ and $\sigma_{\min}$ are the largest and smallest singular values of $\mathbf{A}$. For a symmetric positive-definite $\mathbf{A}$, the singular values coincide with the absolute eigenvalues, so $\kappa(\mathbf{A}) = |\lambda_{\max}|/|\lambda_{\min}|$. A large condition number indicates that the matrix is close to singular, leading to numerical inaccuracies in matrix calculations; in particular, forming $\mathbf{A}^T\mathbf{A}$ (as in the normal equations) squares the condition number, which is one reason SVD-based solvers are preferred for ill-conditioned least-squares problems.

For example,

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & -\epsilon \\ \epsilon & 0 & 1 \\ 0 & \epsilon & 1 \end{bmatrix} \;\Rightarrow\; \kappa(\mathbf{A}) \approx \sqrt{2}/\epsilon, \quad \kappa(\mathbf{A}^T\mathbf{A}) \approx 2/\epsilon^2. \tag{B.21}$$

B.2. Indicator functions and counting

The indicator $\mathbb{1}[\text{predicate}]$ equals 1 when the predicate is true and 0 otherwise. Two notational patterns recur throughout the book.

B.2.1. Sum of a single indicator

Given labels $Y_1, \ldots, Y_M$ and a fixed class c, the expression

$$\sum_{i=1}^{M} \mathbb{1}[Y_i = c] \tag{B.22}$$

turns the label vector into a column of 0s and 1s, with a 1 in row i whenever $Y_i = c$, and then adds those entries. The sum is simply the number of 1s in that column, i.e. the count of samples whose label is c.

B.2.2. Sum of a product of indicators

A product of two indicators is itself an indicator of the conjunction,

$$\mathbb{1}[A]\,\mathbb{1}[B] = \mathbb{1}[A \text{ and } B], \tag{B.23}$$

because the product is 1 only when both factors are 1. For two label vectors (say the truth Y_i and a prediction $\hat{Y}_i$), the sum

$$\sum_{i=1}^{M} \mathbb{1}[Y_i = c]\,\mathbb{1}[\hat{Y}_i = c'] \tag{B.24}$$

reads two 0/1 columns row by row, multiplies entry by entry, and counts the rows where both columns held a 1. It is the number of samples whose true class is c and whose predicted class is c'.

The entry-by-entry product of two vectors is called the *Hadamard product* and written with $\odot$: if $\mathbf{v}, \mathbf{u} \in \{0,1\}^M$ are the two indicator columns, then $(\mathbf{v} \odot \mathbf{u})_i = v_i u_i$, and the sum above is the total of the entries of $\mathbf{v} \odot \mathbf{u}$.

Example B.1: With $M = 5$, true labels $\mathbf{Y} = (c, c, b, c, a)$ and predictions $\hat{\mathbf{Y}} = (c, b, b, c, b)$, taking c as the positive class and asking "true c predicted as c":

i	1	2	3	4	5	sum
$\mathbb{1}[Y_i = c]$	1	1	0	1	0	3
$\mathbb{1}[\hat{Y}_i = c]$	1	0	0	1	0	2
product	1	0	0	1	0	2

The single-indicator column sums give the marginal counts (3 true c, 2 predicted c); the product column counts samples in both at once (2 correctly predicted as c).

This is the mechanism behind confusion-matrix entries such as Eq. (12.24) and the agreement-matrix entries in Sec. 14.3.2.

B.3. Function analysis

B.3.1. Minimum, maximum, argmin, argmax

For a real-valued function $f(x)$:

- $\min_x f(x)$ is the smallest *value* attained by f over all possible x.
- $\arg\min_x f(x)$ is the *location* where that minimum is attained: if $y = \arg\min_x f(x)$, then $f(y) = \min_x f(x)$.

The pair $\max_x f(x)$ and $\operatorname{argmax}_x f(x)$ are defined analogously. The distinction matters in optimisation: a learning algorithm reports the argmin (the parameter values), not the min (the residual loss).

B.3.2. Vector and matrix calculus

For a scalar-valued function $f : \mathbb{R}^N \to \mathbb{R}$ of a vector $\mathbf{z} = [z_1, \ldots, z_N]^T$, the *gradient* is the column vector of partial derivatives,

$$\nabla_{\mathbf{z}} f(\mathbf{z}) = \begin{bmatrix} \dfrac{\partial f}{\partial z_1} \\ \dfrac{\partial f}{\partial z_2} \\ \vdots \\ \dfrac{\partial f}{\partial z_N} \end{bmatrix} \in \mathbb{R}^N. \tag{B.25}$$

The *Hessian* is the symmetric $N \times N$ matrix of second partial derivatives,

$$\mathbf{H} = \nabla^2_{\mathbf{z}} f(\mathbf{z}), \qquad H_{ij} = \frac{\partial^2 f}{\partial z_i \, \partial z_j}. \tag{B.26}$$

At a stationary point ($\nabla_{\mathbf{z}} f = \mathbf{0}$), the function has a local minimum if $\mathbf{H}$ is positive definite, a local maximum if $\mathbf{H}$ is negative definite, and a saddle point if $\mathbf{H}$ has eigenvalues of both signs.

B.4. Chi-square distribution

Definition

Let $Z_1, Z_2, \ldots, Z_k$ be independent standard normal variables, $Z_i \sim \mathcal{N}(0,1)$. Their sum of squares

$$Q = \sum_{i=1}^{k} Z_i^2 \tag{B.27}$$

follows the *chi-square distribution* with k *degrees of freedom*, written $Q \sim \chi^2_k$. Two properties follow directly from (B.27): $Q \geq 0$, because it is a sum of squares, and the parameter k is nothing more than the number of independent squared terms being added.

The probability density function is

$$f(q;k) = \frac{1}{2^{k/2}\,\Gamma(k/2)}\, q^{k/2-1} e^{-q/2}, \qquad q > 0, \tag{B.28}$$

where $\Gamma(\cdot)$ is the gamma function. The density is right-skewed, and the skew decreases as k grows.

Moments,

$$\mathbb{E}[Q] = k, \tag{B.29}$$
$$\mathrm{Var}[Q] = 2k. \tag{B.30}$$

For independent $Q_1 \sim \chi^2_{k_1}$ and $Q_2 \sim \chi^2_{k_2}$,

$$Q_1 + Q_2 \sim \chi^2_{k_1+k_2}. \tag{B.31}$$

The degrees of freedom add exactly as the squared terms of (B.27) do. Combined with the central limit theorem, this also explains the large-k behavior: a χ^2_k variable is a sum of k independent contributions, so

$$Q \;\approx\; \mathcal{N}(k,\, 2k), \qquad k \gg 1. \tag{B.32}$$

Upper-tail quantiles

The *upper-tail quantile* $\chi^2_{k,\,1-\alpha}$ is the value that a χ^2_k variable exceeds with probability α,

$$\Pr\left(Q > \chi^2_{k,\,1-\alpha}\right) = \alpha, \qquad Q \sim \chi^2_k, \tag{B.33}$$

the subscript naming the cumulative level $1-\alpha$ rather than the tail left above it. It is computed numerically in general; two cases are closed form.

By (B.27) a χ^2_1 variable is Z^2 with $Z \sim \mathcal{N}(0,1)$, so its quantile is a two-sided normal threshold squared,

$$\chi^2_{1,\,1-\alpha} = \left[\Phi^{-1}\left(1-\alpha/2\right)\right]^2, \tag{B.34}$$

which returns $3^2 = 9$ at $\alpha = 0.0027$. For $k = 2$ the density reduces to $f(q;2) = \frac{1}{2}e^{-q/2}$, so the tail and its inverse are elementary,

$$\Pr\left(\chi^2_2 > \tau\right) = e^{-\tau/2}, \qquad \chi^2_{2,\,1-\alpha} = -2\ln\alpha. \tag{B.35}$$

Sum of squared deviations For an i.i.d. Gaussian sample $X_1, \ldots, X_n \sim \mathcal{N}(\mu, \sigma^2)$, the sum of squared deviations around the *sample* mean $\bar{X}$ satisfies

$$\frac{1}{\sigma^2}\sum_{i=1}^{n}\left(X_i - \bar{X}\right)^2 \sim \chi^2_{n-1}. \tag{B.36}$$

One degree of freedom is lost because the mean was estimated from the same data: the n deviations are constrained by $\sum_i (X_i - \bar{X}) = 0$, leaving only $n-1$ of them free. Together with $\mathrm{Var}[Q] = 2k$ this is what makes the variance of an estimated variance proportional to $2/n$.

C. Plotting

Contents

C.1. Preface **448**

C.2. A Chart That Cost a Space Vehicle **448**

C.2.1. The Story 448

C.2.2. The same data on axes 449

C.3. Why Plot at All **450**

C.4. What the Eye Does for Free **451**

C.4.1. Two ways a graph is read 455

C.5. Color **456**

C.5.1. Which part of a color can be ranked 456

C.5.2. Color and grayscale are judged in context 457

C.5.3. Colormaps 458

C.5.4. Schemes 462

C.5.5. Color deficiency 463

C.5.6. Color Meaning 466

C.6. Choosing the Encoding **468**

C.6.1. Case Study: Pie and 3D plots 469

C.6.2. Case Study: Three variables on a flat page 470

C.6.3. Case Study: Radar charts 471

C.6.4. Case Study: Plot matrix 474

C.7. Axes **479**

C.7.1. The zero baseline 479

C.7.2. Cumulative totals 480

C.7.3. Aspect ratio 481

C.7.4. Ticks and labels 483

C.7.5. Legends 484

C.7.6. Many plots at once 486

C.7.7. Close-ups 487

C.7.8. Size and gridlines 488

C.7.9. Logarithmic Axes 489

C.8. Choosing a Font **490**

C.8.1. Figure size sets the text size 490

C.8.2. The three kinds of face . . . 490
C.9. How a Reader Groups Marks . . . 492
C.10.Transparency . . . 494
C.10.1. Misuse . . . 494
C.10.2. Accumulation . . . 495
C.11.Ink That Carries No Data . . . 498
C.12.Figures Inside a Document . . . 499

This appendix collects how figures should be prepared: which visual channels a reader can actually decode, which ones only look informative, and which library defaults quietly damage the result. The rules are few, they are supported by measurement rather than taste, and at least some of them are violated by the first plot any tool produces.

Goal: Present rules and guidelines for an effective visual presentation of engineering information.

> "There are three kinds of lies: lies, damned lies, and statistics"
>
> Various Attributions

C.1. Preface

C.2. A Chart That Cost a Space Vehicle

C.2.1. The Story

On the evening of 27 January 1986, engineers at Morton Thiokol recommended against launching the Space Shuttle *Challenger* the next morning. Their concern was the temperature: the rubber O-rings sealing the joints of the solid rocket boosters stiffen in the cold, and the forecast overnight low was far below anything the program had flown in. They had the flight history to support the concern. But they did not have was a chart that clearly showed it.

Fig. C.1 is one of the charts they presented. It carries the whole damage record: twenty-four booster pairs, the O-ring temperature (°F)of each, and every erosion and blow-by incident that had been found on recovery.

Read it against the rest of this appendix and it fails on nearly every count at once.:

- The forty-eight rocket outlines are decoration: they encode nothing, and they consume most of the page (Sec. C.11).
- Temperature, the quantity the entire argument turns on, is printed as rotated text inside each drawing rather than given a position on an axis, which is the one channel a reader decodes accurately (Sec. C.6).
- Damage is encoded as unexplained hatch patterns and the letters `E`, `B` and `S`.
- The flights are ordered by booster number, so the coldest launch on record sits in the middle of the second row with nothing to distinguish it. The relation between temperature and damage is present in this chart and cannot be read from it.

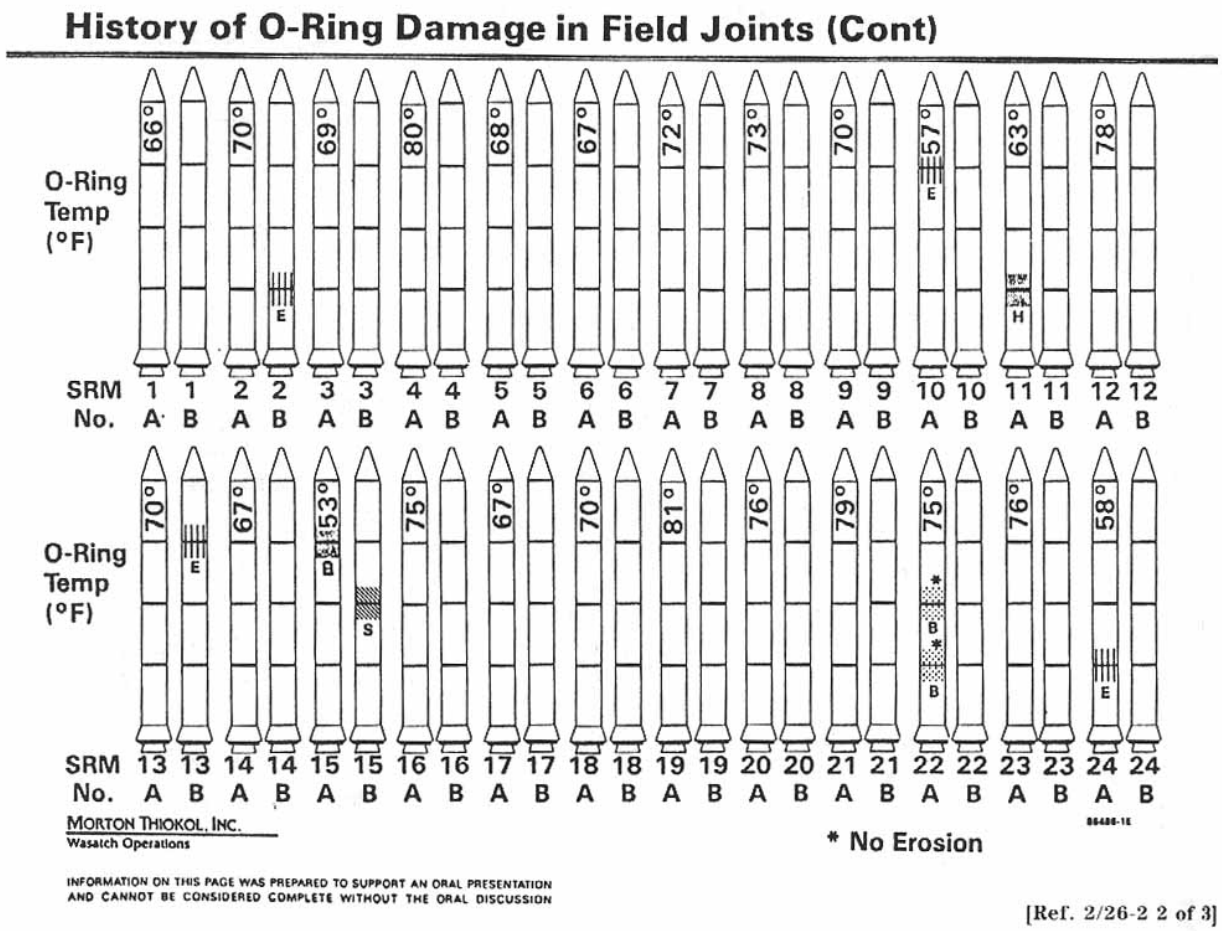


Figure C.1.: "History of O-Ring Damage in Field Joints", presented by Morton Thiokol on 27 January 1986. Every number needed for the decision is on this page. Temperature appears as rotated text inside each rocket, damage as hatch marks and single letters, and the flights run in launch order. Reproduced from the *Report of the Presidential Commission on the Space Shuttle Challenger Accident*, Vol. 5, p. 896, a work of the U.S. Government in the public domain, https://www.nasa.gov/history/rogersrep/v5p896.htm.

C.2.2. The same data on axes

Fig. C.2 plots the identical record. The horizontal axis is temperature, the vertical axis is the number of O-rings damaged on that launch, and nothing else is on the page.

The two panels differ only in which flights they contain. Panel (a) holds the flights that showed damage, which is roughly the selection the argument was built from: six points between 53 and 75 °F, no pattern, and therefore no case. Panel (b) restores the seventeen launches that came back clean. Those seventeen are not absence of evidence; each one is a launch at a known temperature that produced no damage, and they are what turns a scatter of incidents into a relation. Every launch at or below 63 °F was damaged. Only three of the nineteen above it were.

The vertical line is the forecast for the following morning, 31 °F, which is 22 °F colder than the coldest launch ever attempted.

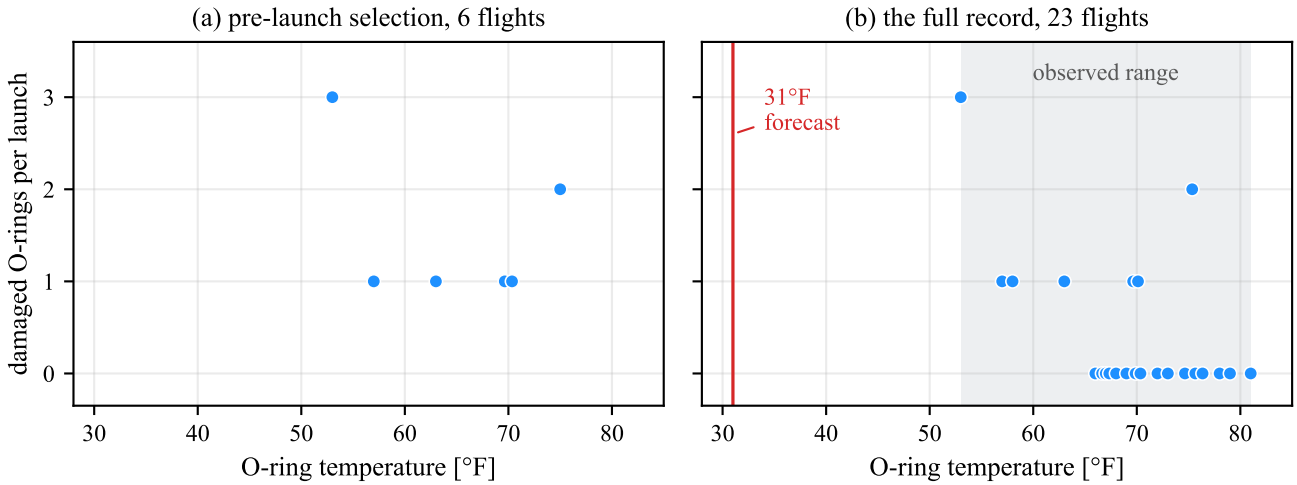


Figure C.2.: The O-ring record of the 23 flights preceding *Challenger*, plotted twice.
(a) The six damage flights that the pre-launch discussion centered on. Damage occurs from 53 to 75 °F with no visible trend, which is the reading that was available on the night.
(b) The same record with the undamaged flights restored: all four launches at or below 63 °F were damaged, while only 3 of the 19 warmer ones were. The vertical line marks the 31 °F forecast for the launch morning and the shaded band the range ever flown. Tied points are spread horizontally by up to 0.35 °F so that the four flights at 70 °F can be counted. Data: `orings` from the R package `DAAG`, after the Presidential Commission report, Vol. 1, pp. 129–131.

What the case actually shows

The engineers were right, they had the data, and they presented it. The failure was in the encoding. A chart that places the causal variable on an axis takes about a minute to make from the same numbers, and it makes the argument by itself. That is the claim this appendix defends: plotting is a technical skill with a right and a wrong answer, not a matter of presentation polish applied after the analysis is finished.

C.3. Why Plot at All

Goal: Why plot?

Fig. C.3 shows four datasets constructed by Anscombe in 1973. All four have:

- mean 9.00 in x
- mean 7.50 in y,
- variance 10.00 and 3.75 (use the $1/n$ normalization used throughout this book),
- correlation 0.816, and
- least-squares fit $\hat{y} = 3.00 + 0.50x$.

Every statistic this book has introduced for describing a bivariate relation is identical across the four panels, to two decimals.

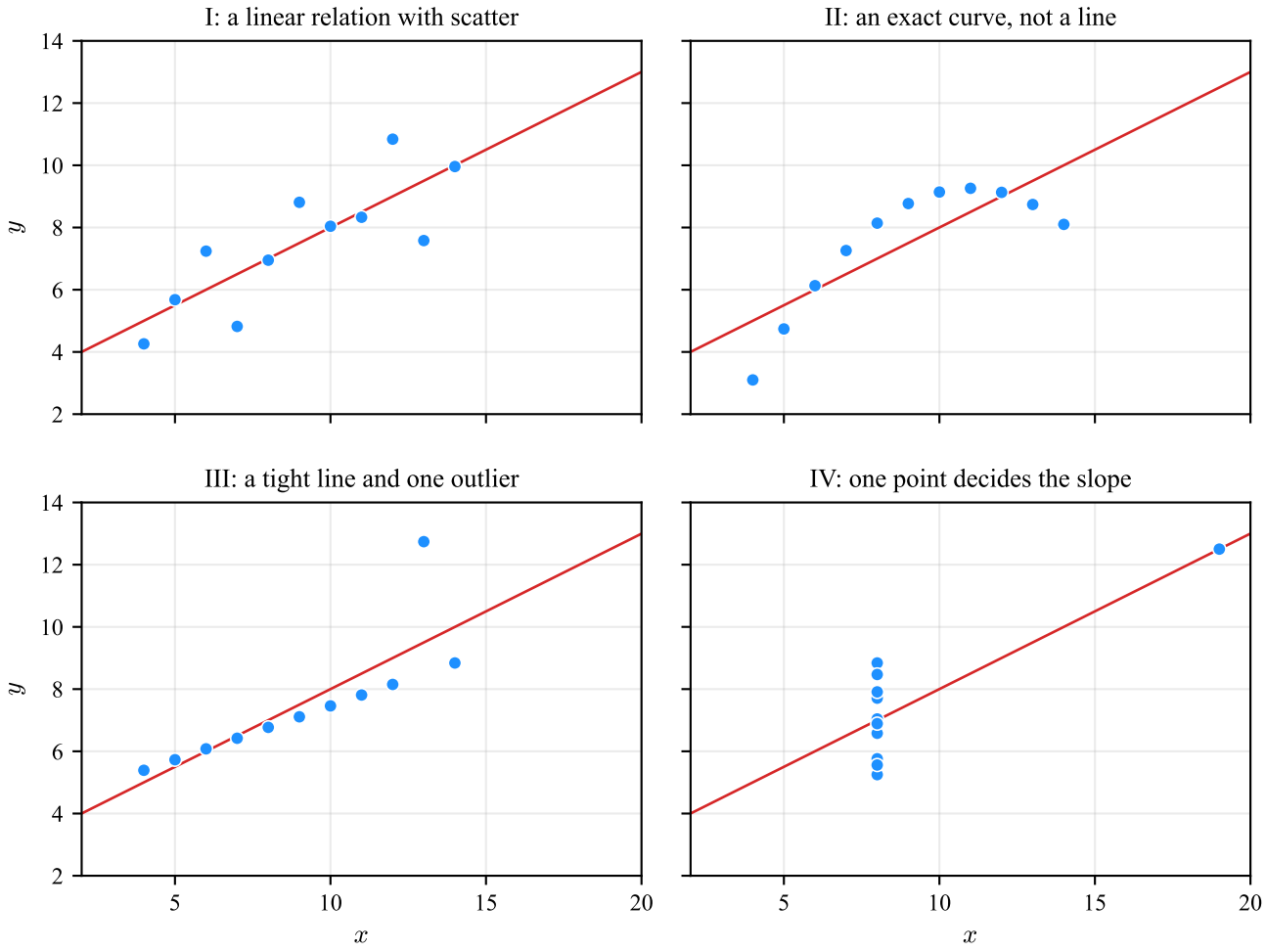


Figure C.3.: Anscombe's quartet.

This is the reason the exploratory sections of this book lead with plots. The five-number summary behind a boxplot (Sec. 1.3) is a compression, and Sec. 1.5 exists precisely because a boxplot cannot show that a distribution has two modes. The same argument runs one level up: a statistic is a compression of a plot, and a plot is a compression of the data. Each step discards structure, and the discarded structure is where the modeling errors hide.

C.4. What the Eye Does for Free

Sight carries more information per second than the other senses combined, but it does not deliver that bandwidth uniformly. Table C.1 puts numbers on the first claim: the eyes alone move about 10^7 bit/s, and the five channels together deliver roughly 11 million bit/s to the brain. Conscious processing keeps up with only about 50 bit/s, so almost everything is discarded before it reaches awareness. A chart is a device for choosing what survives that reduction, and it is effective in

proportion to how much of the reader's throughput it spends on the data rather than on decoding the chart.

Table C.1.: Information transmission rates of the five senses. The eyes carry two orders of magnitude more than the ears, and the five channels together deliver about 11 million bit/s, while conscious processing handles only about 50 bit/s. That gap of more than five orders of magnitude is the compression every figure in this appendix is trying to help the reader perform. After the *Encyclopædia Britannica* entry on information theory.

Sensory system	*Bits per second*
Eyes	10,000,000
Skin	1,000,000
Ears	100,000
Smell	100,000
Taste	1,000

Visual processing runs in two stages:

1. The first stage is *pre-attentive*: a small set of features is extracted from the entire visual field in parallel, in roughly the 200 ms between one eye movement and the next, without conscious effort and without regard to how many objects are present. The list is short and has been established experimentally: length, width, size, curvature, number, line ends, intersection, closure, color, how light or dark a mark is, flicker, direction of motion, and a few depth cues. The second stage is *serial*. It identifies objects and their arrangement, it consults memory, and its cost grows with the number of items to be examined.

Fig. C.4 shows each of these channels in isolation: in every small grid one mark differs from the rest along a single feature, and the eye lands on it without a search.

The three demonstrations of Fig. C.5 show what the difference between the two stages feels like.

The practical consequence is the third row. A reader can be asked to find the red mark, or the round mark, and will find it instantly however many distractors there are. A reader asked to find the mark that is red *and* round has to inspect the marks one at a time, and the time taken grows with their number.

Encoding one variable in color and a second in marker shape, and then asking the reader to locate a particular combination. Two channels are fine for two independent readings, and they fail for a conjunction.

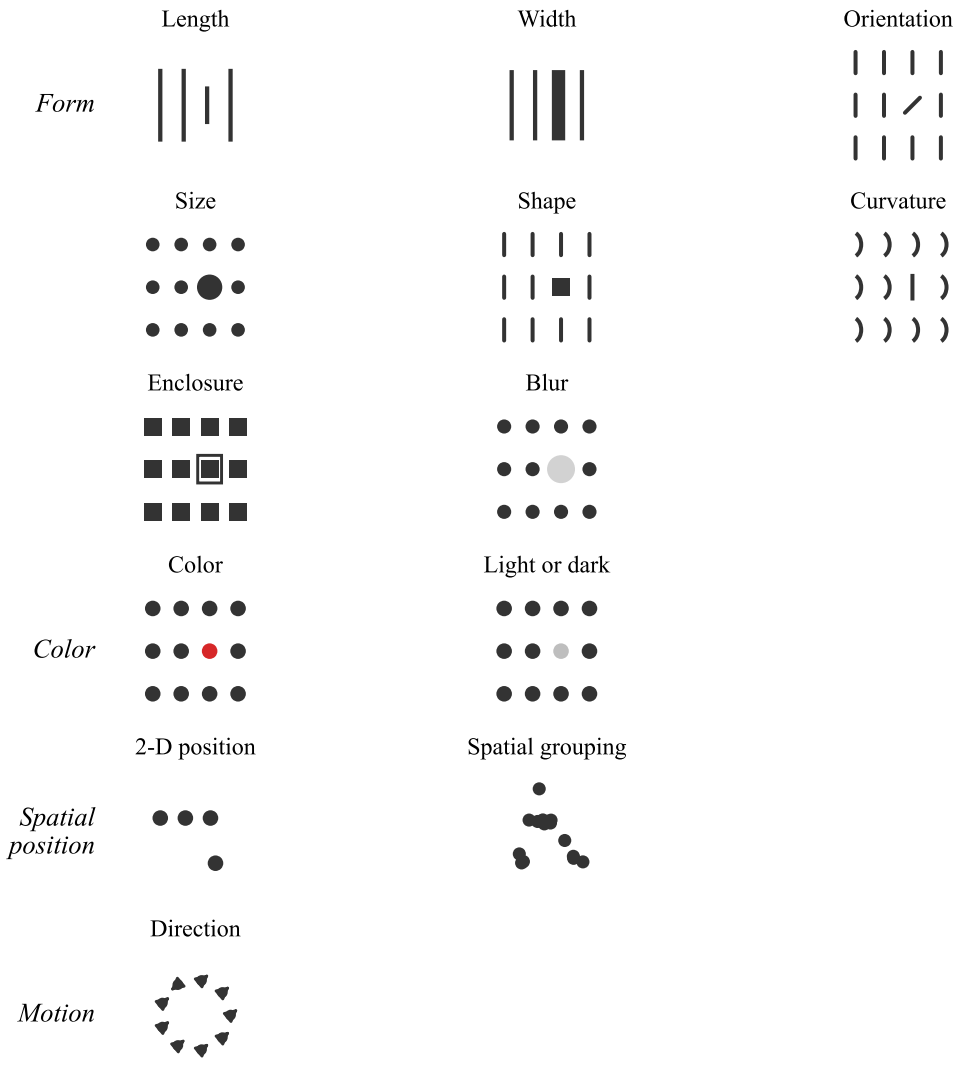


Figure C.4.: The pre-attentive attributes, one small grid each, grouped as *Form*, *Color*, *Spatial position* and *Motion*. Every grid hides a single odd element, and it is found without a search. Flicker and direction of motion are pre-attentive on the same footing but cannot be shown on a static page, so *Direction* only hints a heading through the arrows. After Few, *Show Me the Numbers* (2012), via the DLI human-perception slides.

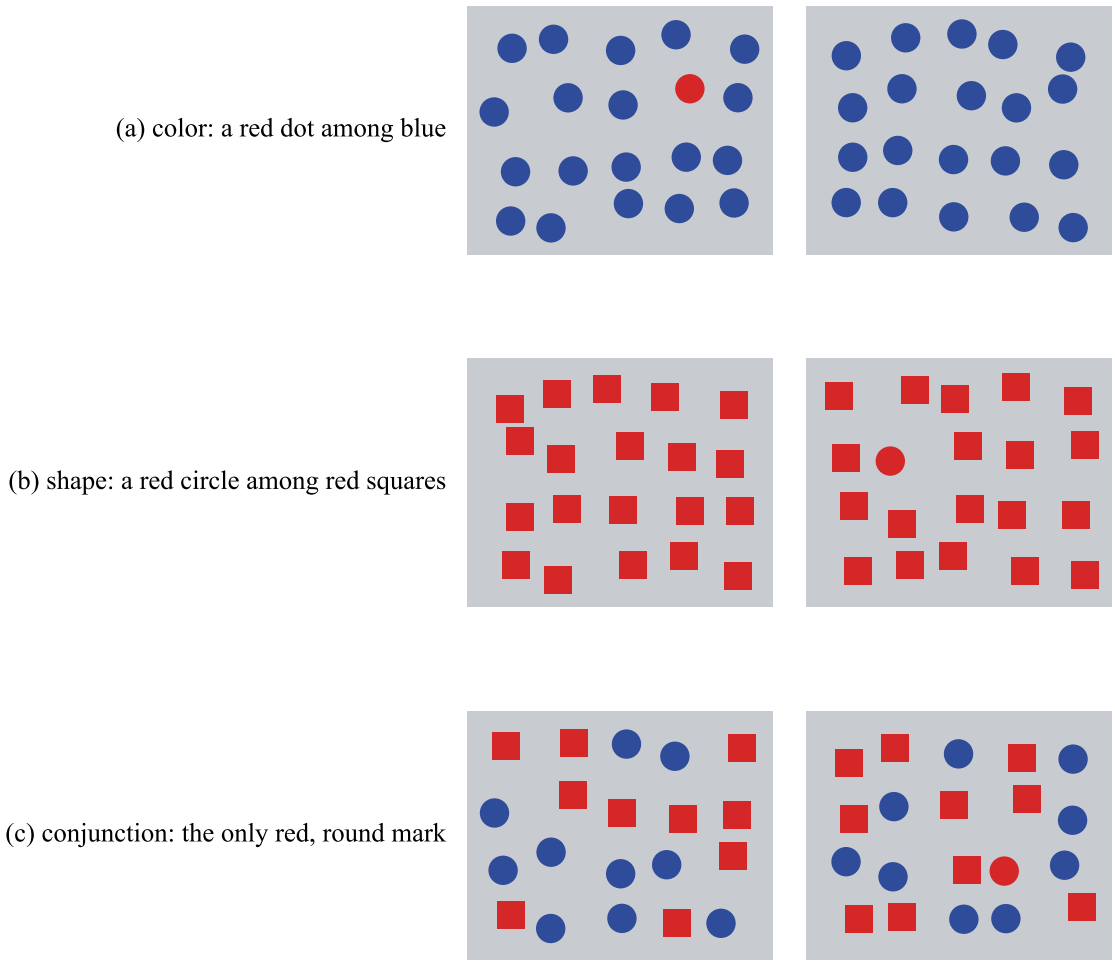


Figure C.5.: Three single-feature searches, each posed the way the source poses it: which side holds the odd mark? (a) A red dot among blue and (b) a red circle among red squares are answered before the eye has moved, however many marks are present, because color and shape are each pre-attentive. (c) The target is the only mark that is both red *and* round, among blue circles and red squares; that conjunction is not pre-attentive, so the side can be found only by checking marks one at a time.

C.4.1. Two ways a graph is read

A reader arrives at a figure with one of two questions, and the two are served by different halves of the machinery above.

- *Pattern perception* asks what the shape is: which values are large, where the group boundaries fall, whether the decay is smooth. It rides the parallel stage, so it is free and it scales, and twenty marks are taken in as fast as five.
- *Table look-up* asks for one named item and its value.[1] It is a search, so it runs in the serial stage, and its cost grows with the number of marks unless the order of the rows tells the reader where to stop.

Fig. C.6 draws one set of twenty numbers twice, changing nothing but that order.

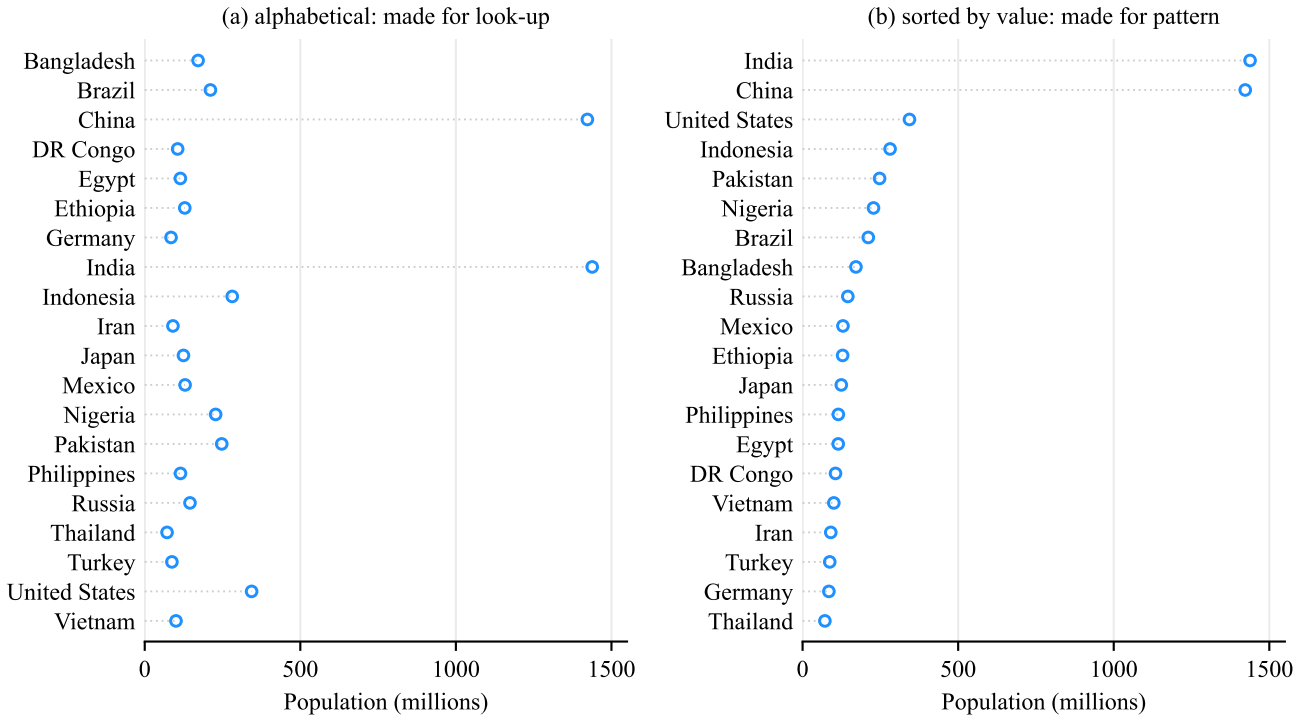


Figure C.6.: The populations of the 20 most populated countries, drawn twice with nothing changed but the order of the rows. (a) Alphabetical: Country is found in one jump, and the distribution of populations has to be assembled row by row. (b) Sorted by value; country now costs a scan of up to twenty rows. Example after Cleveland, *The Elements of Graphing Data*. Data: United Nations, *World Population Prospects*, 2024 revision, mid-2023 estimates.

Neither panel answers both questions, and the one it answers is decided by the ordering rather than by the encoding, which is identical in the two. So the ordering is not a formatting detail. Most figures in a technical document are read for pattern, which is why sorting by value is the default (Sec. C.11).

[1]The two operations are Cleveland's, and the experiments that measured the encodings behind them are Cleveland and McGill, "Graphical Perception: Theory, Experimentation, and Application to the Development of Graphical Methods", *J. American Statistical Association*, vol. 79, no. 387, 1984, pp. 531–554. That paper is what turned chart choice from taste into an experimental question: it isolates the elementary perceptual tasks a reader performs when decoding a graph, measures the accuracy of each on human subjects, and produces the ranking of Sec. C.6. The look-up and pattern example of Fig. C.6 follows Cleveland, *The Elements of Graphing Data* (1985; rev. 1994).

Figure→Table

a figure that is genuinely consulted by name wants alphabetical order; and a figure that is only ever consulted by name wants to be a table.

C.5. Color

Color buys three no other channel offers at the same price:

- it calls attention to one mark among hundreds without enlarging it or moving it,
- it gives a figure something to be remembered by,
- it adds a dimension to a page that has only two.

Color is the easiest channel to misuse

Colors always produce a picture and the picture always looks like data.

C.5.1. Which part of a color can be ranked

A color in a figure is three numbers: how much red, how much green and how much blue the screen mixes. Each runs from 0 to 1, or equivalently from 0 to 255, so every color a figure can use is a point in the box of Fig. C.7 — black at the origin where all three are 0, white at the opposite corner where all three are 1, and the grays on the diagonal between them, where the three are equal.

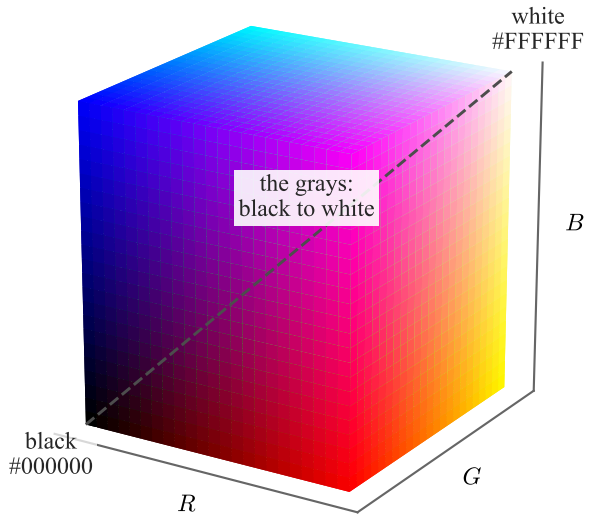


Figure C.7.: The RGB cube: every color a figure can use, as a point in the box of the three numbers. The eight corners are black, white and the six pure mixes; the dashed diagonal, where $R = G = B$, carries the grays from black to white. The two labeled corners give the three numbers in their usual written form.

Hex codes

The same three numbers are almost always written as six hexadecimal digits, two per channel, running from `00` to `FF`: `#000000` is black, `#FFFFFF` white, `#FF0000` red, `#808080` mid gray, and `#1F77B4` is the blue `matplotlib` draws its first line in. Every plotting library, style file and web page accepts this form, and it is the form in which a palette should be recorded — a color called "blue" is a different color in every tool, while `#1F77B4` is not.

Two colors written that way are two points in the cube, so how far apart they are is the straight line between them,

$$d = \sqrt{(R_1 - R_2)^2 + (G_1 - G_2)^2 + (B_1 - B_2)^2}, \tag{C.1}$$

with $d = 0$ for one color written twice and, at the other extreme, $d = \sqrt{3} \approx 1.73$ for black against white, the diagonal of the cube. Equal steps of d are not equally visible, so it is a poor measure of *how* different two colors look. What it does answer is whether they are distinct at all, which is what a palette of several colors has to guarantee.

What a figure keeps when it is printed in black and white is computable from the three numbers, $g = 0.21R + 0.72G + 0.07B$, with $g = 0$ for black and $g = 1$ for white. It is called the *gray level* here. The approximation is the quantity of interest, because it is what a black-and-white printer or a photocopier actually delivers.

Print it in gray

Any figure may be printed in black and white, projected badly, or photocopied. *Test* a figure in gray, not draw it in gray.

Dark-to-light ranking

Only dark-to-light can be ranked. Darker and lighter form a scale a reader orders without being told the rule.

Fig. C.8 is what that sentence means operationally. Eight values are encoded dark to light and shown three times: shuffled, in their true order, and as the gray a black-and-white print would keep.

An ordered quantity therefore needs an ordered channel. If the reader must rank, compare or interpolate, the value has to move the marks from dark to light; if the reader must only tell one thing from another, a different color is exactly the right answer.

C.5.2. Color and grayscale are judged in context

The eye does not measure a color, it compares one. Every patch is reported relative to what surrounds it, so the same ink can be read as two different quantities in two places on the same page. Fig. C.9 is the demonstration, and it uses no color at all.

Gray is a poor default rather than a safe one (Fig. C.9).

A shade is only comparable to another shade on the same background.

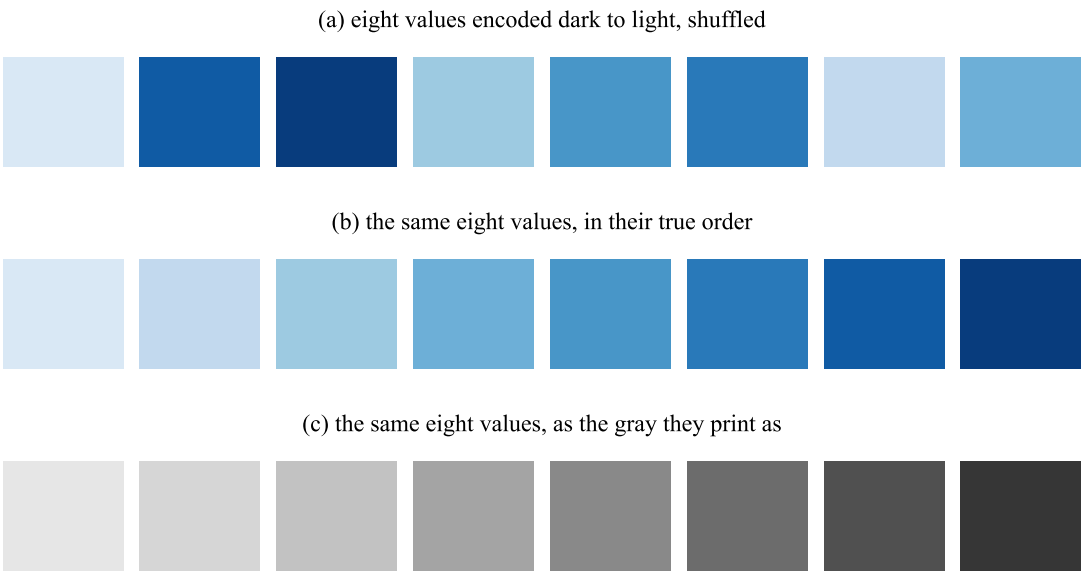


Figure C.8.: Eight values encoded dark to light. (a) Shuffled: the row can be put back in order by eye, without being told which end is large. (b) The true order. (c) The same eight values as the gray a black-and-white print keeps, falling steadily from 0.90 to 0.21. Sorting on that gray level recovers the true order exactly, which is the same operation the reader performs in (a).

C.5.3. Colormaps

Goal: What color for this number?

Colormap

A *colormap* is a function from a scalar to a color, $m : [0, 1] \to (R, G, B)$, applied after the data has been normalized onto $[0, 1]$.

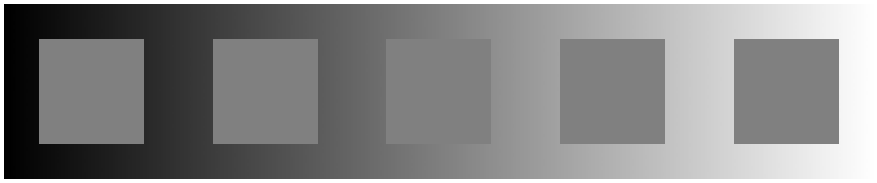
What separates one colormap from another is how the gray level g of Sec. C.5.1 runs along the map, from one end to the other, and the requirement follows from what the map is being asked to encode. A sequential map has to get lighter throughout, because the reader must be able to rank any two values it produces. A diverging map has to turn exactly once, at the value its center is pinned to, which is what makes it read as a distance from that center. A map that turns anywhere else puts an edge on the page where the data has none. Fig. C.10 shows four maps against that requirement.

The gray rows are hard to compare bar against bar, so Fig. C.11 draws the same four gray levels as curves, one against the other.

The rainbow colormaps, of which `jet` is the most common, fail this test outright. Fig. C.12 shows what the failure does to a field, by reducing each map to the gray it prints as.

The dark blob in Fig. C.12(b) is at the maximum of the field. That inversion, and the bright ring the eye reads as a boundary, are artifacts of the colormap, not features of the data. A map such as `viridis`, `magma` or `cividis` avoids them by construction: it is built so that equal steps in the data look like equal steps to the reader, and so that it never stops getting lighter.

(a) five identical squares on a ramp from black to white



(b) the same five squares on a uniform field

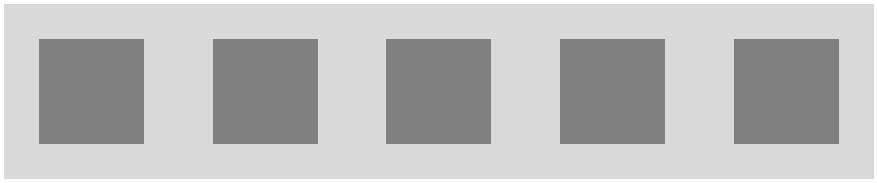


Figure C.9.: Gray in context. (a) Five squares, every one of them painted the identical RGB = $(0.50, 0.50, 0.50)$, placed along a ramp running from black to white. They appear to darken steadily from left to right, and the third square, which sits exactly where the ramp reaches its own value, disappears. (b) The same five squares on a uniform field, where they are visibly one color; the only thing that differed between them in (a) is what lay behind them, which runs from 0.10 to 0.90.

`jet` colormap Why, then, is it still everywhere? Almost never because anyone chose it.

- Was the MATLAB default until R2014b and the `matplotlib` default until version 2.0, so every script and every figure written before those releases is `jet` by omission, and copied code carries the choice forward without ever restating it.
- Instrument software keeps it alive on its own: spectrum analyzers, thermal cameras, Doppler ultrasound, and seismic and weather displays ship it, often with no setting to change it.
- When a field has trained its readers on a fixed color-to-meaning mapping, or a figure must sit beside an archive drawn the same way, changing the map breaks the comparison the figure exists to support.
- The three turns of Fig. C.11 put visible bands and edges on the image, and in color, at full brightness, that extra structure reads as detail rather than as damage; the author sees a sharp picture, and only the reader of the printed version sees Fig. C.12(b).
- Color also names a value well even though it ranks values badly. “The red region” is easy to say and easy to match against a colorbar, so reading one value off the figure feels accurate.

Inherited, not chosen

`jet` on a figure is almost always a default nobody revisited. Change it, unless the figure has to be read against an archive drawn the same way, and (optionaly) say so in the caption.

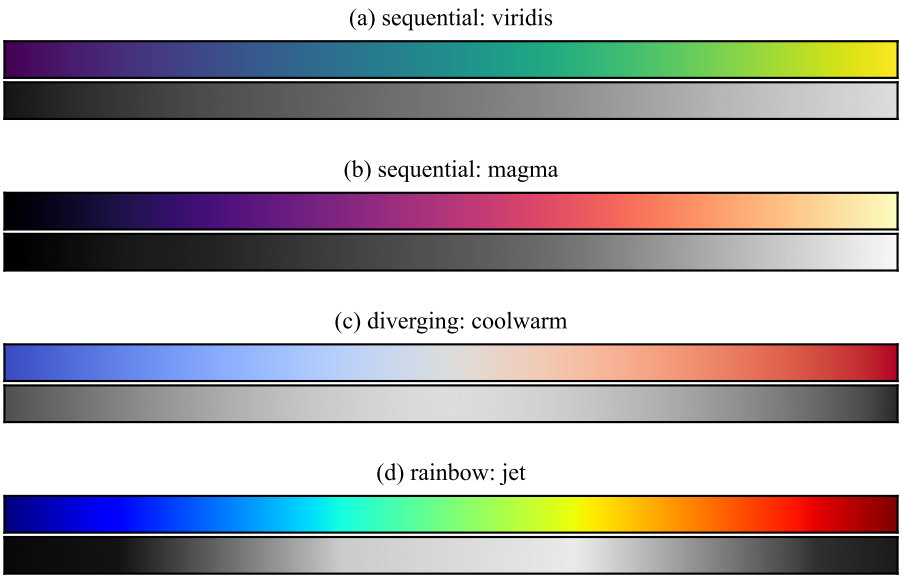


Figure C.10.: Four colormaps, each as the reader sees it and as the gray it prints as. (a) `viridis` and (b) `magma` get lighter throughout, over 0.08 to 0.87 and 0.00 to 0.97, so either can carry a magnitude. (c) `coolwarm` is lightest in the middle and darkens to both ends, which is the center it is built around and not a fault. (d) `jet` covers 0.04 to 0.92 and does neither: it is lightest near the middle, dark at both ends, and reverses on the way.

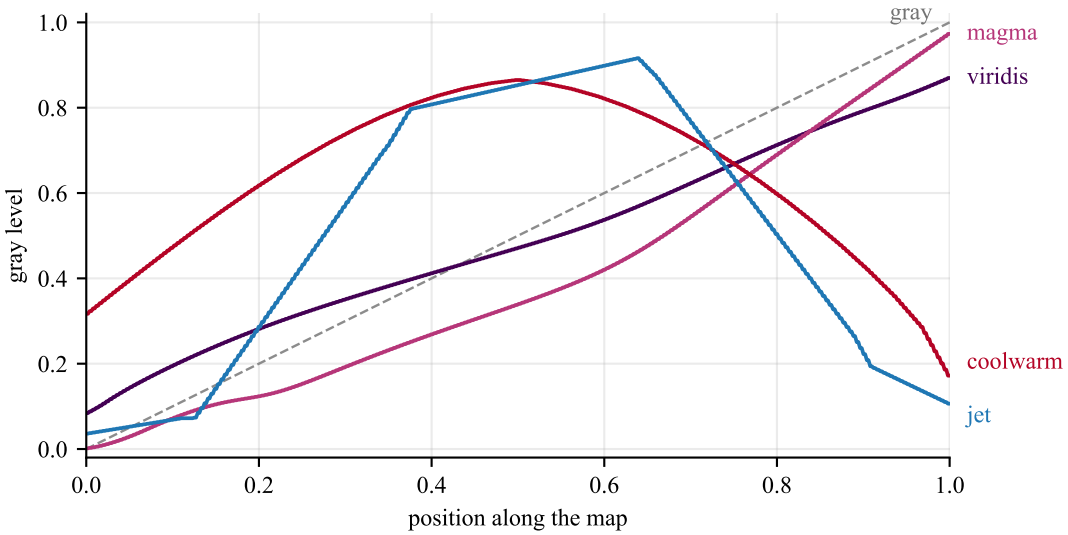


Figure C.11.: The gray level of the four maps of Fig. C.10, against position along the map, with the `gray` map itself as the dashed straight line. `viridis` and `magma` climb without turning. `coolwarm` turns once, at position 0.50. `jet` turns three times, the last at 0.64, after which it darkens again toward its own maximum. The steps are also uneven, measured as std$|\Delta g|$/mean$|\Delta g|$ along the map: 0.20 for `viridis` against 0.69 for `jet`, so equal steps in the data do not arrive as equal steps of gray on the page.

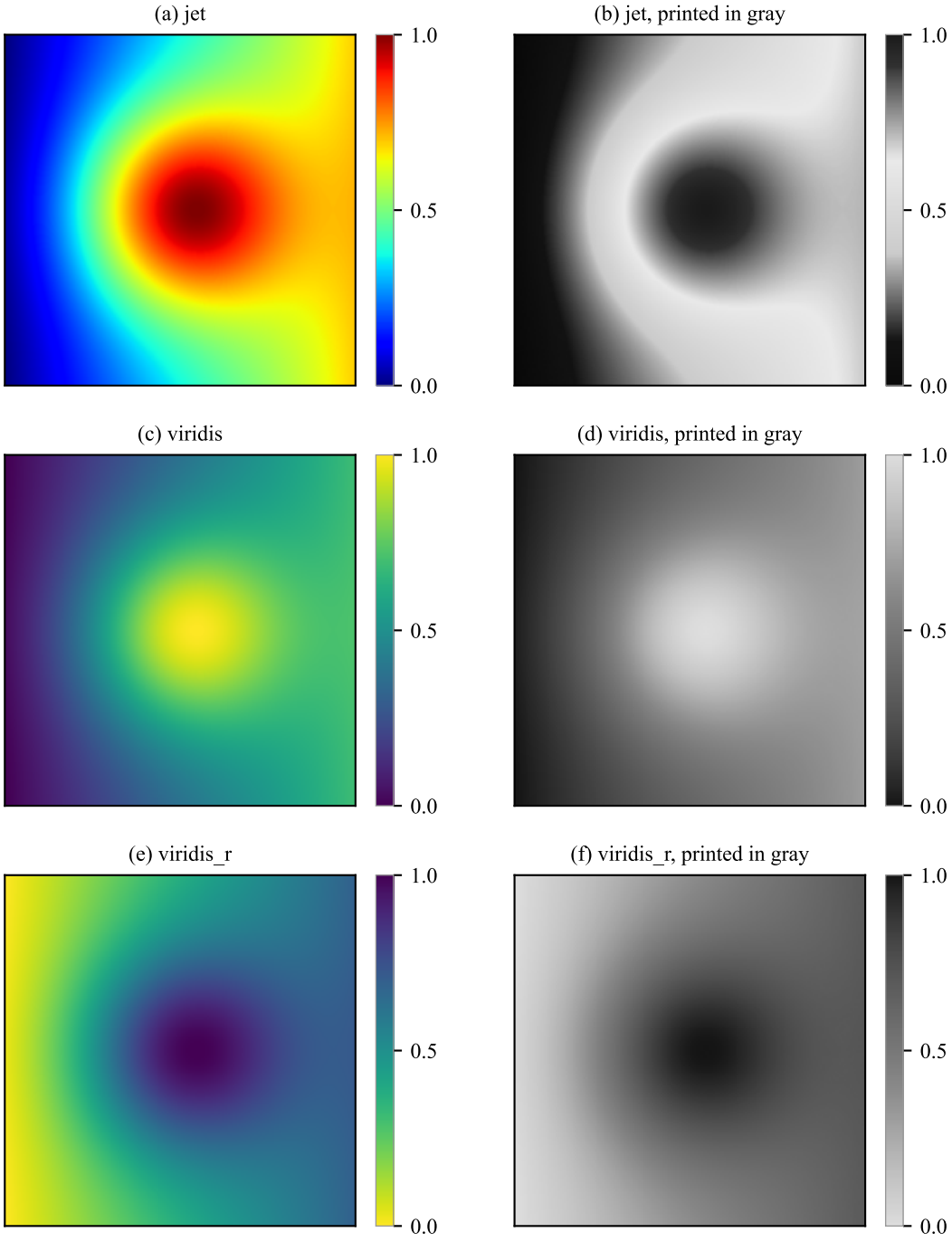


Figure C.12.: Three colormaps, each shown in color and as the gray a black-and-white print leaves. (a) and (b): `jet`, turns around three times on the way, so the peak comes out *dark* and a bright false ring appears around it. (c) and (d): `viridis`, which only ever gets lighter over the same range. (e) and (f): `viridis_r`, the same map run backwards, which never stops getting *darker* and is therefore just as rankable, with the peak now the darkest ink on the page. Every panel carries its own colorbar, over a field normalized to $[0, 1]$, and the bar of a gray panel is the gray that map prints as: the bar of (b) is dark at both ends and light in the middle, where those of (d) and (f) run one way from end to end.

C.5.4. Schemes

Goal: Which colors for these categories?

Scheme

A *scheme*, or *palette*, is a short fixed list of colors chosen as a set, $\{c_1, \ldots, c_k\}$.

Colormaps and schemes are convertible, which is why the words get swapped: sampling a colormap at k points produces a scheme, and interpolating between the entries of a scheme produces a colormap. The first direction is routine for the ordered families below, and is how most sequential schemes are obtained. The second is not, because interpolating between unordered categories produces colors that stand for nothing.

Choosing a scheme

- *Sequential*, one color running light to dark, for a quantity with a low end and a high end: counts, magnitudes, probabilities, densities. Its k colors are a sequential colormap cut into k classes.
- *Diverging*, two colors meeting at a neutral middle, for a quantity with a meaningful center: residuals and errors around zero, correlations, differences between two models. The middle must be pinned to the meaningful value, or the map asserts a center where none exists.
- *Qualitative*, different colors of similar darkness, for unordered categories: class labels, model names, cluster identities (Ch. **??**). Six to eight is the practical ceiling; past that, the colors stop being distinguishable and direct labeling is the answer.

An excellent source of schemes is Cynthia Brewer's picker at `https://colorbrewer2.org`, most of which ships with `matplotlib` under the same names.[2]

Fig. C.13 shows two schemes from each family with its gray beneath, which turns the choice from a matter of taste into a matter of shape.

The gray rows also say what each family must not be used for. A sequential map has no center to put a zero on; a diverging map applied to a quantity with no meaningful center asserts one; and a qualitative map used for a magnitude leaves the reader nothing to rank with.

[2] M. Harrower and C. A. Brewer, "ColorBrewer.org: An Online Tool for Selecting Colour Schemes for Maps", *The Cartographic Journal*, vol. 40, no. 1, 2003, pp. 27–37. The tool also filters on colorblind-safe, print-safe and photocopy-safe, which is the check of Sec. C.5.5 applied before the figure exists rather than after.

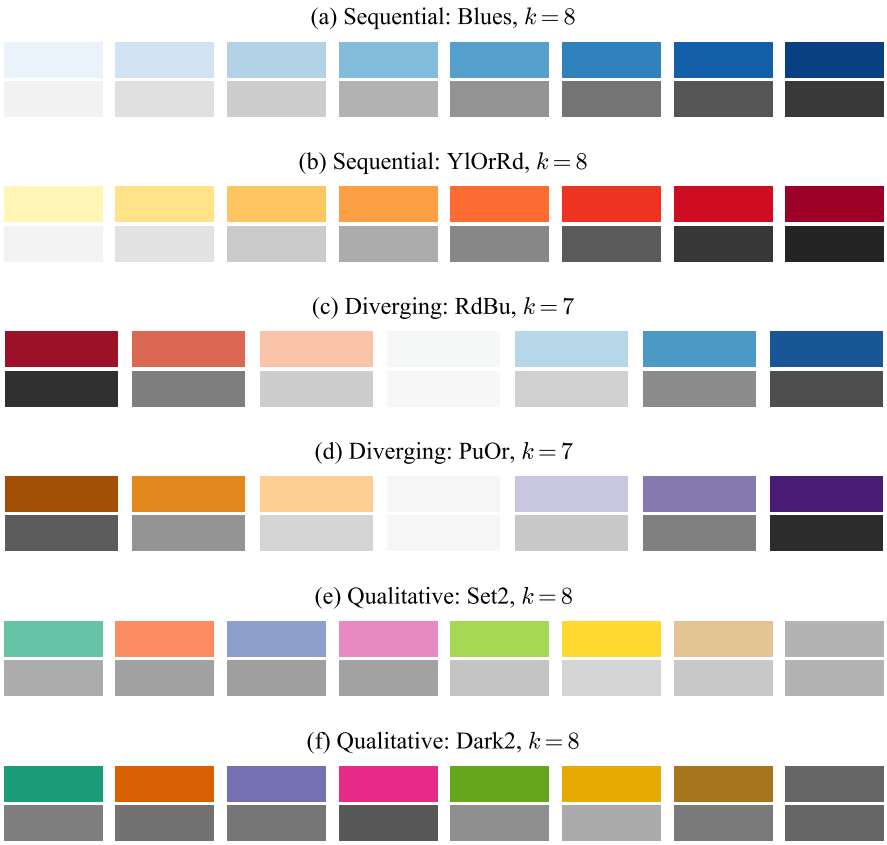


Figure C.13.: Two ColorBrewer schemes from each family, each drawn as the k fixed colors it is, in color and as the gray it prints as. (a) and (b) Sequential, $k = 8$: the gray only ever moves one way. (c) and (d) Diverging, $k = 7$, an odd count so that the neutral class is a color of its own rather than a boundary between two: the gray is lightest at that middle class, at position 0.50 of the map in both, and darkens to both ends, so the scheme reads as a distance from a center. (e) and (f) Qualitative, $k = 8$: the gray is deliberately almost flat, so the categories separate by color and none of them looks larger than another.

C.5.5. Color deficiency

The deficiency is not one condition. Fig. C.14 passes the visible spectrum through each common form, and Table C.2 gives how frequent each one is.

Red-green pair

Never let a **red-green pair** be the only thing separating two series.
Red-green deficiency affects roughly 8% of men of Northern European ancestry and 0.4% of women.

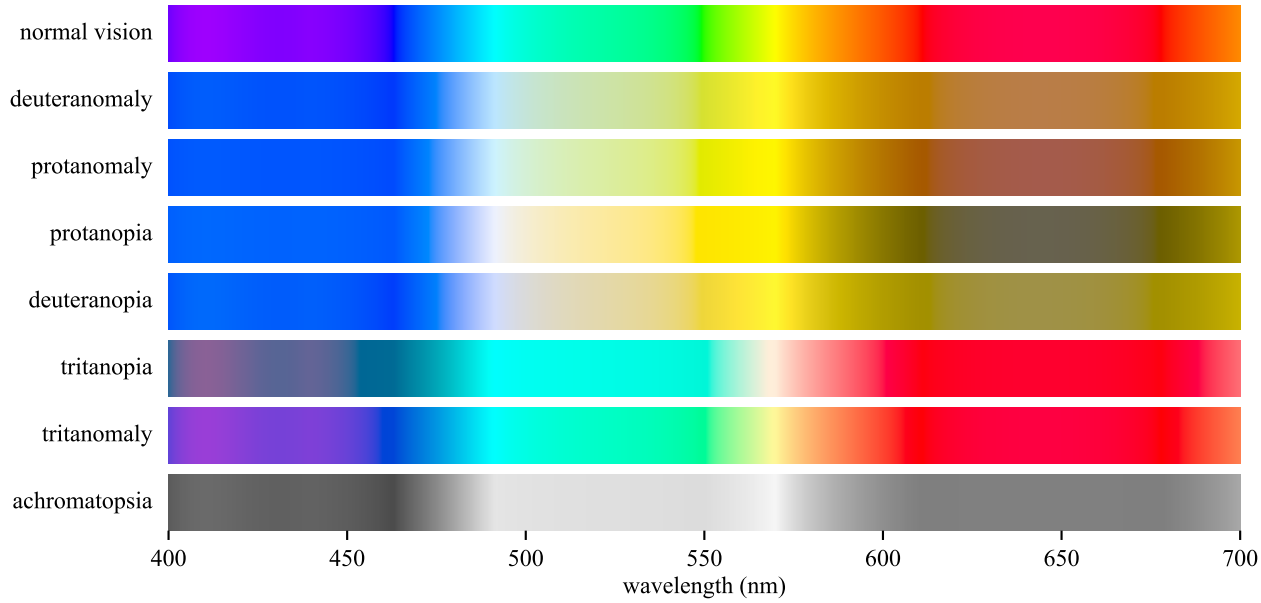


Figure C.14.: The visible spectrum as each common form of color vision deficiency receives it. The bar approximates the spectrum and is not a colorimetric reference.

Table C.2.: How common each form is, among people of Northern European ancestry, in the row order of Fig. C.14. The red-green forms of the first four rows reach about 8% of men, two orders of magnitude above the blue-yellow forms below them.

Form	Men	Women
Deuteranomaly	5%	0.35%
Protanomaly	1.3%	0.02%
Protanopia	1.3%	0.02%
Deuteranopia	1.2%	0.01%
Tritanopia	0.008%	0.008%
Tritanomaly	0.0001%	0.0001%

Fig. C.15 runs the same simulation on one chart drawn with two palettes.[3] The data, the layout and the legend are identical in the two rows, so whatever separates them is the palette alone.

The palette of the lower row is not an invention of that figure. Fig. C.16 gives it in full: eight colors chosen so that no two of them merge under any common form of deficiency, published with a name and a code for each so that a figure can be specified in them.

[3]Both figures simulate deficiency after Machado, Oliveira and Fernandes, "A Physiologically-based Model for Simulation of Color Vision Deficiency", *IEEE Trans. Visualization and Computer Graphics*, vol. 15, no. 6, 2009, pp. 1291–1298. In Fig. C.15 it is applied to the rendered pixels of the chart, so the legend keys degrade exactly as the lines do. Fig. C.14 redraws "Color blindness" by SyntaxTerror, Public domain, via Wikimedia Commons, and Table C.2 follows the epidemiology table of Wikipedia contributors, "Color blindness", *Wikipedia, The Free Encyclopedia*, revision of 11 August 2026, retrieved 18 August 2026.

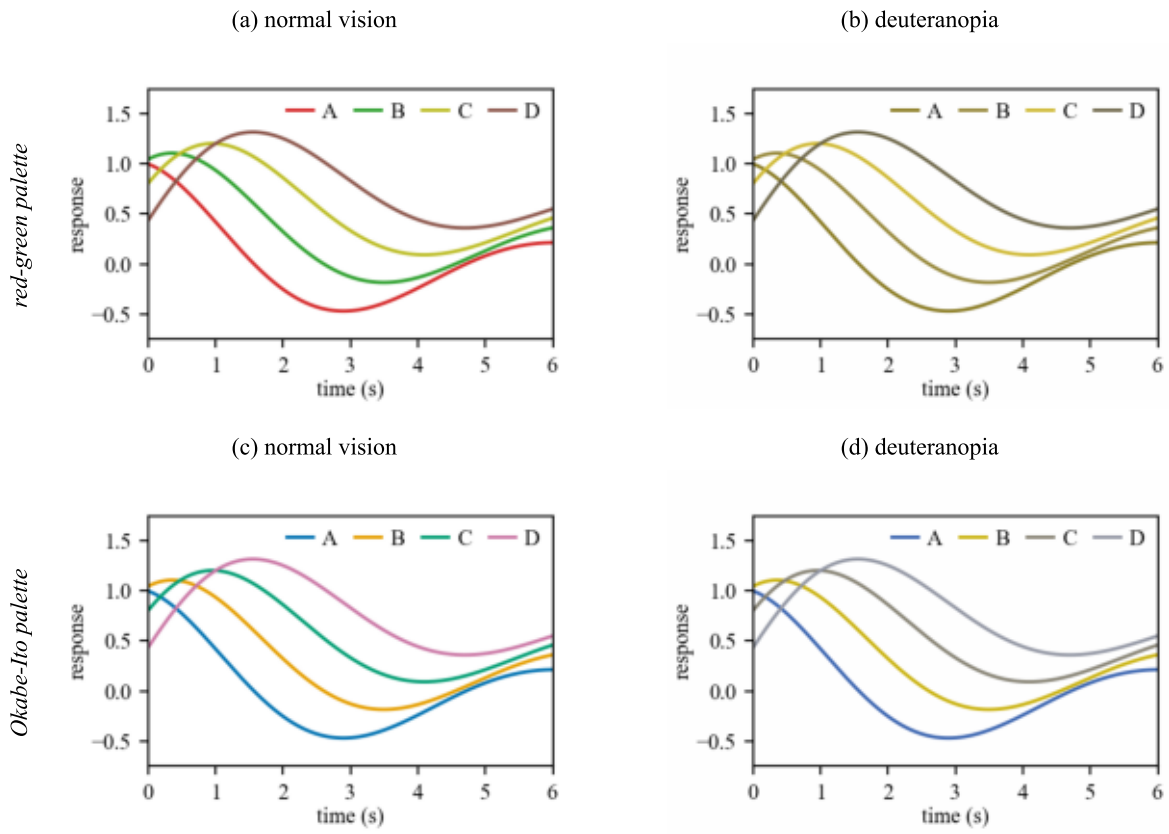


Figure C.15.: One four-series chart under two palettes, each shown in normal vision and under simulated deuteranopia. (c) *Okabe-Ito* palette. (d) all four series stay separable.

Okabe-Ito

Eight colors that survive every common deficiency, in the order they are published: `#000000`, `#E69F00`, `#56B4E9`, `#009E73`, `#F0E442`, `#0072B2`, `#D55E00`, `#CC79A7`. Take them in that order and stop when the figure has enough; four of them are Fig. C.15(c).

A figure that survives *both* black and white print and vision deficiency simulation is legible to almost everyone.

Double encoding

Use *double encoding* whenever a distinction matters: give the series different:

- line styles or markers,
- colors.

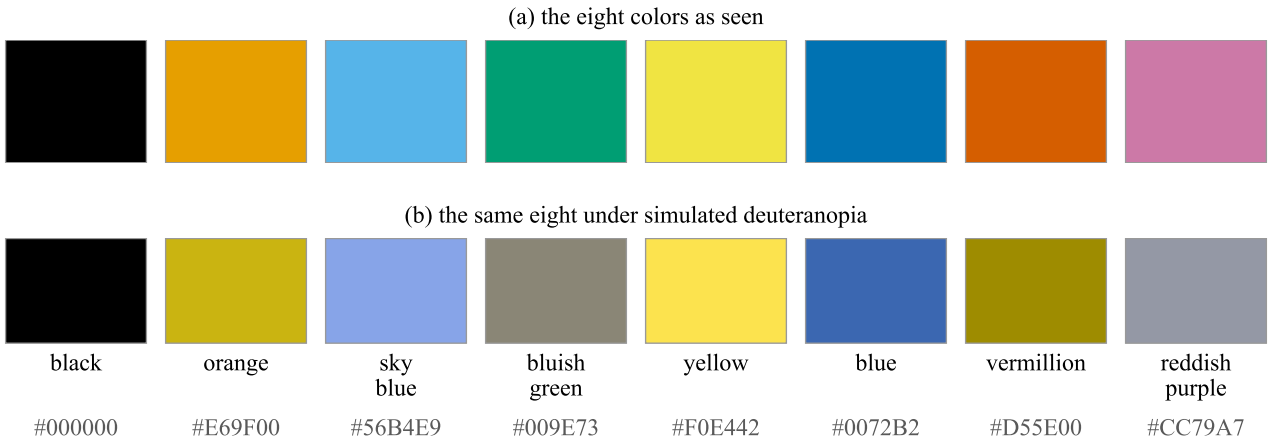


Figure C.16.: The Okabe-Ito qualitative palette, with the name and the hex code of every color. (a) As seen. (b) The same eight under simulated deuteranopia. The closest pair of (a), orange and vermillion, separates by 0.263 in RGB; the closest pair of (b), bluish green and reddish purple, still separates by 0.202, so the eight stay tellable apart. Source: Okabe and Ito, *Color Universal Design* (2008), tabulated in Wong (2011).

C.5.6. Color Meaning

Social aspects

Finally, color carries meaning that the data may not intend. Red reads as loss, danger or failure and green as the opposite, and the reader applies that mapping before reading the axis. A figure that assigns the colors the other way is not merely unhelpful, it is asserting something about every mark on the page that the numbers contradict. Fig. C.17 puts one record through both assignments.

One strong color

Strong color is a budget. Spend it on the one series, region or mark the text is about, draw everything else in gray at 20 to 40% ink, and let the reader find the subject pre-attentively, by the mechanism of Fig. C.5, without being told where to look. Fig. C.18 spends the budget both ways.

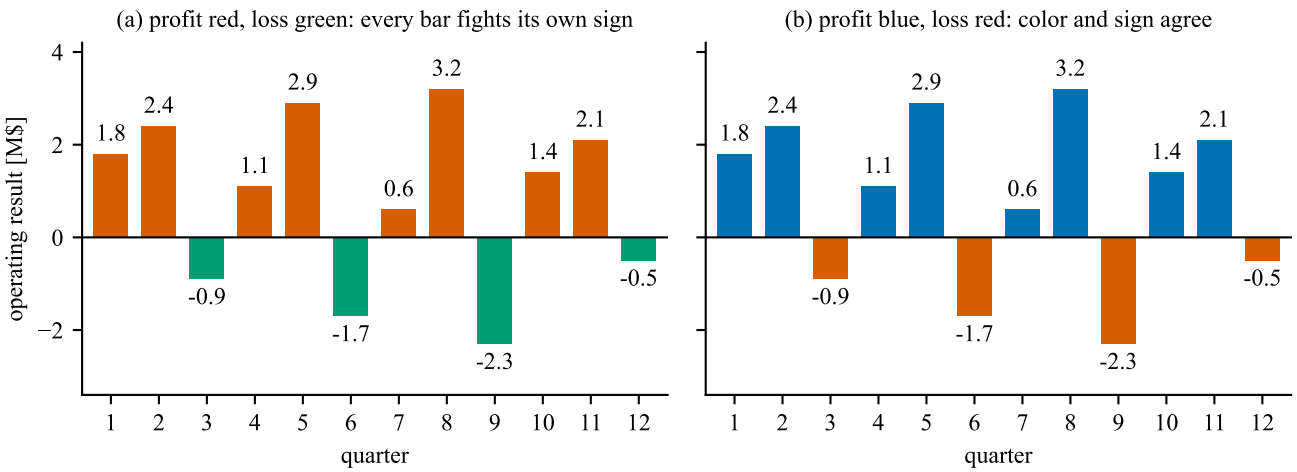


Figure C.17.: Twelve quarters of an operating result, drawn twice from the same numbers on the same zero baseline. (a) Profit in red and loss in green: all 12 bars carry a color that contradicts their own sign, 8 profits in red and 4 losses in green, and the reader has to override the color on every one of them. (b) Profit in blue and loss in red, where none does. The positive side is blue rather than green because *red-green* is the pair that roughly 8% of men cannot separate (Sec. C.5.5), so only the negative side keeps the conventional red.

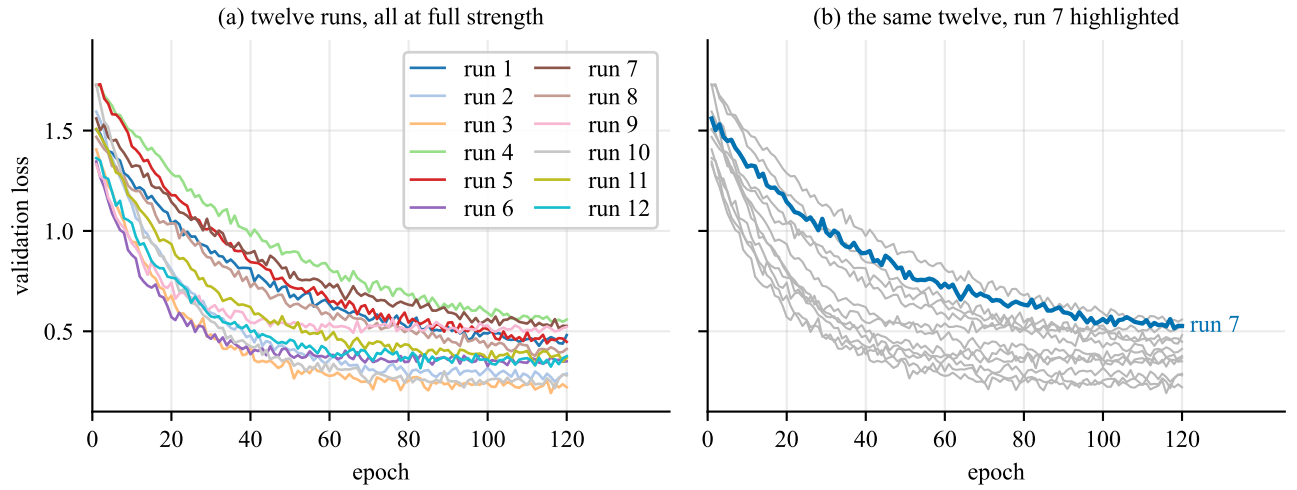


Figure C.18.: Twelve validation curves on identical axes. (a) All twelve at full strength, and 12 legend keys are needed to name them. (b) Eleven dropped to gray 0.72, which is 28% ink, against one at 0.37, or 63% ink: a contrast of 0.35 that makes the highlighted curve the only thing the eye lands on. It is named at its own end, so the panel needs no key at all.

C.6. Choosing the Encoding

A chart maps numbers onto a visual channel, and the channels are not interchangeable. Their accuracy has been measured. The ranking is stable, and it is the most useful result in the field:

1. position along a common scale;
2. length;
3. slope and angle;
4. area;
5. volume;
6. shading and color strength;
7. a different color.

Fig. C.19 puts one set of five values through the first six of these channels, one panel per entry, from panel (a) for the first to panel (f) for the sixth, so the ranking can be felt rather than taken on trust. The seventh has no panel here because it has no answer: a set of colors is a set of names, and names do not come in an order, so the ranking cannot be recovered at all. Only the dark-to-light part of a color can be ranked, which is the subject of Sec. C.5.1.

Three consequences follow, and they account for most of the chart-type advice in circulation.

Bar, line and scatter

Bar, line and scatter are the workhorses. They occupy the top of the ranking. There is rarely a reason to reach past them for quantitative data.

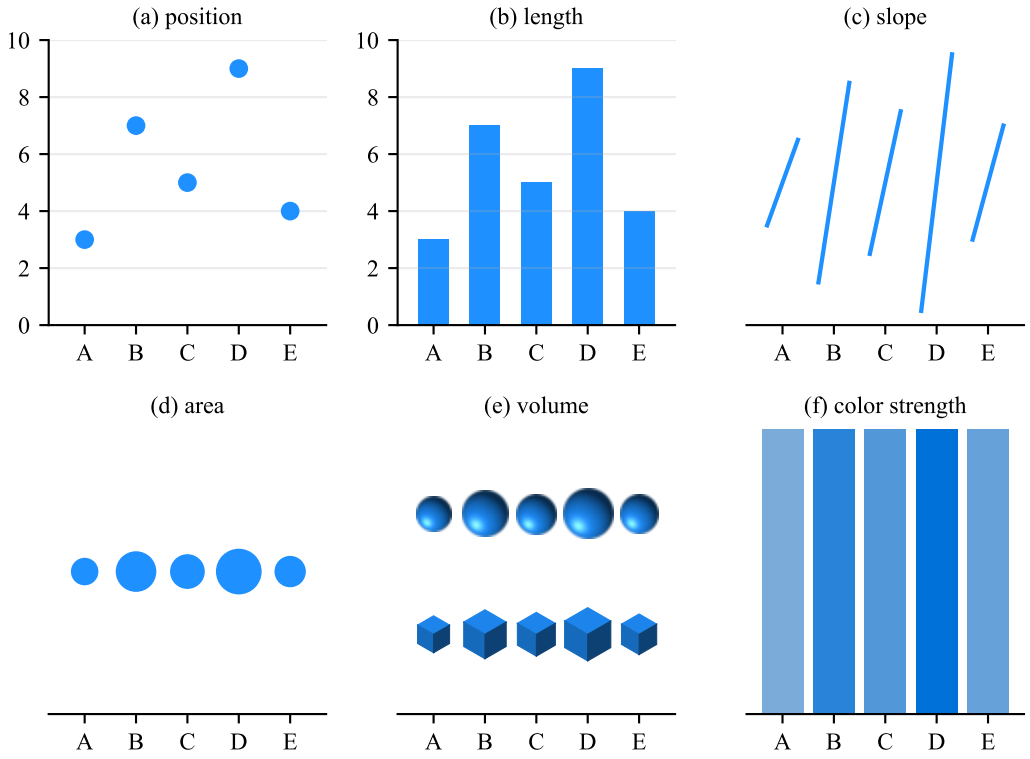


Figure C.19.: Five values, A to E, encoded six ways, one panel per entry of the ranking above. Rank the categories from smallest to largest in each panel. The answer is $A < E < C < B < D$ throughout. Panels (a) and (b) give it up immediately. Panels (c) to (e) take effort and invite errors between the close pair B and C, and the two size channels differ in how much of the difference survives: D is three times A, and its mark has three times the area in (d) and only 2.1 times in (e), where the value is a volume, so the radius of the ball and the edge of the cube both follow its cube root. Panel (f) cannot be done reliably at all.

C.6.1. Case Study: Pie and 3D plots

Pie charts encode angle and area Both sit low in the ranking, which is why comparing two slices of similar size is guesswork, and why a pie with more than three slices cannot be ordered by eye. The same numbers as a sorted bar chart are read exactly, in the same space.

Three dimensions on a flat page cost accuracy and buy nothing A 3D bar chart replaces length, which is second in the ranking, with volume under projection, which is fifth, and adds occlusion so that some bars hide others. Adding perspective makes it worse than guesswork: tilting the pie shrinks the back slices and enlarges the front ones, so the picture is no longer even a faithful rendering of the angles.

Both paragraphs above can be checked on one set of numbers. Fig. C.20 takes the five values 37, 36, 24, 2 and 1 through the four charts a spreadsheet offers for them. The two largest differ by one part in a hundred, and that difference is the test: every panel contains it, and one panel shows it.

Panel (d) is not a more careful version of the others, it is a different encoding. It reads the values as length from a common baseline, second in the ranking, while (a) and (b) read them as angle and

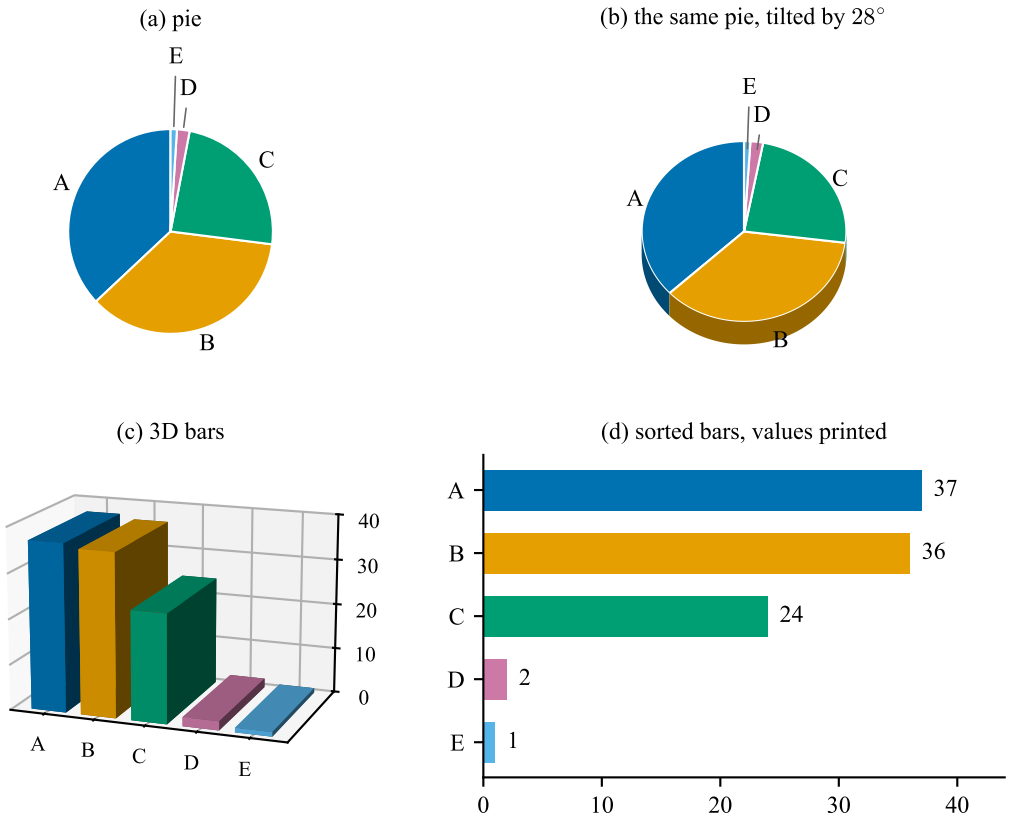


Figure C.20.: Five values, $A = 37$, $B = 36$, $C = 24$, $D = 2$, $E = 1$, drawn four ways. (a) A pie: A and B differ by 3.6° of arc out of 360°, so the two cannot be ordered, and the slivers D and E need leaders to be labeled at all. (b) The same pie tilted by 28°: B lies at the front and collects the slab wall, so it is drawn with 1.27 times the apparent area of A although it is the smaller value. (c) The same values as 3D bars: the front bar tops reads between 7 and 11 units low, which distorts the differences as well as the values. (d) Sorted bars with the values printed: the ranking is immediate and the numbers are exact.

area, third and fourth, and (c) reads them as a height that has to be carried across a gap in depth before it meets a scale.

C.6.2. Case Study: Three variables on a flat page

Two variables get the two axes. A third has to go into a channel, and which channel is the whole question, because the ranking above is the answer to it. Fig. C.21 takes one record of forty training runs, each with a model size, a training set size and the accuracy it reached, and moves the training set size through four channels, one per panel. The reader is meant to take off the page that model size stops paying past about ten million parameters and training set size does not, so which of the two is worth the next budget.

The four panels are four entries of the ranking, and they come out in its order.

- Panel (a) is not on the ranking at all: a projected point has no depth to decode, which is the argument of Sec. C.6.1 moved from a pie to a scatter.

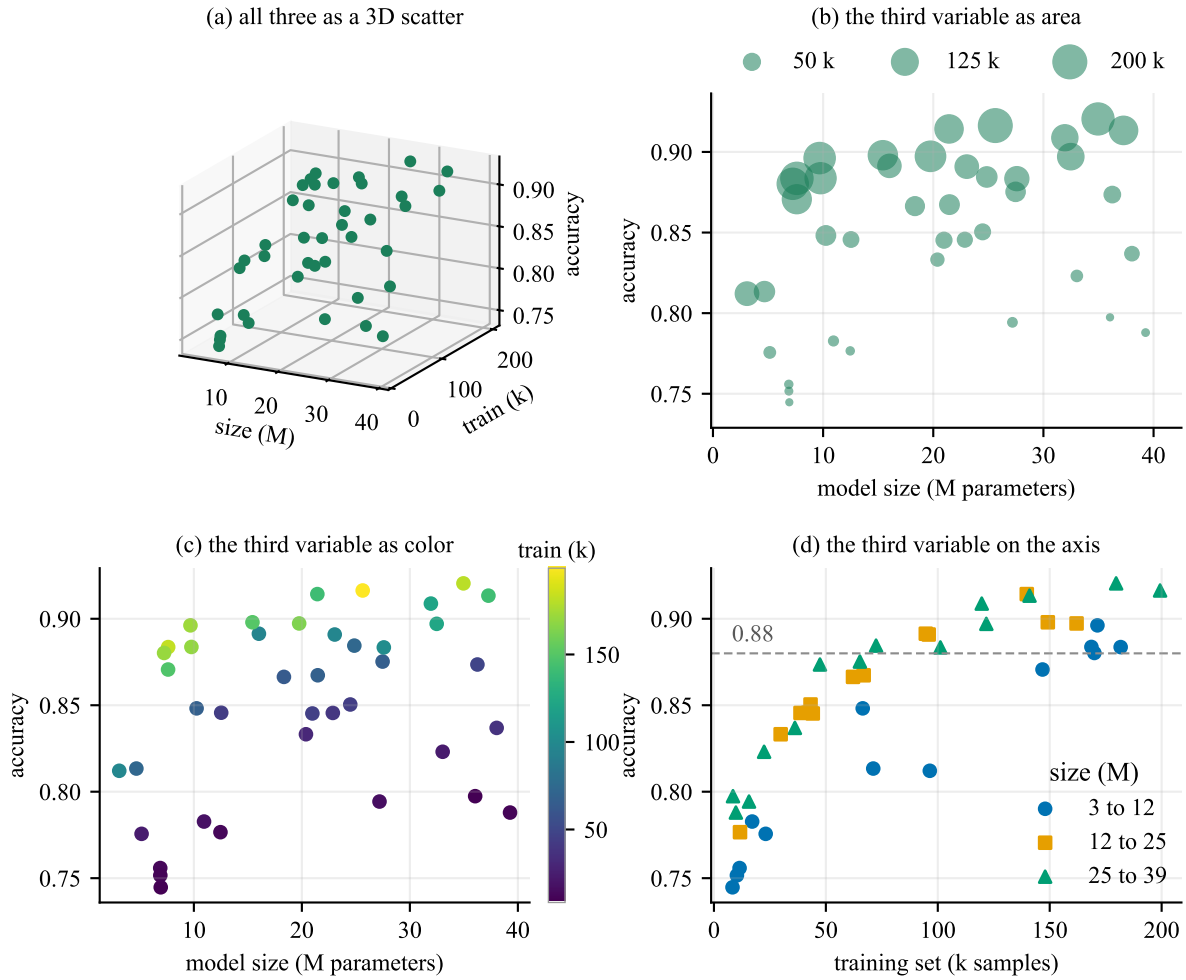


Figure C.21.: Three variables. (a) All three as a 3D scatter. (b) The training set as marker area. (c) The training set as color, on `viridis` with a colorbar. (d) The training set on the horizontal axis, model size as three bins in color and marker shape at once.

- Panel (b) reads it as area, fourth.
- Panel (c) reads it as color strength, sixth, which supports *more* and *less* and stops there.
- Panel (d) reads the third variable as position on a common scale, first, which is why the number the question turns on is read off an axis instead of estimated.

Note (d) carries model size in two channels at once, color and marker shape.

C.6.3. Case Study: Radar charts

Radar chart

A *radar chart*, also called a spider or a web chart, places k categories at k equally spaced angles, draws each value as a radius along its own spoke, and joins the ends into a closed polygon. The angle is a *slot* rather than a quantity, which is what separates it from a polar plot: there the angle carries a direction, a phase or a time of day, and none of what follows applies.

The form has one honest use and it is narrow: a small number of profiles, measured on axes that genuinely share a scale, read for their shape rather than for their values. Every attribute is scored by the same panel on the same scale, so the radial axis means one thing on every spoke. Fig. C.22 takes one such record, four samples scored on six attributes, draws it in the form the chart usually takes, and measures beside it what that form does to the levels.

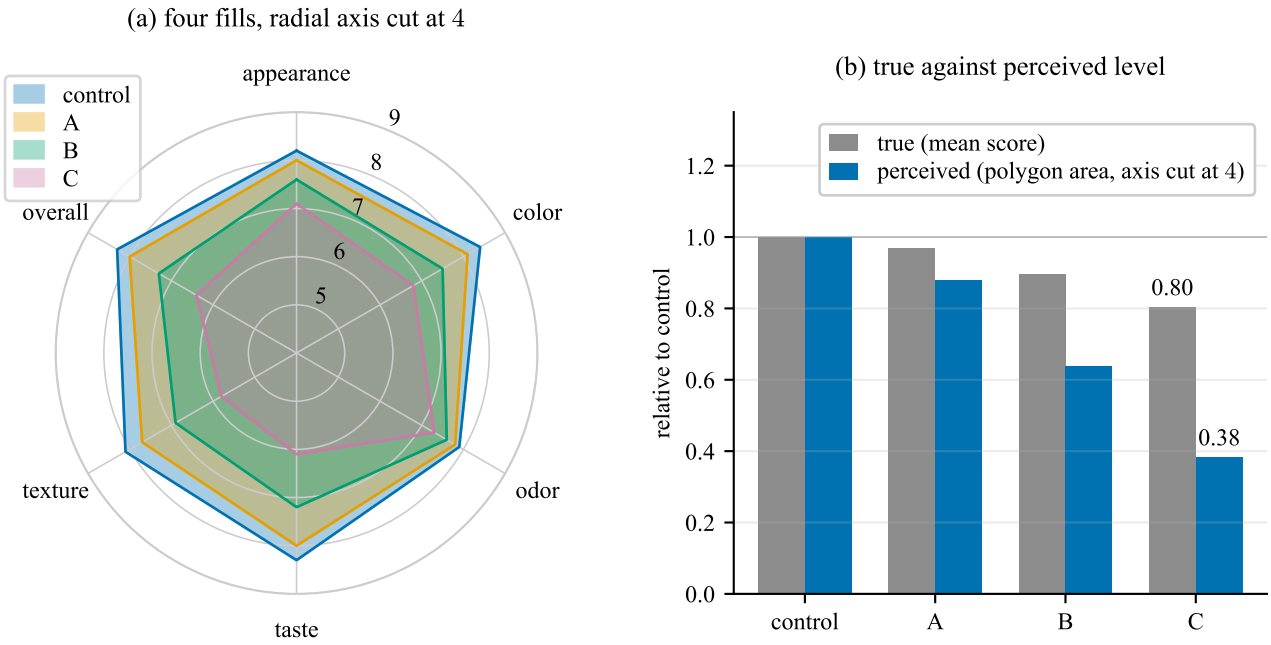


Figure C.22.: One synthetic sensory record: four samples, six attributes, every attribute scored on the same 9-point scale, drawn as a radar chart and measured against it. (a) All four filled at $\alpha = 0.35$ on a radial axis starting at 4. The cut removes the inner 44 % of every radius and takes the apparent area of sample C; the fills occlude one another in the order they were drawn. (b) The cost of that encoding, as a length: for each sample, the true level, its mean score relative to the control, beside the level the polygon area of panel (a) makes the reader perceive, both on a zero baseline.

Panel (b) measures what panel (a) costs. The gray bar is what a sample scored, the blue bar is the level its polygon in panel (a) makes the reader perceive. The gap between the two is contributed by the encoding rather than by the data: nothing on panel (a) discloses that they differ. The gap widens with the drop, from 0.09 for sample A to 0.42 for sample C, so the chart is least trustworthy exactly where the finding is.

What panel (a) does wrong follows from what the polygon is. Its area on k equally spaced spokes with radii $r_1, \dots, r_k$ is

$$A_{\text{poly}} = \frac{1}{2} \sin\!\left(\frac{2\pi}{k}\right) \sum_{i=1}^{k} r_i \, r_{i+1}, \qquad r_{k+1} = r_1, \tag{C.2}$$

and two properties of that expression decide how the chart reads.

The area is quadratic in the values. The eye compares the filled shapes and not the radii. Sample C scores 0.803 of the control level and its polygon covers 0.646 of the control area, which overstates the drop by a factor of 1.80; that comparison is the pair of bars panel (b) draws. That is the exaggeration of a cut bar chart (Sec. C.7) in a form no axis label discloses.

The area depends on the order of the spokes. Eq. (C.2) pairs adjacent radii only, so permuting the axes changes it without changing a number. On this record the effect is small. The ordering is a choice the author makes and the reader cannot see.

The fix for panel (a) is not a better radar chart. The same 24 numbers as a dot plot, one row per attribute and one marker per sample on a shared horizontal scale, drawn in Fig. C.23(b), put all of them at the top of the ranking of Sec. C.6: the largest difference in the record, 2.3 points of texture between the control and sample C, becomes a length rather than a shape. It also settles the comparison the radar chart cannot serve at all. Two attributes of one sample sit on one scale in a dot plot; on a radar chart they sit in two different directions.

If you draw one

One zero-anchored scale on every spoke, and name it. Two profiles at most, or one per panel as small multiples (Sec. C.7.6). Fill one polygon or none. And say what fixes the axis order, because Eq. (C.2) says the picture depends on it.

Fig. C.23(a) is that box applied to the record of Fig. C.22, and panel (b) is the dot plot beside it. Panel (a) is as good as the form gets, and it is still the weaker of the two. Anchoring the radial axis at zero fixes the baseline and nothing else: by Eq. (C.2) the area stays quadratic in the scores, so sample C keeps 0.803 of the control level and 0.646 of the area, the same factor of 1.80 as before; the spoke order still moves the picture; and the finding of the record is a difference between two radii in (a) where panel (b) makes it a length on one scale.

Do not normalize the axes into agreement

Do not normalize the axes into agreement.

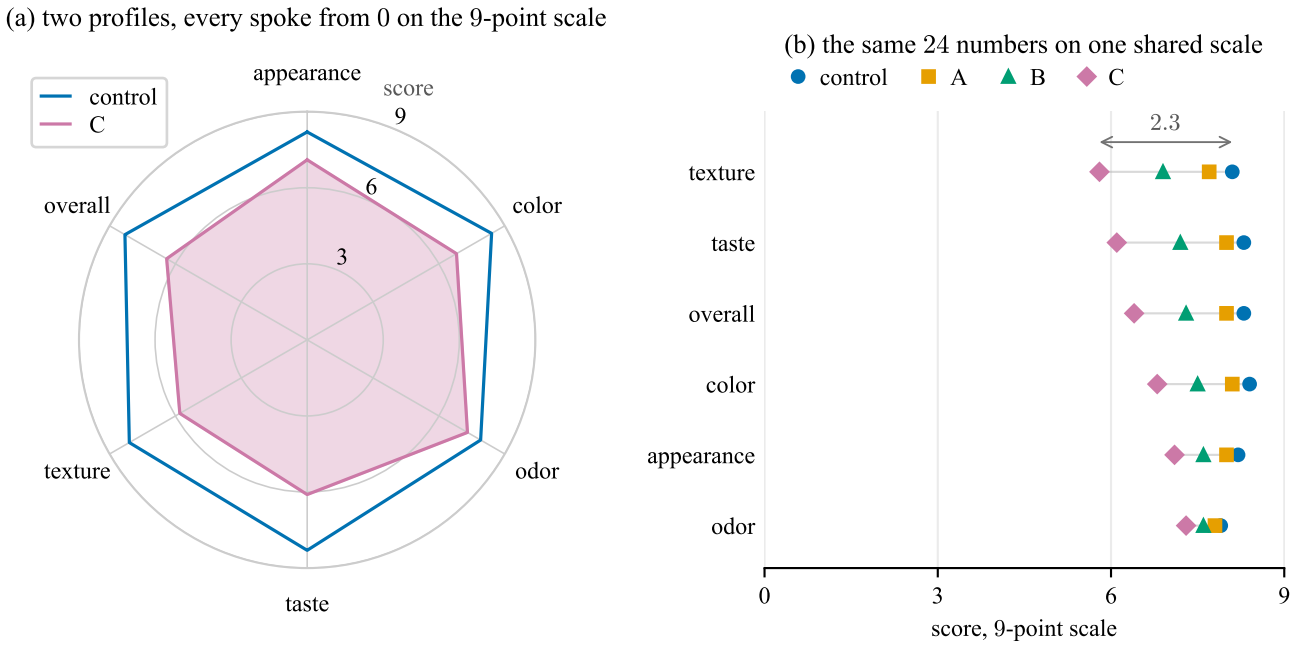


Figure C.23.: The record of Fig. C.22, drawn under the rules of the box and then not drawn as a radar chart at all. (a) Two profiles rather than four, every spoke starting at 0 on the named 9-point scale, one polygon filled, and the spokes in the order the questionnaire asks them, which is the order that is not chosen for how the polygon comes out. (b) The same 24 numbers as a dot plot, one row per attribute, sorted by the gap against the control, the sample carried by color and marker shape at once so that the panel survives a gray print and a deficiency simulation (Sec. C.5.5). The finding, 2.3 points of texture between the control and sample C, is the arrow in (b) and a comparison of two directions in (a).

C.6.4. Case Study: Plot matrix

A *relation* is an $n \times n$ table whose entry (i, j) describes the *pair* rather than a single item: confusion matrices, correlation and distance matrices, and more. The default rendering for all of them is a colored grid, which encodes the value as shading, sixth in the ranking above.

Goal: Draw a relation for the comparison it was measured to make. Here cell (i, j) is the accuracy of a classifier trained on channel i and tested on channel j, and the comparison is against the target channel's own model, on the diagonal.

Fig. C.24(a) is the grid as its script produced it, from the single line `imshow(M, cmap='viridis', vmin=M.min(), vmax=M.max())`. The map is stretched across the observed range, 0.875 to 0.926, so the entire excursion from deepest purple to full yellow is 5.1 points of accuracy, and no colorbar reports it.

Colorbar

A colormap fitted to the data range, with no colorbar to disclose it, is an error to fix.

Scaling

- `vmin=M.min()` is the cut baseline of Fig. C.27(a) moved into color: the scale is pinned to the data rather than to a meaningful zero, so every apparent difference is inflated by an amount the reader cannot detect. Subtract the reference before you color it!
- Switch the text color on the gray level of the cell rather than on the value.
- Do not re-encode the diagonal, which position already identifies; define every glyph, since an asterisk without a key is a legend without a key.
- The ordering rule of Sec. C.4.1.

The comparison the grid is being asked to support is between each transfer and the target channel's own model, and there are two ways to make it: show that reference, or subtract it. Panels (b) and (c) show it. They keep the accuracies exactly as they were measured and mark each target's own model beside them, so the shortfall is a distance on the page rather than a number the script arrived at, and the level of the results survives. That level is what ch 3's row is for: a model transferred into ch 3 is the least accurate destination of the four, 0.8772 against 0.8928 into ch 2.

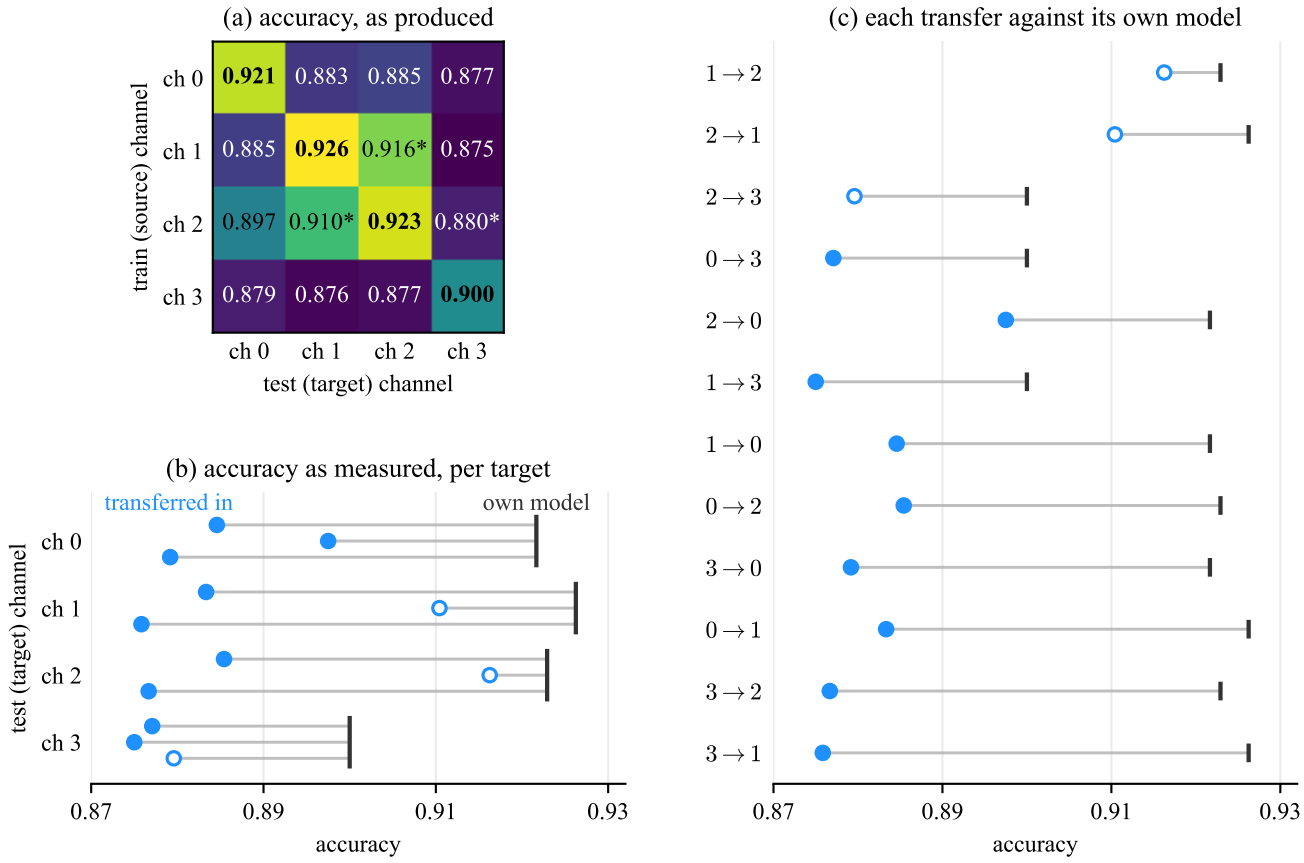


Figure C.24.: One 4 × 4 cross-channel record, as it was measured. (a) As produced: `viridis` stretched over the observed range 0.875 to 0.926, no colorbar, the diagonal in bold, an unkeyed asterisk. (b) One row per target channel, three dots in it, one per model transferred in, each joined to the dark rule of that channel's own model. The segment is the loss the next figure computes, here a length on the page and nothing else. Ch 3's whole row, rule included, sits left of the others. (c) The same twelve, sorted and named; the open markers are the three closest transfers. The axis is cut, which points allow and bars do not.

The two panels are not equally successful, and the reason is the reference. Panel (b) draws each of the four references once, so the levels are visible as levels and ch 3's rule at 0.900, against 0.921 to 0.926 for the others, is the finding. Panel (c) draws the same four references twelve times, in whatever order the sort produces; and by sorting on the segment it turns the shortfall into a length measured from twelve different starting points. Sort by a difference only when the marks that carry it start together, which is what a common zero gives and what the next figure draws.

Subtracting the reference is the other way. The quantity the argument turns on is then not the accuracy but the loss against that reference,

$$\Delta_{ij} = a_{jj} - a_{ij}, \tag{C.3}$$

where a_{ij} is the entry of the grid, the accuracy of a classifier trained on channel i and tested on channel j: the row index i is the source the model was trained on, the column index j is the target it is tested on, and a_{jj} on the diagonal is the target's own model, the reference every entry of column j is measured against. The loss is therefore zero on the diagonal by construction. Fig. C.25(a) draws the same numbers as Δ, anchored at zero. The row reading survives, the column reading inverts: column 3 is the darkest of Fig. C.24(a), yet its mean loss of 0.0228 is the *smallest* of the four. What is dark in that column is ch 3's own model, 0.900 against 0.921 to 0.926, reported by the raw matrix as though it were the effect. Ch 3 is at once the easiest target and the least accurate destination, and it takes both figures to say so: the loss alone cannot state a level, and the accuracies alone do not isolate the reference.

Panels (b) and (c) put the same twelve losses at the top of the ranking, as position on a common scale. Panel (c) also answers what a grid invites and no reader performs, whether the relation is symmetric. It is not.

Panel (d) is the form the same numbers invite once they are called a relation between channels. Normalizing each channel's three transfers to fill its sector is what makes the diagram drawable, and it discards the row reading at once. A ribbon joins a pair and has two ends, one for each direction, so nothing on the ring can be read until the direction is marked: the arrowheads are the minimum the form needs, and they are still 12 glyphs spent on what a row and a column of a grid say by position.

Shading is not required either. Fig. C.26 keeps the layout and changes only the mark. Only length can be ranked by eye.

Every magnitude channel needs a zero and a key

Shading, area and stroke width are all magnitude channels, and each needs a zero the scale is anchored to, and a key on the page stating what one unit of the mark is worth.

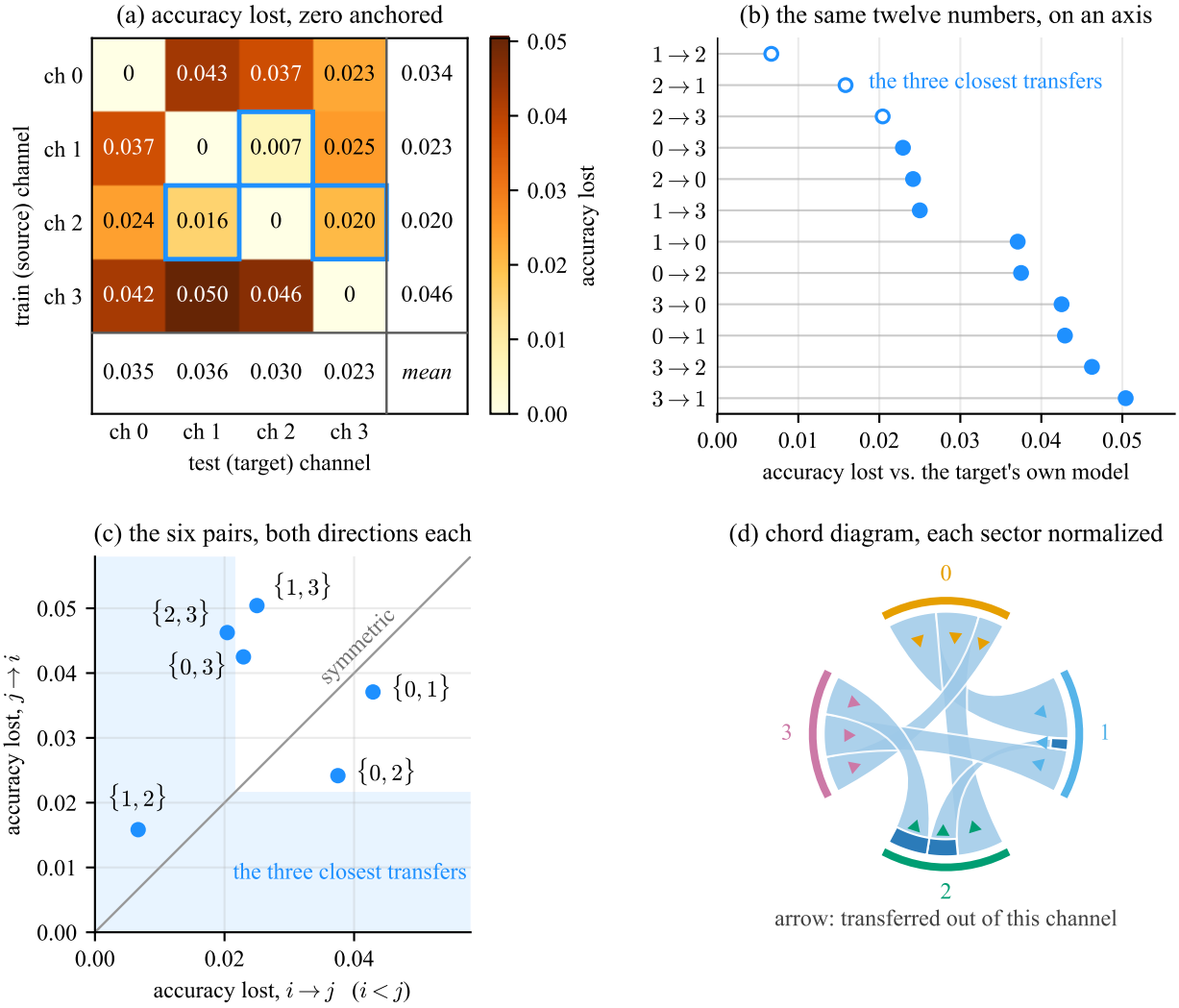


Figure C.25.: The same record as the loss Δ_{ij} of Eq. (C.3), read four ways. (a) The loss anchored at zero, the row and column means in the margin band. (b) The same twelve losses on a common scale, sorted. (c) One point per pair, its two directions its two coordinates, the lower-indexed source on the horizontal axis; the shaded band holds the three closest transfers, so a point inside it has a direction among them, and $\{1, 2\}$, inside both arms, transfers well each way. (d) The twelve as a chord diagram, each channel's three normalized to fill its own sector.

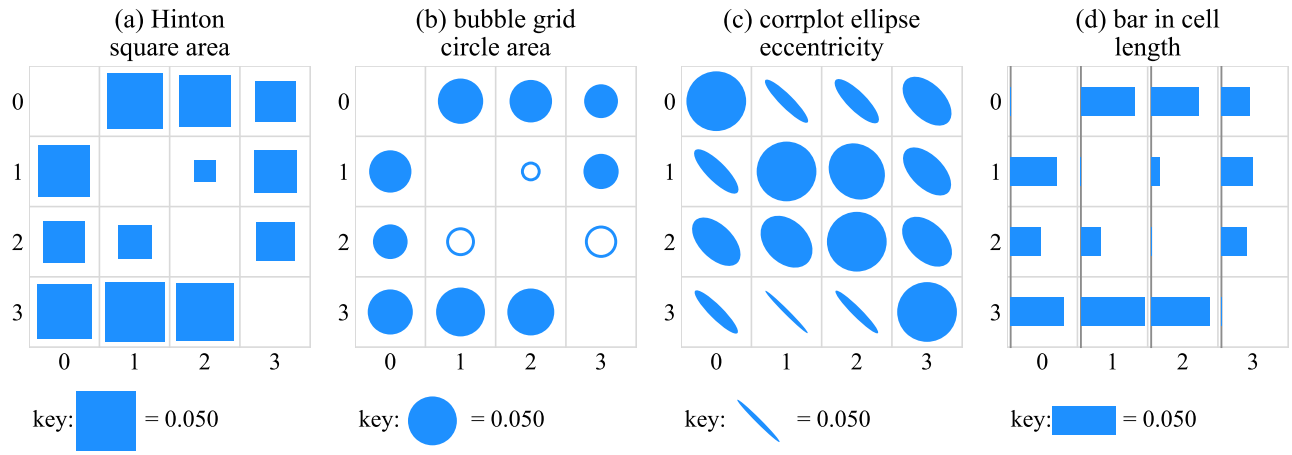


Figure C.26.: One matrix of losses, four marks, one cell size, one key per panel. (a) The Hinton diagram, square area, and (b) the bubble grid, circle area, both 4$^{\text{th}}$ in the ranking: an exact zero draws nothing, so the diagonal disappears without being told to, and (b) leaves color free for a second variable, here the three closest transfers. (c) The `corrplot` ellipse, eccentricity, third, built for a signed bounded quantity, so the largest mark of the panel lands on the diagonal. Difference between diagonal circiles is indistingushable. (d) The bar in cell, length from a common baseline, second, the only panel that can be ranked by eye. The extremes differ by a factor of 7.6, which is 7.6 times the area and 2.7 times the side.

C.7. Axes

The axis is where a plot is most often quietly wrong, because the same choice is legitimate for one encoding and misleading for another.

C.7.1. The zero baseline

Whether an axis must include zero is decided by the encoding, not by convention. A bar encodes its value as a *length* measured from the baseline. A line encodes *position*, so cutting the axis rescales the whole picture uniformly and the reader can see the axis labels. Fig. C.27 is the bar case and Fig. C.28 the position case.

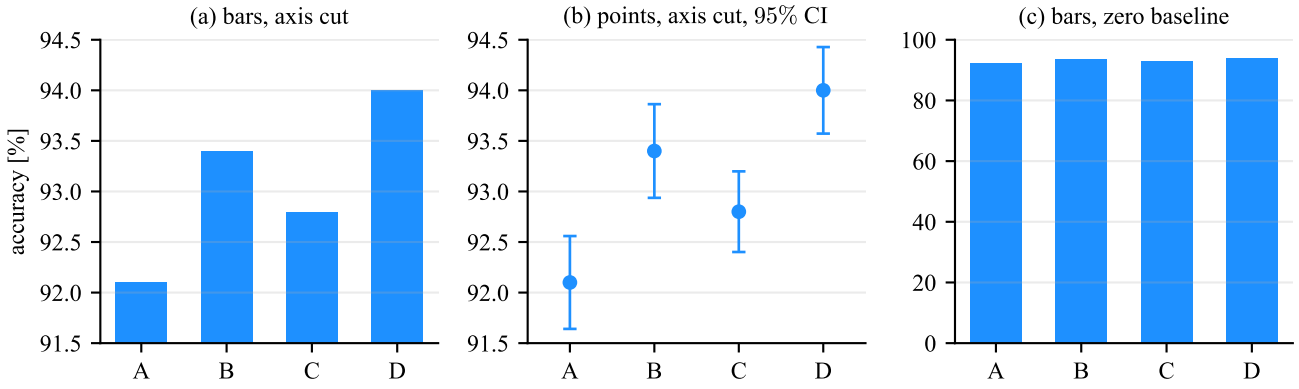


Figure C.27.: Four test accuracies, 92.1, 93.4, 92.8 and 94.0 percent. (a) With the axis cut at 91.5, the bar for D is 4.2 times the length of the bar for A, though the values differ by 2%: an exaggeration of a factor of 4.1. (b) The same numbers as points, on the same cut axis as (a), with the 95% confidence interval of each over its 5 folds. The cut is legitimate here and not in (a), because a point is at a position while a bar is a length measured from the baseline. The panel also reports what neither bar panel can: B at 93.40 ± 0.46 and D at 94.00 ± 0.43 are separated from A at 92.10 ± 0.46, and C at 92.80 ± 0.40 is not. (c) The same numbers as bars on a zero baseline, where the four models are correctly seen to be nearly tied.

A cut bar chart is not a style choice

Panel (a) of Fig. C.27 is the most common misleading chart in circulation, and it is usually produced by accident: many libraries auto-scale the axis to the data range, so cutting the baseline is what happens when nothing is specified. Any bar chart whose baseline is not zero should be treated as an error to be fixed, not a decision to be defended.
If the differences are too small to see on a zero baseline and they matter, plot the differences themselves, with their confidence intervals, on a zero that means *no difference*.

The rule is about bars, and it does not run backwards: an axis that includes zero is not automatically the safer one. Fig. C.28 is the same argument from the other side, on a quantity whose marks encode position.

Forcing zero onto panel (b) does not make it honest, it deletes the finding: the plateau is left with is less than the separation at which a reader resolves anything at all. Nothing is gained in exchange,

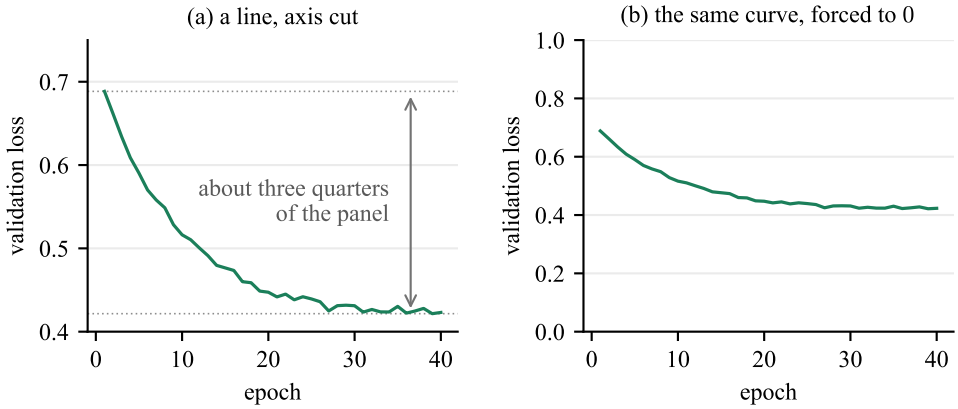


Figure C.28.: A 40-epoch validation loss, drawn twice. (a) The marks encode position, the axis is labeled, the decay and the plateau are both readable, and the limits are chosen so that the curve fills the marked 76% of the panel. (b) The identical curve forced onto 0 to 1.

because no mark in the panel is a length measured from the baseline, so there is no length for the cut to distort.

How far to cut

A cut is a choice of limits, not just a decision to leave zero out. Choose them so that the data fill about two thirds to three quarters of the panel.

C.7.2. Cumulative totals

The baseline is not the only way a chart can be exactly right about its numbers and wrong about the question. A running total is monotone by construction: it cannot fall, whatever the process underneath it does, so it always draws a rising curve. Presented as evidence that an effort is working, it is evidence of nothing. Fig. C.29 plots one record both ways.

Plot the quantity the question is about!

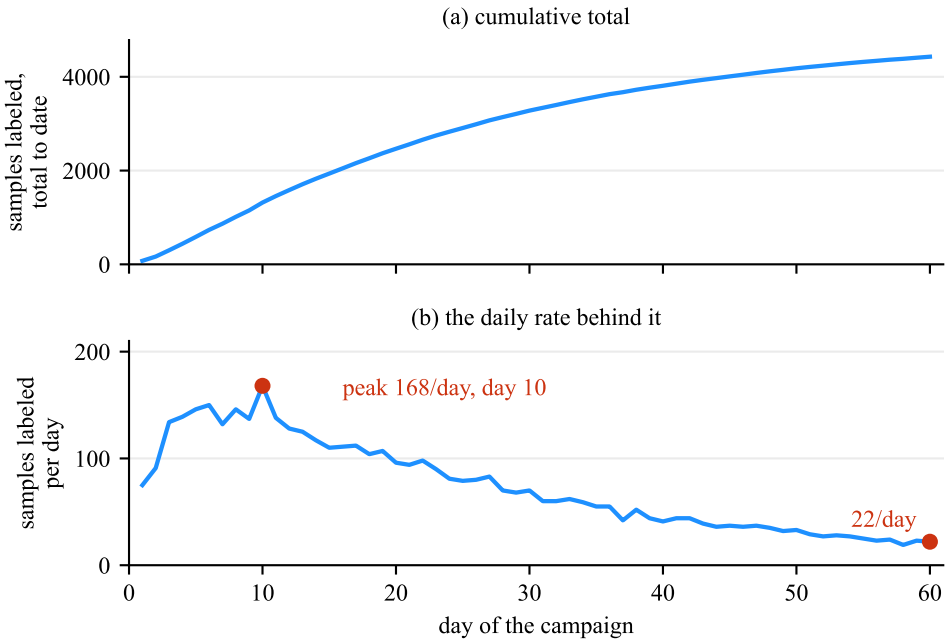


Figure C.29.: A 60-day labeling campaign, plotted as a total and as a rate. (a) The running total climbs to 4429 samples and rises on every one of its 59 steps, since a cumulative curve cannot do anything else. (b) The daily rate that produced it: a peak of 168 per day on day 10, falling to 22 per day by day 60, which is 13% of the peak. The first week contributed 20% of the total and the last week 4%. Both panels are exact, and only (b) answers whether the campaign is still working.

C.7.3. Aspect ratio

The physical shape of the axes changes the slope a reader perceives, and slope is how a line chart is read. Fig. C.30 draws one series five times with identical axis limits: the first three change the height of the axes, and the last two repeat the third of them at two thirds and at a third of its width.

Ad hoc Cleveland's rule of thumb is to *bank to* 45°: choose the aspect ratio so that the typical line segment meets the horizontal near 45°, since that is where a change in slope is easiest to detect.

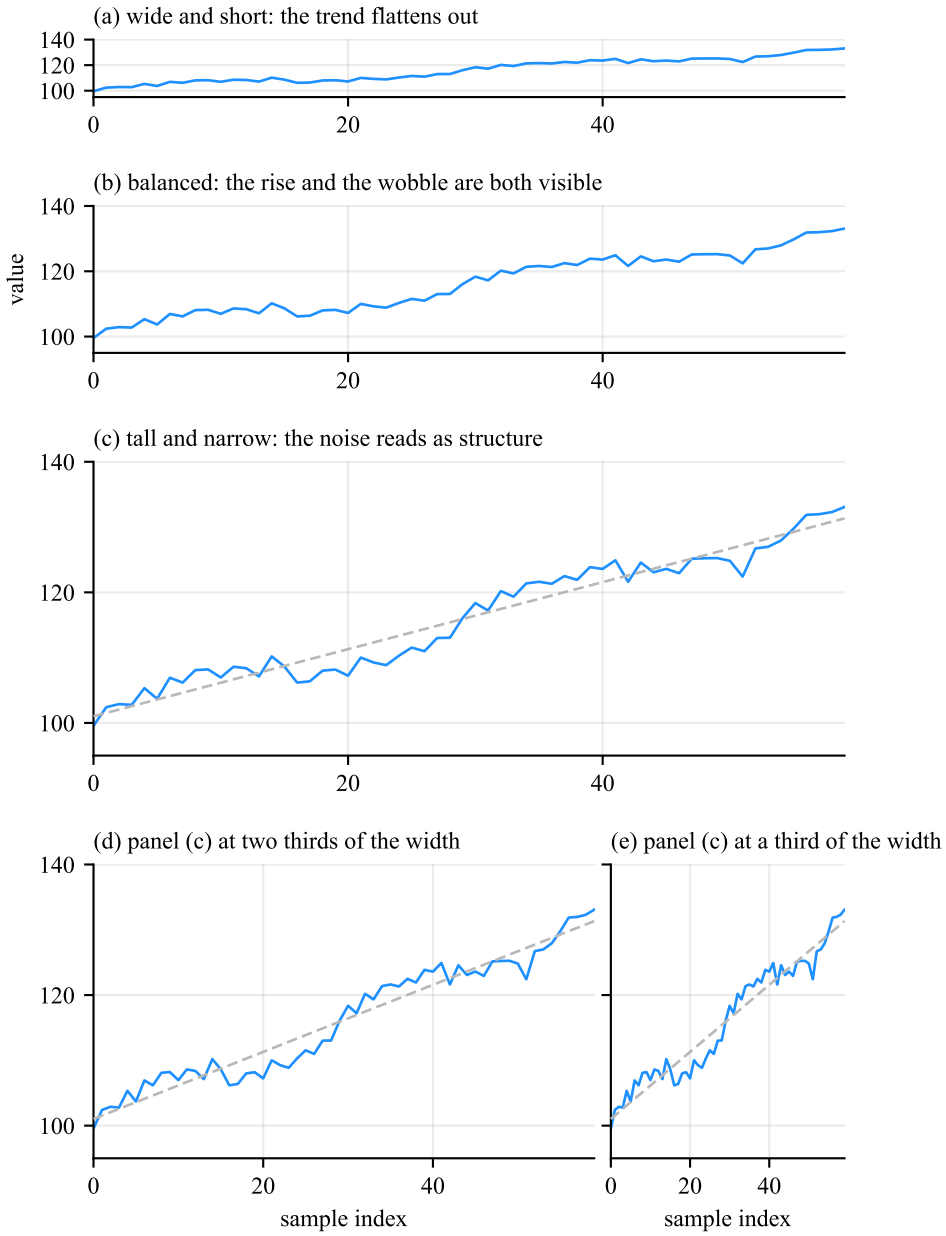


Figure C.30.: One series, one pair of axis limits, five axes shapes. The median segment of the line meets the horizontal at 7.0° in (a), 17.5° in (b) and 32.2° in (c), so the slope the reader perceives grows with the height of the axes. Panels (d) and (e) are (c) narrowed to two thirds and to a third of its width, which delivers 43.4° and 63.7°.

C.7.4. Ticks and labels

Easy numbers

Ticks at 1, 2, 5, 10 and their mulitplications/powers, never at the 7 or 13 an algorithm sometimes lands on.

Labeled axes

Every axis names a quantity and a unit. A number without a unit is not a measurement.

Compress offset

A quantity far from zero has its level stated on the panel and its deviation ticked in a unit sized to the drift. An offset parked in a corner box, or spelled out on every tick, both leave the reader doing arithmetic.

Fig. C.31(a) does what a reader normally asks for and puts the full value on every tick, and this record will not support it: six digits a label to report a drift of 131 mK, so the level is printed five times over, the quantity the figure is about is carried by the last two digits, and the ticks fall at 7 s. State the level on the panel, and tick the deviation from it in a unit sized to the drift, which is panel (b).

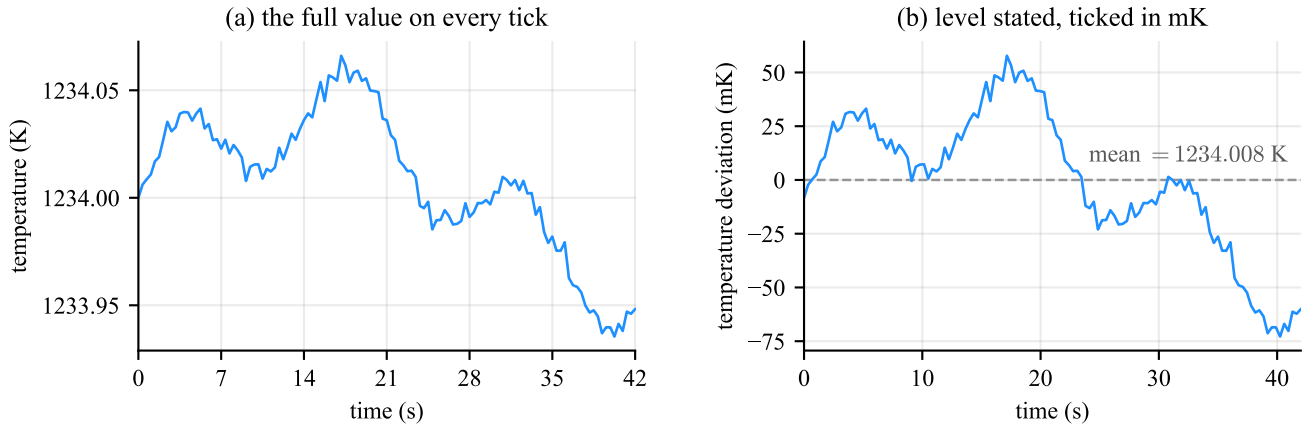


Figure C.31.: (a) The full value on every tick: the axis runs 1233.90 to 1234.10 K in steps of 0.05 K, which is six digits a label of which only the last two carry the drift, and the ticks fall at 7 s. (b) The same record with the level subtracted and the deviation from it ticked in millikelvin: ticks at 10 s and 25 mK, every one of them a small round number. Zero on this axis is the mean and nothing else, the dashed line marks it and names it in full, 1234.008 K, so any mark on the panel converts back to an absolute temperature, and the record is seen to run −73 to +58 mK about it.

And treat rotated tick labels as a diagnostic rather than a solution: if the label names do not fit horizontally, the chart wants to be horizontal.

Rotating a label is not a style preference (Fig. C.32). The six names are the same in both panels and so are the six numbers; what changes is where the names are kept. Tilted under the axis they

claim 18.3 mm of a 66 mm figure, which is 28% of its height taken out of the bars, and the reader still has to read them at an angle. Turned on their side they claim 22.8 mm of width, which comes out of the margin rather than the panel, and the same figure gives the bars 47.2 mm of height instead of 37.6.

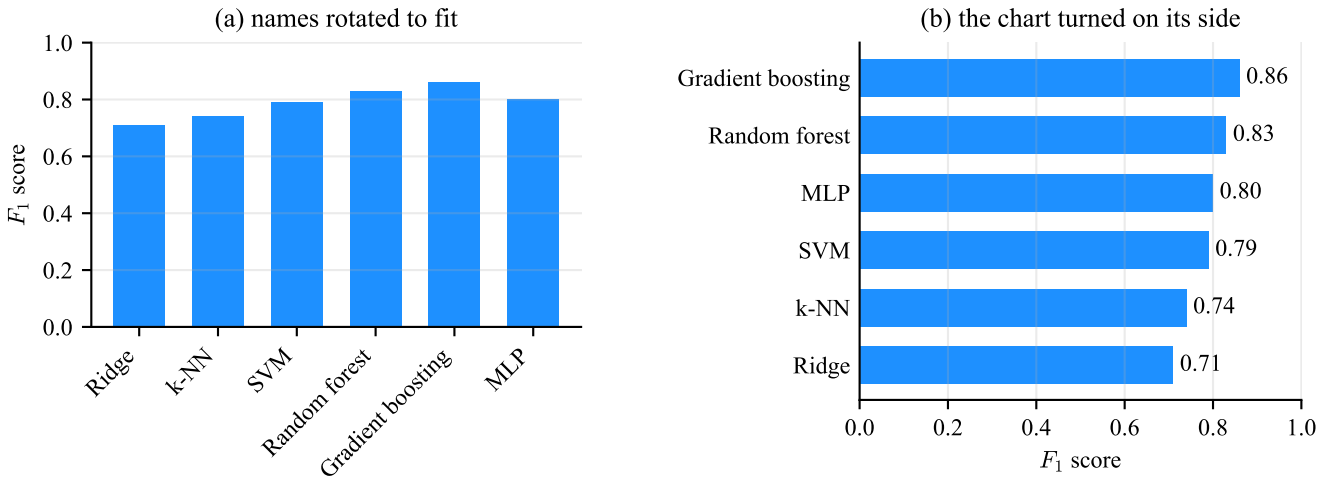


Figure C.32.: Six F_1 scores drawn twice in one figure box with nothing changed but where the category names are kept. (a) Names tilted at 45° to fit under a vertical axis. (b) The chart turned on its side.

C.7.5. Legends

A series has to be named somewhere, and there are only two approaches to put the name; the two differ in which of the two reading operations of Sec. C.4.1 the reader is made to perform.

- A *boxed legend* is a key set apart from the data, mapping a color or a line style to a name. Reading it is a table look-up, and it is a look-up per mark: the reader carries a color/style to the box, finds the matching key, carries the name back, and repeats it for the next series.
- An *end-of-line label* is the name set at the end of the series it belongs to, in that series' color. There is nothing to carry anywhere, because the name is already where the reader is looking.

Fig. C.33 draws four curves both ways.

Three consequences, in the order they usually matter.

Label directly whenever there is room. It removes the look-up entirely.

Where a box is unavoidable, order it to match the page. Sort its entries to match the vertical order of the series at the edge where the reader meets them, and place it over a region carrying no data.

What the key is made of decides what the look-up costs. A key is read no faster than the channel it maps. Dash patterns and marker shapes are *nominal*: the entries have no order among them, so each is matched glyph by glyph, and the match is made again wherever the curves cross, which is exactly where the glyphs are hardest to tell apart. Line width and lightness are *ordered*: moved together they turn the key into a sequence, read once and then predicted, and its entries come out in the order the curves appear on the page, which is the previous rule obtained for nothing. Fig. C.34 is one record under both.

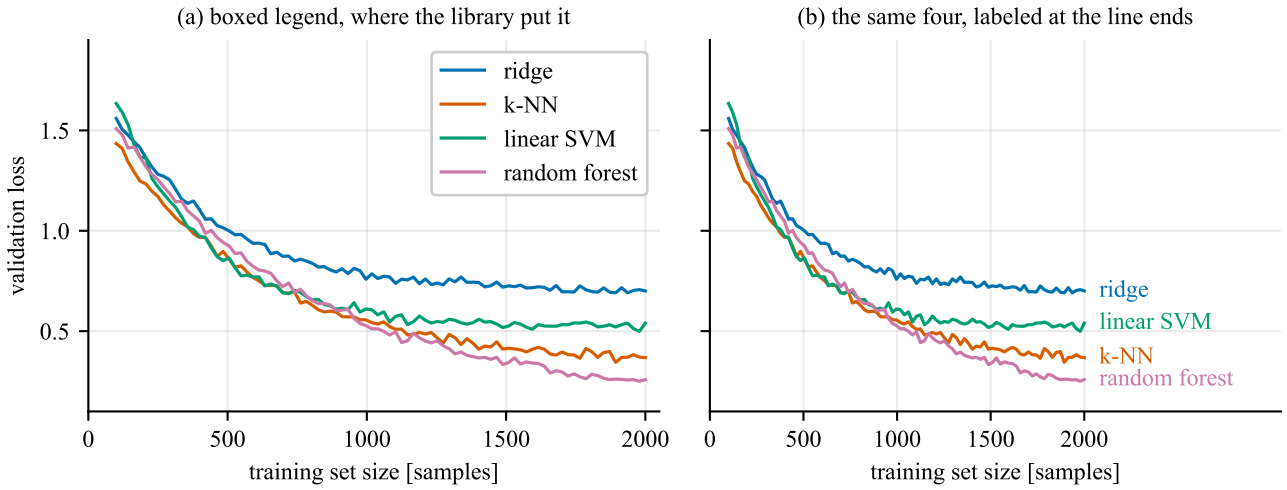


Figure C.33.: Learning curves for four models on identical axes, named two ways. (a) A boxed legend at the position the library chose: it spends 15% of the panel on the key, and its entries are in the order the series were plotted, which is not the order they arrive in at the right edge, so 2 of the 4 are out of place and the reader re-sorts on every look-up. (b) The same four named where they end. No panel area is spent, no color has to be carried anywhere, and the names survive a gray print and a color vision deficiency simulation because there the name identifies the curve and the color only decorates it.

The one place redundancy is worth its ink

Encoding a quantity twice is normally waste (Sec. C.11), and panel (b) does it on purpose: the rank of a series is in its width and in its lightness at once.

With 6+ *series, neither form works.* End labels start to collide wherever the series converge, a box with a dozen keys is a second figure to read before the first one can be, and no ladder has a dozen separable steps either. That is the point at which the answer is small multiples (Sec. C.7.6) or highlighting one series against gray (Fig. C.18), not a smaller font.

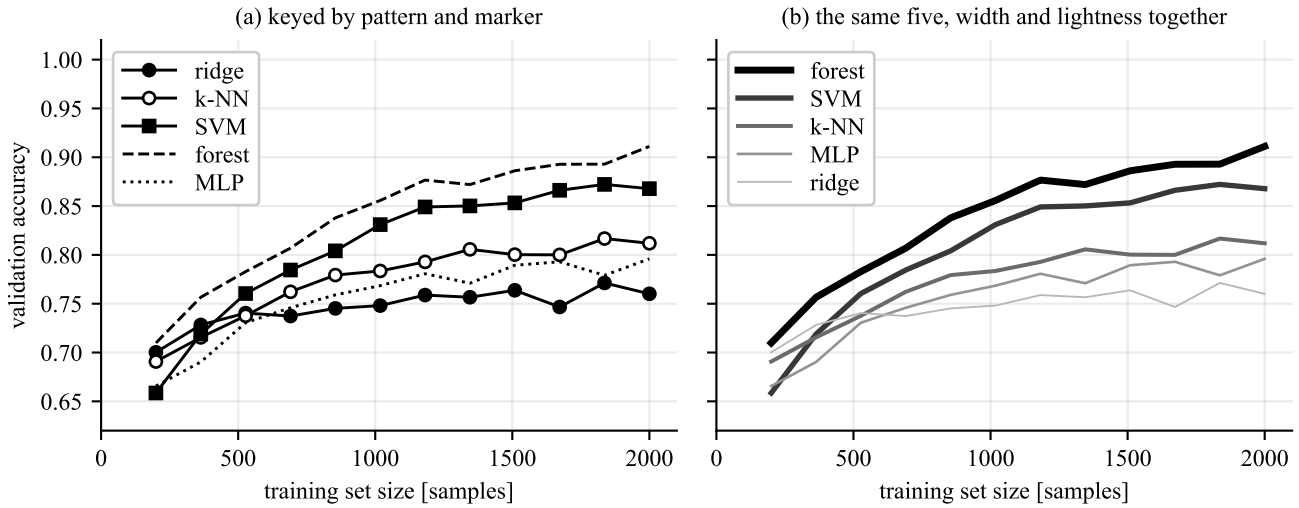


Figure C.34.: Five plot with a boxed legend in both panels. (a) Keyed by dash pattern and marker shape. The five entries have no order among them. (b) The same five keyed by one ladder, line width from 2.6 to 0.7 pt and gray from 0.00 to 0.72.

C.7.6. Many plots at once

Goal: What does a reader do with ten series that will not fit on one axis?

Legend won't help. Stop drawing one plot and draw many: one panel per series, laid out as a grid, every panel on the same range. The grid is called *small multiples*. Fig. C.35 is the overlay beside its replacement.

- *One range, set once.* Give the panels a single range on both axes and link them, so that the range is a property of the block rather than ten separate decisions. Per-panel autoscaling is the default in most libraries, and it produces ten plots that look like a comparison and *are not* one.
- *Values on the outer edge only.* The left column carries the y values and the bottom row the x values, and one label of each names the block. Ten copies of one axis is nine copies of ink, which is Sec. C.11 applied to a layout.
- *Rule every panel, at the ticks it keeps.* A panel stripped of its scale reports a shape and no quantity, exactly as a close-up does (Sec. C.7.7), and the separation still has to clear the limit of Sec. C.7.8 at the panel's reduced size.

Sec. C.4 to answer "where is this one among the rest" at a glance, and can be repeated across a row of panels to walk through several. Fig. C.18 is the comparison, on twelve series that an overlay cannot separate.

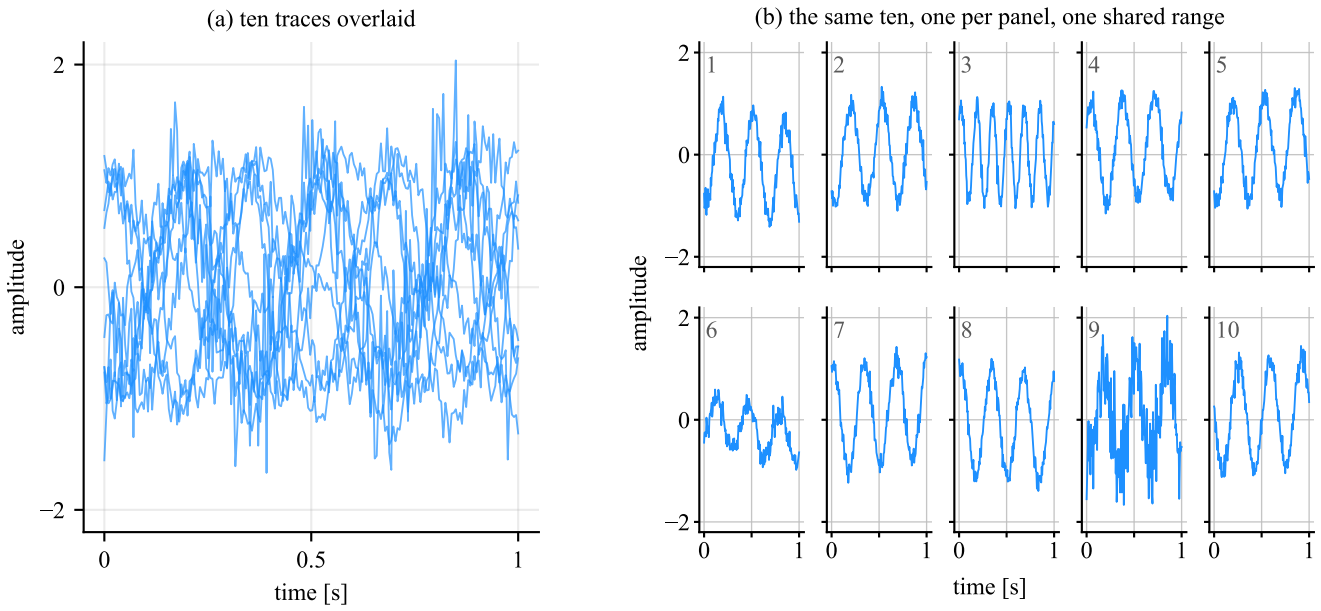


Figure C.35.: Ten recordings from one process. (a) Overlaid: the envelope is readable and nothing else is. (b) The same ten as *small multiples*, one panel each, on one linked range. Three of the ten are not like the rest, in three different ways: trace 3 runs at 6 Hz against the 3 Hz of the others, trace 6 has 0.55 of their amplitude, and trace 9 carries 4.1 times their noise. Not one of the three can be found in (a).

C.7.7. Close-ups

One axis range cannot always serve both the extent of a record and a feature inside it. A transient a few milliseconds wide, drawn across a panel that covers seconds, gets a fraction of a millimeter of the page whatever the aspect ratio, and no amount of care with the ticks recovers it. The answer is a second view at a second scale, and it comes in two forms.

- An *inset*: a small second axes drawn inside the panel, in a region that carries no data. It keeps the overview and the detail in one frame, and it costs nothing but the space it occupies.
- A *broken-out panel*: the same window as a full panel beside the overview. It is the answer when the parent has no free corner, when the zoom needs its own axis labels, or when the required magnification is large enough that a small inset would not resolve the feature anyway.

Fig. C.36 draws one record all three ways.

- *Mark the source window* with a rectangle on the parent panel, and connect the rectangle to the view it produced.
- *Keep the ticks on the close-up, and rule it at them.* A zoom without a scale shows a shape and reports no quantity, and the reader has no way to tell a 10-fold magnification from a 100-fold one. Ticks are the minimum; the grid at those ticks is what turns the magnified shape into values the reader can take off the page (Sec. C.7.8), and a close-up is read for values rather than for where the feature sits.
- *Put the inset where there is no data.* If the panel has no empty region, that is the signal to break the view out into its own panel rather than to cover the trace with it.

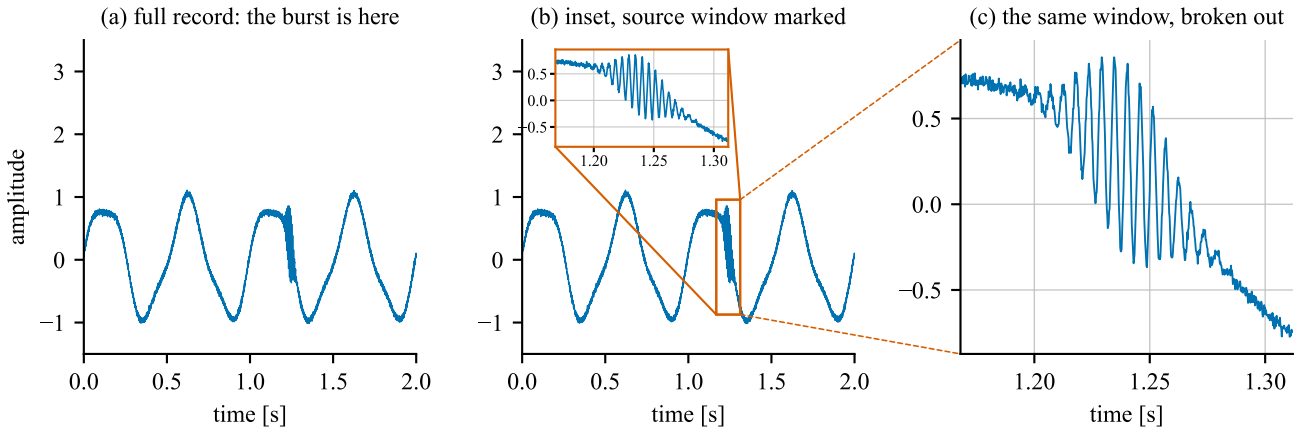


Figure C.36.: A 2 s record sampled at 5 kHz, carrying a 180 Hz burst on a 2 Hz carrier, a frequency ratio of 90. (a) The full record: the burst is at 1.24 s, it is the only thing the record is about. (b) An inset over the 144 ms window that contains it, a magnification of 14, with the source window boxed on the parent and connected to the inset so the reader can locate what was magnified. (c) The same samples as a panel of their own, which is what the window needs when the parent has no room.

C.7.8. Size and gridlines

Goal: How tall must a panel be, and how finely may it be ruled?

Two rules for the grid

- Rule the panel. The grid is the route from a mark to a number, and adding one improved accuracy at every size measured.
- Do not shrink a panel below the height at which its scale can be read.
- Do not rule it more finely than the reader can resolve.
- At a typical rendering of 96 pixels per inch that is a height of about 2 cm and a gridline separation of at least 2 mm, both measured on the printed page rather than on the screen the figure was authored on.

Both numbers are measured rather than chosen, and two results are what the measurement leaves for use.[4]

- On a 0 to 100 scale, gridlines every 10 or 20 units beat gridlines every 50 or 100.
- Not usefull once the lines fall closer together than the reader can separate them, which is about 8 pixels, or 2 mm at final size.

The default is often no grid at all, or a line every 50 units, and both leave the reader interpolating a value across half the panel. Fig. C.37 is that case beside its fix, at one panel size, so the ruling is the only thing that changes between the two.

[4]Heer and Bostock, "Crowdsourcing Graphical Perception" (CHI, 2010), third experiment.

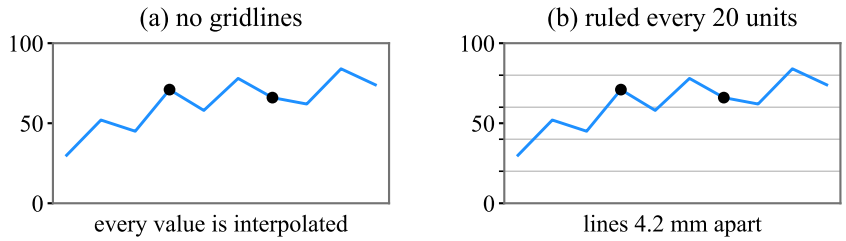


Figure C.37.: One series on a 0 to 100 scale, with the two values a reader is asked to compare, 71 and 66, marked. Both panels are 80 pixels tall, that is 21.2 mm at 96 pixels per inch, and they differ in nothing but the ruling. (a) No gridlines by default. (b) The same panel ruled every 20 units, which is 4.2 mm at this height and twice the 2.1 mm the eye needs to keep two lines apart: each marked value now sits against a line instead of being estimated.

C.7.9. Logarithmic Axes

When a quantity spans several orders of magnitude, a linear axis shows the largest decade and compresses everything else onto the baseline (Fig. C.38).

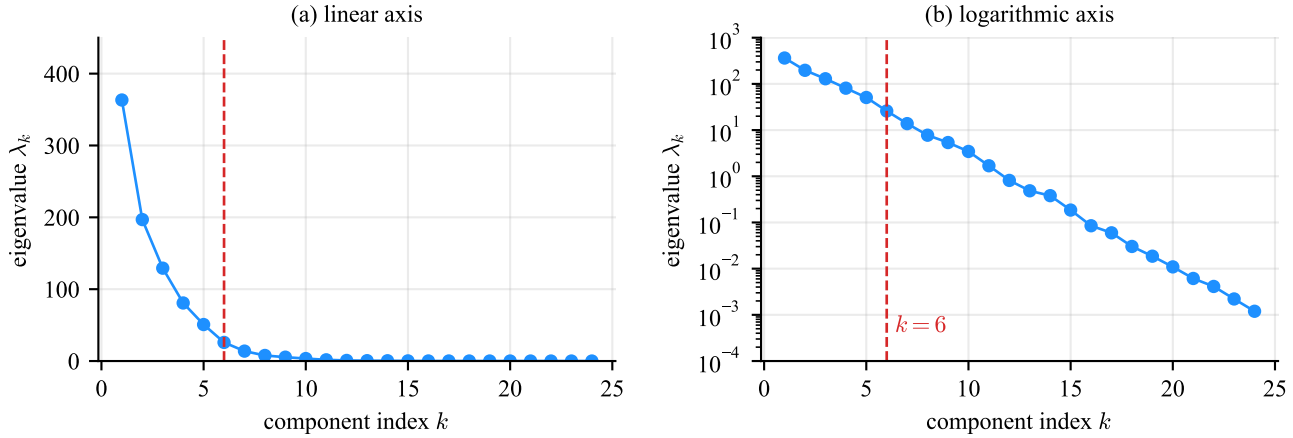


Figure C.38.: An eigenvalue spectrum spanning five and a half decades, from 363 to 0.0012. (a) On a linear axis, 15 of the 24 eigenvalues fall below 1% of the largest and are pinned to the baseline, indistinguishable from one another and from zero. (b) On a logarithmic axis, the same spectrum is a straight line, which identifies the decay as exponential, and the change of regime at the dashed line becomes something the reader can locate rather than guess.

No zero or negative values

A log axis cannot show zero or negative values, so a curve that reaches zero needs a different treatment

C.8. Choosing a Font

A figure's text is document text. It is read on the same page, at the same distance, as the body around it.

- Text sized at the width the figure will actually be printed at.
- One font (at least family) throughout the figure, text and math.

C.8.1. Figure size sets the text size

Point size is fixed in the script while the figure is scaled to fit the page, so the two are tied by one relation. A figure authored at width w_a and placed at width w_p has every glyph scaled by $s = w_p/w_a$, and the size the reader actually gets is

$$p_{\text{page}} = s\, p_{\text{script}}. \tag{C.4}$$

Authored at 15 cm and placed at 8.8 cm gives $s = 0.59$, so a 10 pt tick label arrives at 5.9 pt. Nothing in the script announces this, and nothing in the figure looks wrong until it is printed.

The reliable fix is $s = 1$: set the figure width to the width it will occupy and place it without a `width=` override, so a point in the script is a point on the page. Fig. C.39 is drawn that way, which is what lets each of its panels be read twice over.

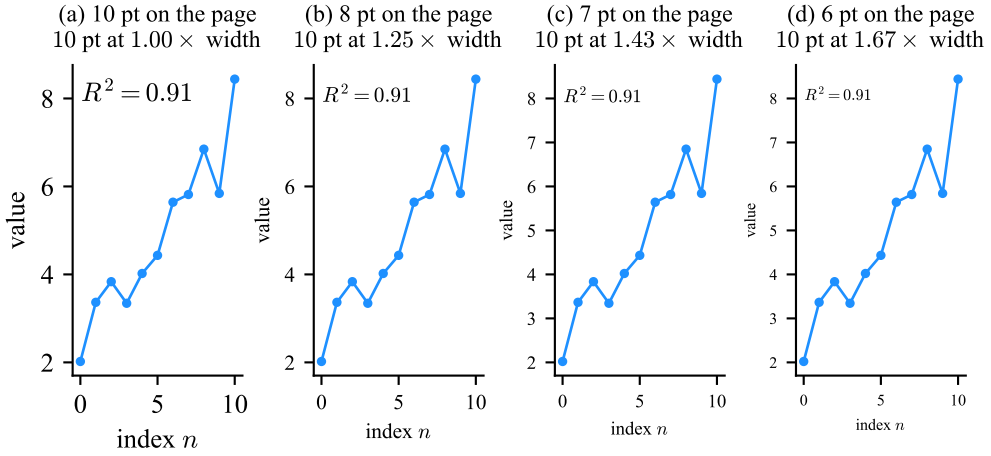


Figure C.39.: Type in a figure, at the size it is printed at. (a) to (d) One panel drawn four times, its labels at 10, 8, 7 and 6 pt. Each panel is also the second reading of its own subtitle: 6 pt on the page is what a 10 pt setting delivers once the figure has been authored at 1.67 times its final width.

C.8.2. The three kinds of face

Three distinctions cover what a plotting library will offer, and only the last of them is a matter of mechanism rather than appearance.

- *Serif*: small terminal strokes on the letters, the feet and heads magnified in Fig. C.40. It matches most printed body text, which makes it the default for a figure that will sit inside a document.
- *Sans-serif*: no terminal strokes, and a more even stroke weight. It survives small size, low resolution and a poor projector better, for the direct reason that the first detail to disappear in a serif face is the serif.
- *Monospace*: every character claims the same width, so an `i` occupies as much room as an `m`. That is the entire difference, and Fig. C.41 is it: eight `i` and eight `m` end at the same place in the monospace column and nowhere else.

The first two of the three differ in one detail, and it is a small one. At the size a label is actually read, a serif is a fraction of a millimeter of ink at the end of a stroke, which is why it takes a magnification to see what is being talked about, and why it is the first thing lost when the figure is shrunk or projected.

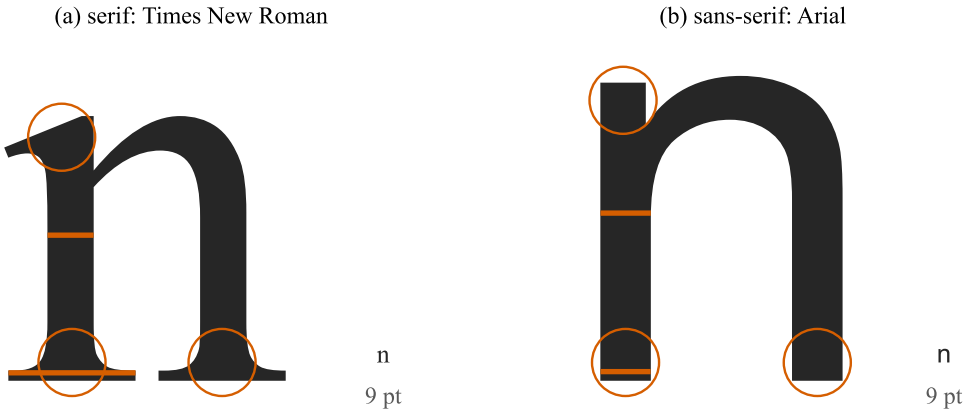


Figure C.40.: The letter `n` at 200 pt, a magnification of 22 over the 9 pt it is read at, drawn in the two text faces at one scale, with the same letter at reading size beside it. The bars measure the left stem where it is a stem and where it meets the baseline: in Times New Roman the foot is 2.8 times the stem, and in Arial the foot is the stem, 1.00. The rings mark the head of that stem and both feet of the letter, each placed where the ink was found rather than by hand; in (b) they enclose plain cut ends.

Monospace is worth reaching for when the text is something that was literally typed, since it preserves indentation and lets a reader count characters: code, file names, hex color codes such as the `#1F77B4` of Sec. C.5.1.

Monospace is for programers

Monospace is commonly used for programing environments. Modern font families features a "texture healing" technique that softens the wide gaps usually found in fixed-width text, improving code readability while keeping a strict vertical grid.

Between serif and sans there is no reliable difference in reading speed to appeal to. That choice is settled by the medium and by the document the figure will sit in, not by preference.

That is a statement about reading speed, and it is not a statement about everything else. Fig. C.42 is the case that makes the difference.

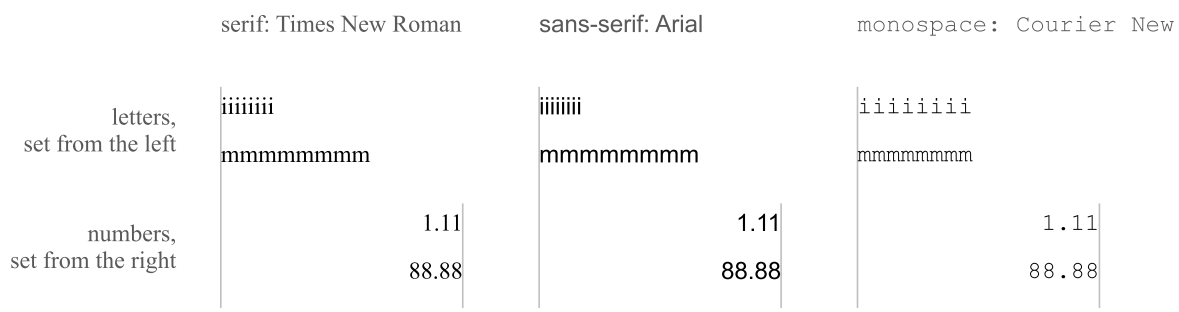


Figure C.41.: The same eight letters and the same two numbers in the three kinds of face, set against a left guide and a right guide. The `i` and `m` rows end together only in the monospace column, where both letters claim 6.00 pt; in the other two an `i` claims 0.36 and 0.27 of an `m`. The number rows align in all three, because digits are drawn to one width in text faces as well.

Two faces that look alike, and did not perform alike

In 2012 a newspaper ran a quiz that was really a typography experiment.[a] Readers were shown one passage, arguing that the Earth is unlikely to be destroyed by an asteroid, and asked whether they agreed with it. What varied was the typeface: each reader was assigned one of six at random, among them Baskerville, Computer Modern, Georgia, Helvetica and Comic Sans. About 45 000 answers came back.

Agreement ran about 1.5 percentage points higher in Baskerville than in the rest, significant at $p < 0.01$ on that sample. Computer Modern was the runner-up and Comic Sans came last. The two at the top are the pair in Fig. C.42, and that is the finding worth carrying: they are two serif book faces that an ordinary reader cannot name, cannot tell apart in running text, and was not asked about. Whatever produced the difference, it was not anybody noticing the typeface.

What may not be concluded is that a typeface can be chosen for trust. It is one experiment, on one passage, with an effect of a percentage point and a half: large enough to measure at that sample size, and far too small to design around. What it does establish is that the face is never neutral, which is a reason to set it deliberately rather than to inherit it from the library.

[a] Errol Morris, "Hear, All Ye People; Hearken, O Earth", *The New York Times*, 2012, with the analysis by David Dunning and Benjamin Berman.

C.9. How a Reader Groups Marks

A reader does not perceive marks individually and then assemble them. Grouping happens first, automatically, and follows a set of regularities. Three of them do most of the work inside ordinary charts, and Fig. C.43 shows each doing it.

Each law is a tool and a hazard, because it applies whether or not the grouping it produces is real.

- *Proximity* groups by spacing. Deliberate use: widen the gap between categories that belong

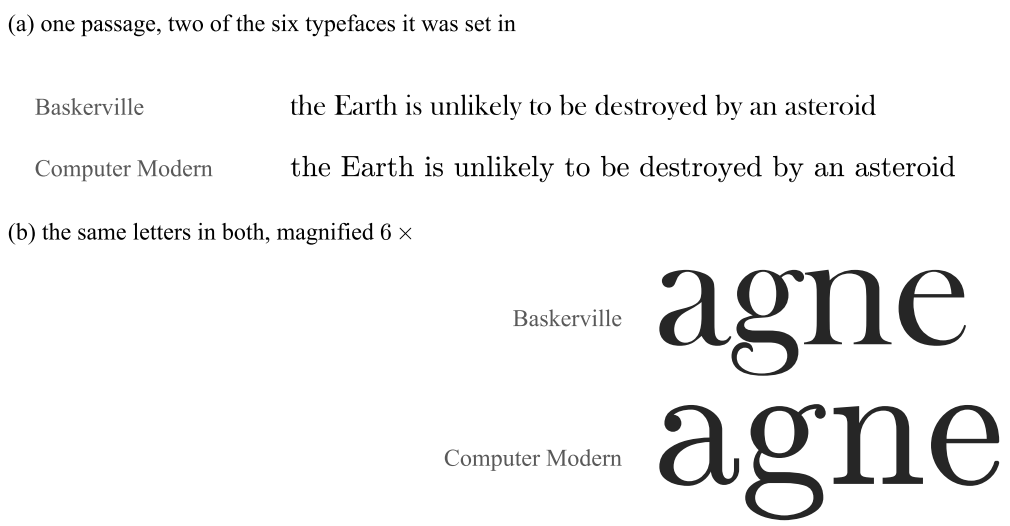


Figure C.42.: Two of the six typefaces of the experiment described below, the one that came first and the one that came second. (a) The same passage in both, at 11 pt, which is how the readers met it. (b) The same four letters magnified 6 times. The two faces differ far below the 2 mm at which detail stops being resolved.

to different conditions, and the reader sees the conditions without a legend. Accidental use: uneven spacing produced by a default layout invents groups that are not in the data.

- *Similarity* groups by shared appearance. This is what makes a color legend work at all. It also means that two unrelated series drawn in similar colors will be read as one, and that reusing a color across panels asserts that the two things are the same thing.
- *Continuity* is asserted by a connecting line, and a line is a claim that the samples between the marks lie on the path drawn. That claim is true for a time series and false for measurements at unordered categories. Connecting points across a gap in the recording makes the same false claim; break the line at the gap instead.
- *Closure* lets a reader complete a shape from part of its outline, which is why a chart needs no box around it and no axis line on the side that carries no scale.

Similarity is more than alike or not alike. Where the series being grouped have an order of their own, the appearance that groups them can carry that order as well, as illustrated in Fig. C.44.

Shading a set of series

- If the series have an order, encode it: one ramp, monotone in lightness, and let dark stand for one end of it throughout the figure.
- If they have no order, use a qualitative scheme and do not imply one.

Either way the reader is meant to get the grouping and the ranking from the same channel, without consulting the key twice.

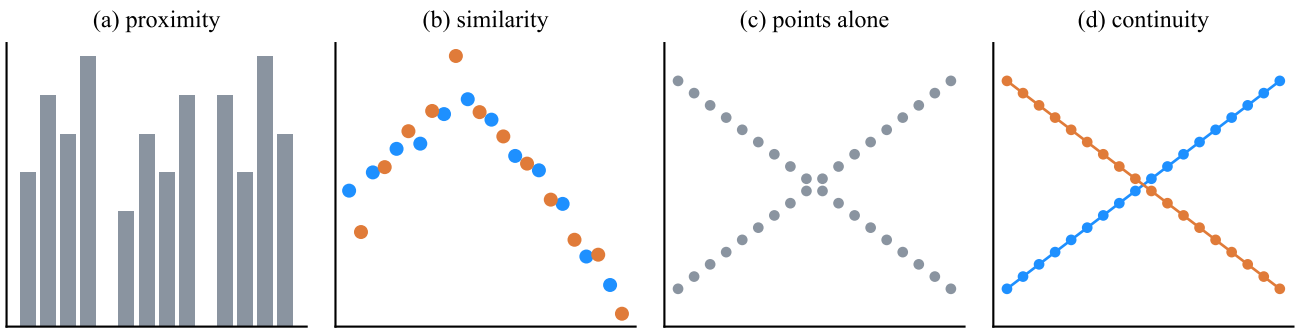


Figure C.43.: Grouping laws inside charts. (a) Proximity: twelve bars at uniform height spacing read as three groups of four, purely from the gaps. (b) Similarity: one set of points in two colors reads as two interleaved series. (c) and (d) Continuity: identical coordinates, drawn as loose points and then connected. Only the connected version resolves into two crossing curves.

C.10. Transparency

Marks land on marks. In a dense scatter at full opacity the mark drawn last covers the ones drawn before it, so the panel reports the outline of the cloud and the accident of the draw order rather than what is inside it. Every library offers an opacity setting for exactly this problem: `alpha` in `matplotlib`, `FaceAlpha` and `EdgeAlpha` in MATLAB. A mark of color c drawn at opacity α does not replace background color c_{bg} that lies under it, it is mixed into it,

$$c_{\text{out}} = \alpha\, c + (1 - \alpha)\, c_{\text{bg}}, \tag{C.5}$$

so a second mark on the same spot mixes into the result of the first, a third into the result of the second, and after n of them the ink accumulated is

$$a_n = 1 - (1 - \alpha)^n. \tag{C.6}$$

Two consequences follow from the same two lines, and they are the two halves of this section. Darkness now reports how many marks are there, which is what makes transparency work at all, and darkness now reports how many marks are there whether or not that is what the figure meant to say.

C.10.1. Misuse

Both failures in Fig. C.45 come from the same place: an opacity was set for the appearance of a single mark, and Eq. (C.5) was then applied to the places where the marks meet. Both fixes are the panel beside the failure, and they are the same fix: give the distinction to a channel that does not composite.

Two categorical colors must never overlap at α. Fig. C.45(a) is the failure: the intersection of a blue region and an orange one is a third color, present in the figure and in neither key, and by Eq. (C.5) which of the two possible mixtures it is depends on the order the two were drawn in. The legend is no help. The remedy is to keep the classes in channels that cannot mix. Fig. C.45(b) fills both classes solid, outlines each in its own key color so the shared area has a boundary of its own,

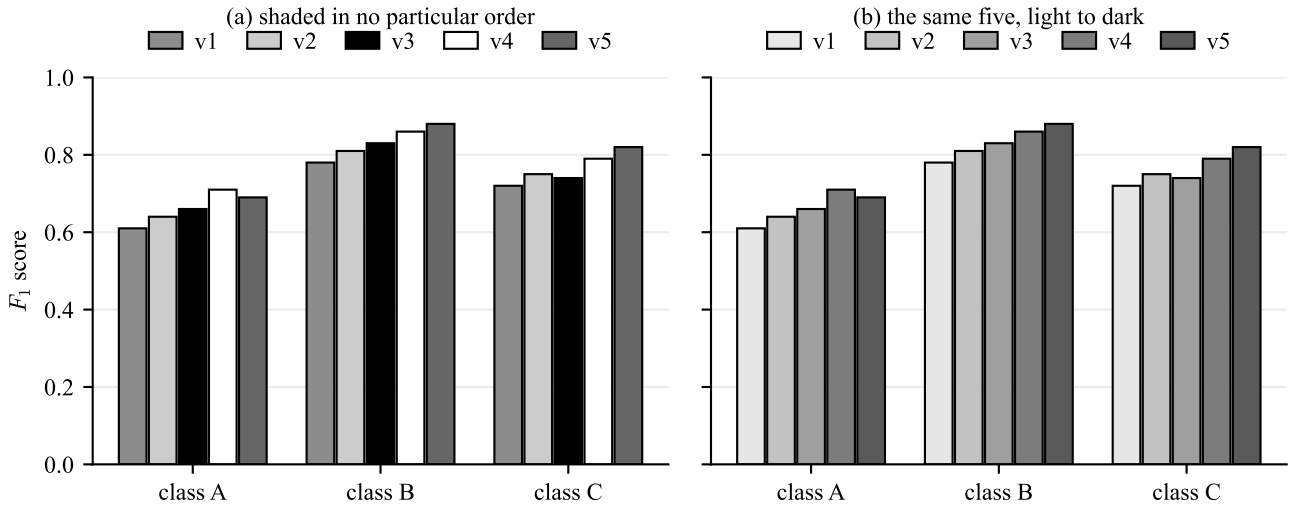


Figure C.44.: Five successive model versions scored on three classes, shaded two ways. Proximity makes the three groups in both panels, so what differs is whether a version can be traced from one group to the next. (a) Gray levels 0.55, 0.80, 0.00, 1.00 and 0.40, taken in the order the versions come in. (b) The same numbers on a ramp from 0.90 to 0.35 in equal steps.

and hatches that area in the other class color: both class colors reach the page exactly as the key shows them, the shared area is visible without being a third color, and it is a third entry in the key rather than something the reader has to decode. Separate panels, or a single hue where the darkness means only "more", do the same job.

Transparency changes the color

By Eq. (C.5) the color on the page is not the color in the palette, and it changes again if the panel is later given a background: the same mark at $\alpha = 0.5$ is one color over white and another over a gray panel. So the gray print and the color vision deficiency checks of Sec. C.5.5 have to be run on the rendered figure rather than on the palette it was specified from, and a colormap loses its perceptual uniformity as soon as an opacity is applied to it.

For de-emphasis, use a light solid color rather than α. Drawing the context series in black at low opacity, is not the same as drawing them light: the crossings accumulate by Eq. (C.6) and darken wherever the series happen to bunch. Fig. C.45(c) and (d) are the same twenty runs de-emphasized both ways, matched so that a lone curve renders identically in the two panels. The alpha panel grows a dark bar exactly where the runs converge, which competes with the highlighted curve for the attention the highlight was supposed to own, and which reports a bunching that no key on the page quantifies.

C.10.2. Accumulation

Used deliberately, the same accumulation is the point: a dense scatter stops being a silhouette and becomes a density display. That makes α a magnitude channel, sixth in the ranking of Sec. C.6. It is also a channel that runs out, as Fig. C.46 shows.

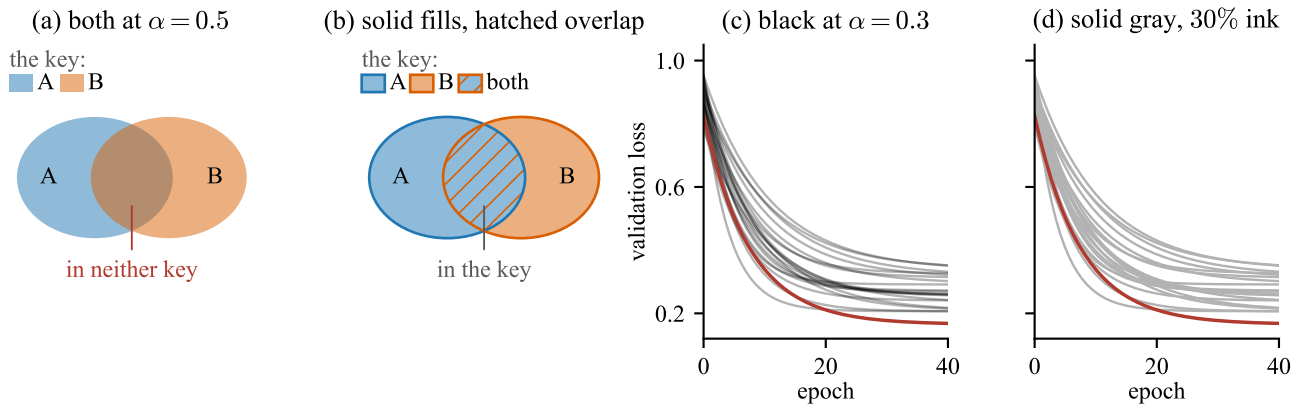


Figure C.45.: Two ways an opacity setting says something the figure did not intend. (a) Two class regions at $\alpha = 0.5$. The intersection renders as a third color that is in neither of them. (b) The same two regions with no opacity anywhere: both classes filled solid and outlined in their own key color, and the area they share carrying one fill under the other class color as a hatch, bounded by one arc of each outline. The page holds only the two colors the key names, the key names the shared area as well, and the draw order changes nothing. (c) and (d) Twenty runs behind the one under discussion: black at $\alpha = 0.3$ composites over white to exactly the 30% ink gray of (d), so a lone curve is identical in the two panels and every difference between them is a crossing. Where the runs converge, up to 7 of them fall within one line width, and that patch carries 92% ink in (c) against the 30% of a lone curve, a factor of 3.1.

Set α from the overlap, not by taste. Eq. (C.6) saturates, and quickly. Ink reaches 95% of full at $n^* = \ln(0.05)/\ln(1-\alpha) \approx 3/\alpha$, which is 4.3 marks at $\alpha = 0.5$ and 58.4 at $\alpha = 0.05$. Past that count the channel is exhausted and the picture is a silhouette again, which is why Fig. C.47(b) is barely an improvement on Fig. C.47(a). The working rule is

$$\alpha \approx \frac{1}{n_{\text{typ}}}, \tag{C.7}$$

where n_{typ} is the number of marks that typically share one marker footprint, which is a property of the data and the marker size and can be counted rather than guessed. It puts the typical spot at $1 - e^{-1} = 0.63$ of full ink, in the middle of the channel with room above it for the crowded spots and below it for the sparse ones.

The same choice hides the isolated point. An α small enough to resolve a crowd renders a lone mark at α of full ink and nothing more, 4.8% in Fig. C.47(c), so the outlier that a scatter is often drawn to find is the first thing the setting removes. Where both matter, the two populations are two layers: the crowd at low α, and the points that are alone drawn separately at full strength.

Transparency is a magnitude channel without a key

A scatter at $\alpha < 1$ encodes count as darkness, so it falls under the rule of Sec. C.6.4: every magnitude channel needs a zero and a key. It has neither, and Eq. (C.6) is not linear in the count either, so the darkness cannot be read back even in principle. Use it to reveal that structure exists, never to report how much of it there is.

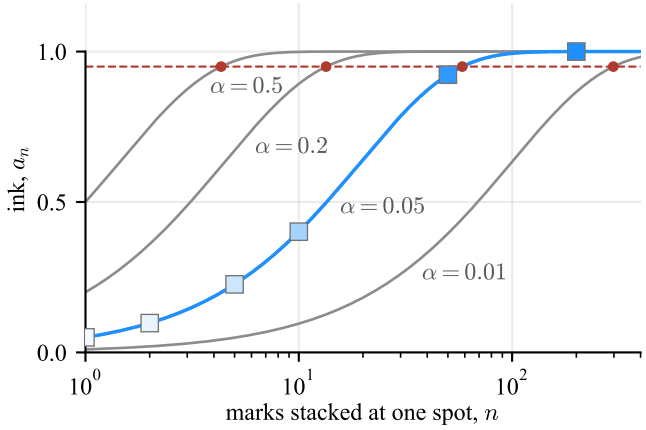


Figure C.46.: The ink a_n of Eq. (C.6) against the number of marks stacked at one spot, for four opacities. The squares on the $\alpha = 0.05$ curve are filled with the color the page actually renders at $n = 1$, 2, 5, 10, 50 and 200, so the curve can be checked against its own ink. Past the dashed level at 0.95, marked on each curve by a dot, a further mark changes nothing the reader can see.

PDF file size

A transparent mark forces a transparency group into the PDFT. The twenty thousand markers of each panel of Fig. C.47 make a 0.9 MB file that is slow to open and slower to print. *Rasterizing* the dense layer alone, keeps the axes and every label vector while the cloud becomes an image at the export resolution, and brings the same figure to 205 kB.

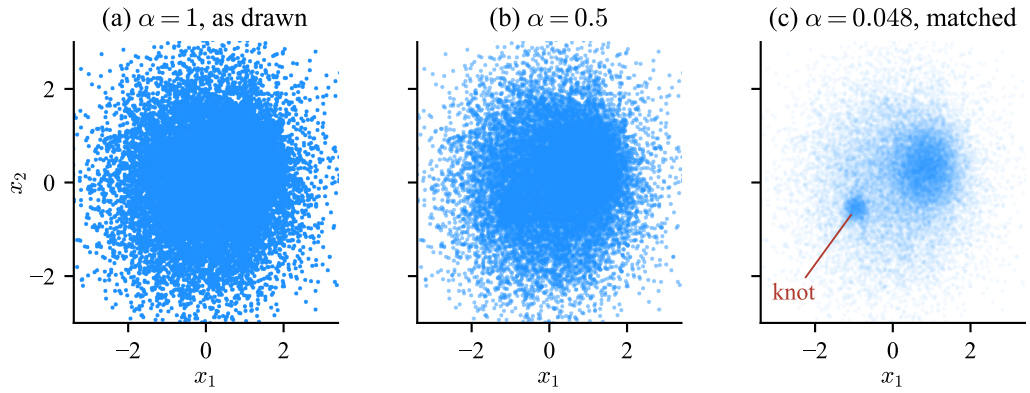


Figure C.47.: Twenty thousand points from a broad cloud holding a dense core and a small tight knot, drawn three ways on identical axes with identical 1.6 pt markers. (a) At $\alpha = 1$, 88.0% of the points land on ink that is already full, so the panel reports the outline of the cloud and the accident of draw order, and neither the core nor the knot survives. (b) At $\alpha = 0.5$, 54.0% of them still do, because ink is 0.99 of full by the seventh mark: halving the opacity does not halve the problem. (c) At $\alpha = 0.048$, which is $1/n_{\text{typ}}$ for a measured typical overlap of $n_{\text{typ}} = 21$ marks per marker footprint, the core and the knot both appear. The price is that a lone point now carries 4.8% ink, near the edge of what can be seen at all.

C.11. Ink That Carries No Data

Tufte's *data-ink ratio* is the fraction of the ink on the page that encodes data. Everything else is either structure the reader needs, such as axis lines and tick labels, or decoration. Decoration is not neutral: it competes for the same attention the data needs, and the grouping laws of Sec. C.9 act on it just as readily.

Panel (b) removes nothing that a reader uses.

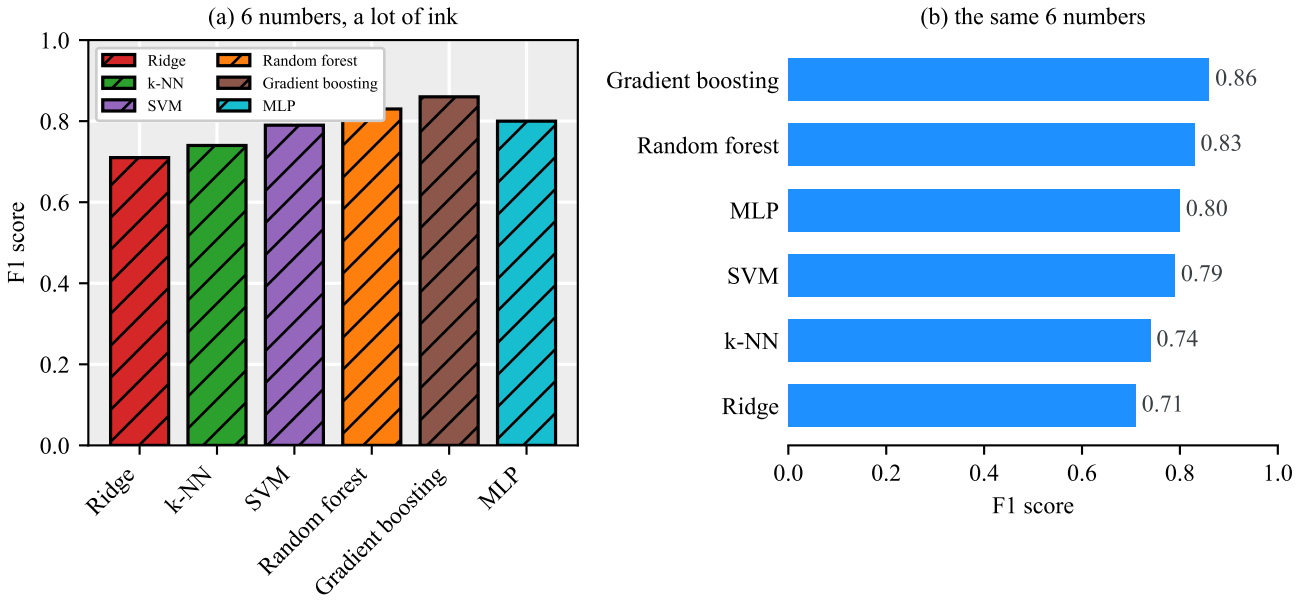


Figure C.48.: Six F_1 scores. (a) Defaults plus decoration: a gray panel, a full frame, a white grid over it, hatched fills, heavy outlines, tick labels rotated to fit, and a small-font legend that repeats the model names already on the axis and covers a bar while doing it. The model name is encoded three times over. (b) The same six numbers, sorted, horizontal so the names read normally, with the values printed at the bar ends so no axis lookup is needed.

C.12. Figures Inside a Document

A figure in a book or a paper is read on its own, out of order, before the surrounding text. Two requirements follow.

The figure must be self-contained Axes labeled with quantity and unit, series identified in the figure rather than in the body text, and any parameter the reader needs to interpret the panels stated in the caption. The test is whether a reader who has seen only this page can say what is plotted.

The caption states the finding, not the axes A caption reading "Accuracy versus training set size" repeats the axis labels and adds nothing. A caption reading "Accuracy saturates beyond 2000 samples, so the remaining error is not a data-quantity problem" tells the reader what the figure is for. The captions throughout this book are written this way, and the numbers they quote are printed by the scripts that draw the figures, so the two cannot drift apart.

Export so the figure survives the page The conventions of this book are a worked instance of everything above: figures are drawn at the width they will occupy, so Eq. (C.4) gives $s = 1$ and the point sizes in the script are the point sizes on the page, and they are exported to PDF and SVG so they stay vector at any zoom, plus JPEG at 300 dpi where a raster is needed. LaTeX then includes them without an extension, so the same source produces the PDF and the HTML build.

Further Reading

- The best stats you've ever seen, TED, Feb. 2006.
- Wikipedia: Anscombe's quartet and Anscombe, "Graphs in Statistical Analysis" (*The American Statistician*, 1973): four pages, and the origin of Fig. C.3. The argument that a regression should never be reported without the scatter it was fitted to.
- ColorBrewer, and Harrower and Brewer, "ColorBrewer.org" (*The Cartographic Journal*, 2003): the scheme families of Sec. C.5, with a picker that filters on colorblind-safe, print-safe and photocopy-safe.
- Wong, "Points of View: Color Blindness" (*Nature Methods*, 2011): two pages, and the source of the eight-color palette used in Fig. C.15. See also Machado, Oliveira and Fernandes (2009) for the simulation model behind that figure.
- "Color blindness" (Wikipedia): the conditions the second check of Sec. C.5.5 tests against — which types exist, how common each is, and which pairs of colors they merge.

Software

- `matplotlib`, plot types: one thumbnail and one call per chart type, grouped by the kind of data. The fastest way to answer the "which chart" question of Sec. C.6 without inventing something exotic.
- `matplotlib`, the gallery: the same catalog with complete runnable source for every figure, including the colormap comparisons of Sec. C.5 and the small-multiple layouts of Sec. C.7.6.
- MATLAB plot gallery: the MATLAB counterpart, organized the same way, for the figures produced by the scripts in `matlab/`.
- "Types of MATLAB Plots": the same catalog inside the documentation, indexed by data type, with the function name for each.
- "Choosing colormaps in Matplotlib": the sequential, diverging and qualitative families of Sec. C.5, with the gray level of each map plotted along it, which is the measurement behind Fig. C.11.
- ColorBrewer: interactive scheme selection with color-blind-safe and print-safe filters, originally built for cartography and applicable unchanged to any categorical or sequential encoding.
- `colorcet`: colormaps beyond the `viridis` family that are built to the same rule — equal steps in the data look like equal steps — including maps designed for cyclic quantities such as phase.
- `matplotlib`, `hexbin`: the display to fall back on when a scatter is too dense for any opacity setting: the count per cell in one call, with the colorbar that α cannot carry and the log-count option that a heavy-tailed density needs.
- `datashader`: the same idea at the size where binning has to happen before the plot exists, tens of millions of points aggregated into the pixel grid of the figure rather than handed to the renderer as marks.
- `matplotlib`, radar chart: the polygon of Sec. C.6.3 built on a polar axes, with the category-to-spoke transform written out, for the cases where the form is the right one.
- `matplotlib`, Hinton diagram: the square-area matrix of Fig. C.26(a) in about twenty lines, including the sign handling that a signed weight matrix needs.

Methods

- Cleveland and McGill, "Graphical Perception: Theory, Experimentation, and Application

to the Development of Graphical Methods” (*J. American Statistical Association*, 1984): the experiments that produced the encoding ranking of Sec. C.6, and the source of the banking-to-45° rule.

- Heer and Bostock, “Crowdsourcing Graphical Perception” (CHI, 2010): the ranking replicated on a large sample, extended to rectangular area encodings and to chart sizes below the range Cleveland and McGill tested. Its third experiment is the chart-size and gridline-spacing study of Sec. C.7.8, whose recommendation is that gridlines be separated by at least 8 pixels and that chart height beyond 80 pixels buys little on a 0 to 100 scale.
- Behrisch, Bach, Henry Riche, Schreck and Fekete, “Matrix Reordering Methods for Table and Network Visualization (*Computer Graphics Forum*, vol. 35, no. 3, 2016): the sorting rule of Sec. C.4.1 applied to the rows and columns of the matrix of Sec. C.6.4, with a survey of the algorithms and guidance on which to use for a given size and structure.
- Krzywinski *et al.*, “Circos: An Information Aesthetic for Comparative Genomics (*Genome Research*, vol. 19, no. 9, 2009, pp. 1639–1645): the chord diagram of Fig. C.25(d) where it is the right answer, on data that is a genuine flow between many entities rather than four near-equal levels.
- Porter and Duff, “Compositing Digital Images (*SIGGRAPH*, vol. 18, no. 3, 1984, pp. 253–259): the paper that introduced the alpha channel and the *over* operator of Eq. (C.5), whose repeated application is Eq. (C.6) and whose order dependence is Fig. C.45(a).
- Flannery, “The Relative Effectiveness of Some Common Graduated Point Symbols in the Presentation of Quantitative Data (*The Canadian Cartographer*, vol. 8, no. 2, 1971, pp. 96–109): why a circle whose area encodes a value is read slightly small, and the correction cartographers apply for it. The measurement behind the caution on Fig. C.26(b).
- Hinton, “Connectionist Learning Procedures (*Artificial Intelligence*, vol. 40, 1989, pp. 185–234): a survey from the period in which a weight matrix was routinely shown as a grid of scaled squares, the display that Fig. C.26(a) is named after.
- Tufte, *The Visual Display of Quantitative Information* (2nd ed., 2001): data-ink ratio, chart junk and the lie factor, which are Sec. C.11 and Sec. C.7 in their original form.
- Tufte, *Visual Explanations* (1997), Ch. 2: the *Challenger* analysis of Sec. C.2, reproducing the charts presented on the night and reconstructing what a temperature axis would have shown.
- *Report of the Presidential Commission on the Space Shuttle Challenger Accident* (1986): the primary source, including the O-ring damage record and the full set of charts of Fig. C.1.
- Dalal, Fowlkes and Hoadley, “Risk Analysis of the Space Shuttle: Pre-Challenger Prediction of Failure” (*J. American Statistical Association*, 1989): the same data treated as a logistic regression, with the failure probability at 31 °F and an honest account of how much the extrapolation can support.
- Few, *Show Me the Numbers* (2nd ed., 2012): the pre-attentive feature list of Sec. C.4 and a practical treatment of tables, which most visualization books skip.
- *Encyclopædia Britannica*, “Information theory: Physiology”: the per-sense transmission rates of Table C.1 and the estimate of conscious throughput behind the compression argument of Sec. C.4.
- Wong, *The Wall Street Journal Guide to Information Graphics* (2010): one page per problem, in the before-and-after form of Fig. C.48.
- Borland and Taylor, “Rainbow Color Map (Still) Considered Harmful” (*IEEE Computer Graphics and Applications*, 2007): why `jet` produces the false boundaries of Fig. C.12, with examples from medical imaging where the artifacts were mistaken for findings.
- Crameri, Shephard and Heron, “The misuse of colour in science communication” (*Nature*

Communications, 2020): a survey of colormap use across published work, with the perceptual-uniformity and color-blind-safety tests of Sec. C.5 applied at scale.

- Munzner, *Visualization Analysis and Design* (2014): the systematic treatment, organizing the whole field by what task the reader is performing and which channel serves it.
- NVIDIA, Georgia Institute of Technology and Prairie View A&M University, *Accelerated Data Science Teaching Kit*, Modules 7 and 8 (CC BY-NC 4.0): the course material this appendix was drafted from, covering the same ground as lecture slides with additional worked examples.

Bibliography

[1] Tomas Andersson. *Selected topics in frequency estimation*. PhD thesis, KTH Royal Institute of Technology, 2003.

[2] Dima Bykhovsky. Experimental lognormal modeling of harmonics power of switched-mode power supplies. *Energies*, 15(2), 2022.

[3] Dima Bykhovsky and Asaf Cohen. Electrical network frequency (ENF) maximum-likelihood estimation via a multitone harmonic model. *IEEE Transactions on Information Forensics and Security*, 8(5):744–753, 2013.

[4] Lorenzo Ciampiconi, Adam Elwood, Marco Leonardi, Ashraf Mohamed, and Alessandro Rozza. A survey and taxonomy of loss functions in machine learning. *arXiv preprint arXiv:2301.05579*, 2023.

[5] Angus Dempster, François Petitjean, and Geoffrey I Webb. Rocket: exceptionally fast and accurate time series classification using random convolutional kernels. *Data Mining and Knowledge Discovery*, 34(5):1454–1495, 2020.

[6] Angus Dempster, Daniel F Schmidt, and Geoffrey I Webb. Minirocket: A very fast (almost) deterministic transform for time series classification. In *Proceedings of the 27th ACM SIGKDD conference on knowledge discovery & data mining*, pages 248–257, 2021.

[7] Bo Diao, Kun Wen, Jian Chen, Yueping Liu, Zilin Yuan, Chao Han, Jiahui Chen, Yuxian Pan, Li Chen, Yunjie Dan, Jing Wang, Yongwen Chen, Guohong Deng, Hongwei Zhou, and Yuzhang Wu. Diagnosis of acute respiratory syndrome coronavirus 2 infection by detection of nucleocapsid protein. *medRxiv*, 2020.

[8] Sharon Gannot, Zheng-Hua Tan, Martin Haardt, Nancy F Chen, Hoi-To Wai, Ivan Tashev, Walter Kellermann, and Justin Dauwels. Data science education: The signal processing perspective [sp education]. *IEEE Signal Processing Magazine*, 40(7):89–93, 2023.

[9] Toni Giorgino. Computing and visualizing dynamic time warping alignments in r: the dtw package. *Journal of statistical Software*, 31:1–24, 2009.

[10] Monson H Hayes. *Statistical Digital Signal Processing and Modeling*. John Wiley & Sons, 1996.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[12] Steven M. Kay. *Fundamentals of Statistical Signal Processing, Volume I: Estimation Theory*. Prentice Hall, 1993.

[13] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2017.

[14] Jason Lines, Sarah Taylor, and Anthony Bagnall. Hive-cote: The hierarchical vote collective of transformation-based ensembles for time series classification. In *2016 IEEE 16th international conference on data mining (ICDM)*, pages 1041–1046. IEEE, 2016.

[15] Boaz Porat. *Digital processing of random signals: theory and methods*. Courier Dover Publications, 2008.

[16] Pavel Senin and Sergey Malinchik. Sax-vsm: Interpretable time series classification using sax and vector space model. In *2013 IEEE 13th international conference on data mining*, pages 1175–1180. IEEE, 2013.

[17] Albert Wong, Athena Nguyen, Eugene Li, Yew-Wei Lim, Mike Wu, and Shuk Wai Tsang. Combining classifiers for improved accuracies -voting and linearly weighted algorithms, Feb 2026.

[18] Lexiang Ye and Eamonn Keogh. Time series shapelets: a new primitive for data mining. In *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 947–956, 2009.