Machine Learning & Signals Learning

Dima Bykhovsky

March 9, 2026

Link to PDF version of this file.

Contents

Part I.

Machine Learning

1. Descriptive Statistics Basics

Goal: Describe the concise characteristics of a data.

Contents

1.1. Basic Characterization

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

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

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

1.1.1. Mean

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

$$\bar{\mathbf{x}} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{1.1}$$

Properties:

- 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.

1.1.2. 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 $x_{(1)} \leq \cdots \leq x_{(n)}$, the *sample median* is

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

1.1.3. 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.

1.1.4. Variance

Sample variance is given by

$$s_{n-1}^2 = \frac{1}{n-1} \sum_{i=1}^{n} \Big(x_i - \bar{x} \Big)^2 \tag{1.2}$$

This formulas is unbiased.

The more intuitive (biased) formula is

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

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)

1.1.5. Repeated Experiments Characterization

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

Mean

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

$$\lim_{k\to\infty} \frac{1}{k}\sum_{j=1}^{k} \bar{\mathbf{x}}_j \to \mu \tag{1.4}$$

This principle is illustrated in Fig. 1.1).

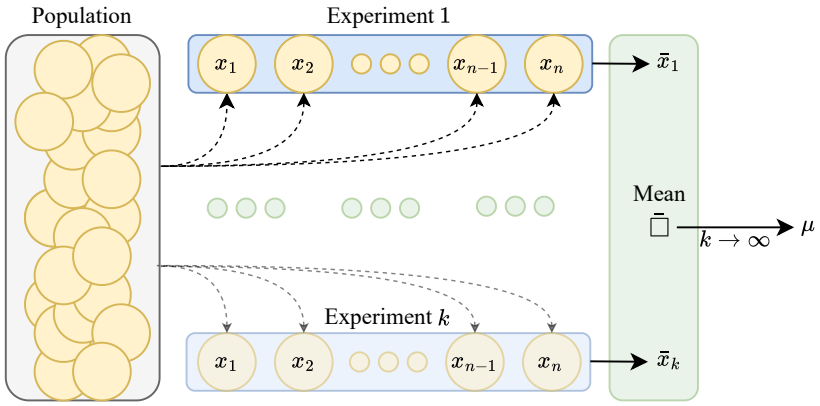


Figure 1.1.: Visualization of convergence of k repeated experiments to the true mean.

Variance

In the case of biased variance, the mean of the $s_{n_1}, \ldots, s_{n_k}$ values converges to $\frac{n-1}{n}\sigma^2$ and systematically underestimates the true population variance σ^2,

$$\lim_{k\to\infty} \frac{1}{k} \sum_{j=1}^{k} s_{n_j} \to \frac{n-1}{n}\sigma^2 \tag{1.5}$$

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

$$\lim_{k\to\infty} \frac{1}{k} \sum_{j=1}^{k} s_{n-1_j} \to \sigma^2 \tag{1.6}$$

This principle is illustrated in Fig. 1.2.

Note, the difference between s_n and s_{n-1} goes smaller for high n.

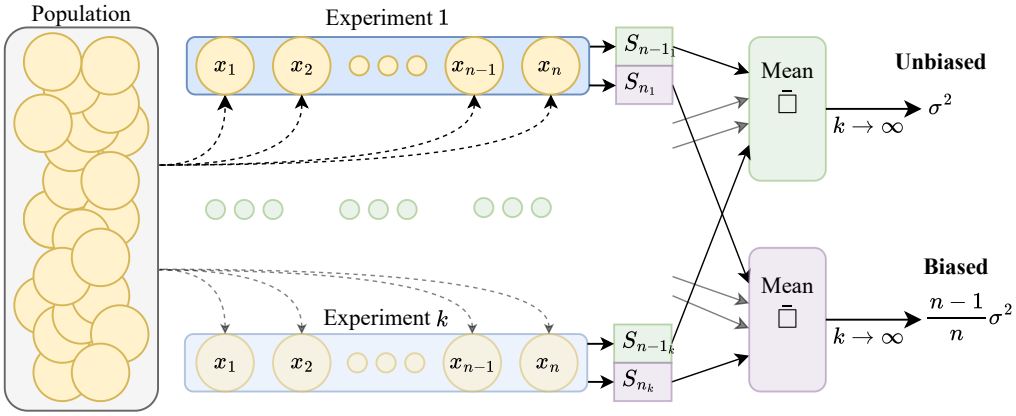


Figure 1.2.: Visualization of convergence of k repeated experiments to the biased and unbiased variances.

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

$$\mathrm{MSE}_{n-1} = \frac{1}{k} \sum_{j=1}^{k} \left(s_{n-1_j} - \sigma^2\right)^2 \to \frac{2}{n-1}\,\sigma^4 \tag{1.7}$$

$$\mathrm{MSE}_{n} = \frac{1}{k} \sum_{j=1}^{k} \left(s_{n_j} - \sigma^2\right)^2 \to \frac{2n-1}{n^2}\,\sigma^4 \tag{1.8}$$

and

$$\mathrm{MSE}_{n-1} > \mathrm{MSE}_{n} \tag{1.9}$$

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

Example 1.1: 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.

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

n	$\widehat{\mu}_n$	Empirical $\mathrm{Var}[\bar{x}]$	Theoretical $\mathrm{Var}[\bar{x}] = 1/n$	$\widehat{\sigma}^2_{\text{biased}}$	$\widehat{\sigma}^2_{\text{unbiased}}$	$\widehat{\mathrm{MSE}}_{\text{biased}}$	$\widehat{\mathrm{MSE}}_{\text{unbiased}}$
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

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`.

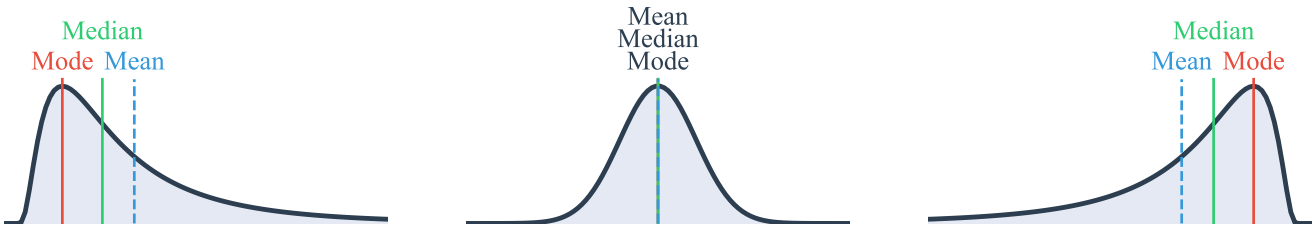
1.1.6. Skewness

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.

1.2. Histogram

Goal: Visualization of the experimental data.

1.2.1. `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. 1.4(a).

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

Figure 1.4.: Example of histograms: (a) `count`, (b) `probability`.

1.2.2. `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,

$$p_X[x_i] \approx \frac{n_i}{N}, \qquad i = 1, \ldots, k. \tag{1.10}$$

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

A graphical example of this histogram type is shown in Fig. 1.4(b).

1.2.3. Large number of outcomes

When the number of possible outcomes, k, is large (on the order of hundreds or more, or even 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

$$\tilde{x}_i = x_{\min} + \left(i - \frac{1}{2}\right)\Delta x, \qquad i = 1, \ldots, k, \tag{1.11}$$

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. 1.5.

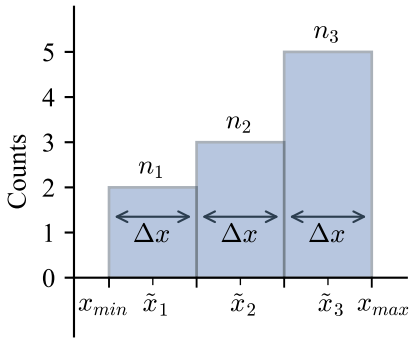


Figure 1.5.: A `count` histogram for a large number of experimental outcomes.

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

1.2.4. 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:

$$f_X(x_i) \approx \frac{n_i}{N} \cdot \frac{1}{\Delta x}, \quad i = 1, \ldots, k \tag{1.12}$$

Note the normalization factor $1/\Delta x$, which distinguishes this from the discrete case in (1.10).

A graphical example of this histogram type is shown in Fig. 1.6.

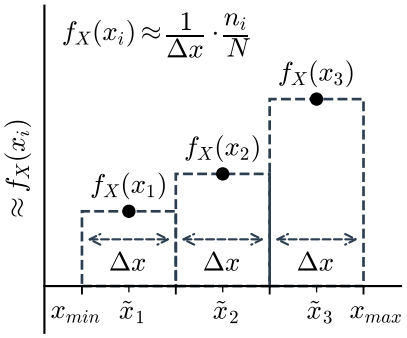


Figure 1.6.: A `pdf` histogram for a large number of experimental outcomes.

Derivation

Based on the principle

$$\Pr(a < X \leq b) = \int_a^b f_X(x)\, dx, \tag{1.13}$$

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

$$\Pr\left(x_0 - \tfrac{\Delta x}{2} < X \leq x_0 + \tfrac{\Delta x}{2}\right) = \int_{\Delta x} f_X(x)\, dx \approx f_X(x_0)\, \Delta x. \tag{1.14}$$

An illustration of this principle is shown in Fig. 1.7. Rearranging,

$$f_X(x_0) \approx \Pr\left(x_0 - \tfrac{\Delta x}{2} < X \leq x_0 + \tfrac{\Delta x}{2}\right) \cdot \frac{1}{\Delta x}. \tag{1.15}$$

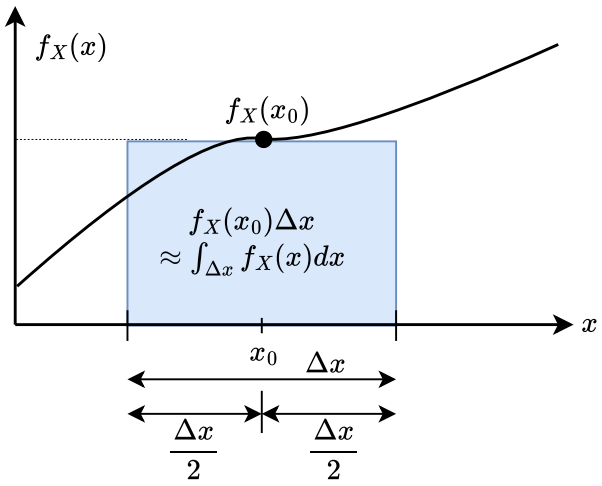


Figure 1.7.: Illustration of the histogram principle: the shaded area $f_X(x_0)\,\Delta x$ approximates the probability of falling within the bin.

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

$$\Pr\left(\tilde{x}_i - \tfrac{\Delta x}{2} < X \le \tilde{x}_i + \tfrac{\Delta x}{2}\right) \approx \frac{n_i}{N}, \quad i = 1, \ldots, k. \tag{1.16}$$

Substituting (1.16) into (1.15) yields the PDF approximation formula in (1.12).

Example 1.2: 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.

1.3. Boxplot

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

1.3.1. Quartiles

Quartiles: Given the order statistics $x_{(1)} \le \cdots \le x_{(n)}$, 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.

1.3.2. Five-Number Summary

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

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

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.

1.3.3. Construction

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. 1.8.

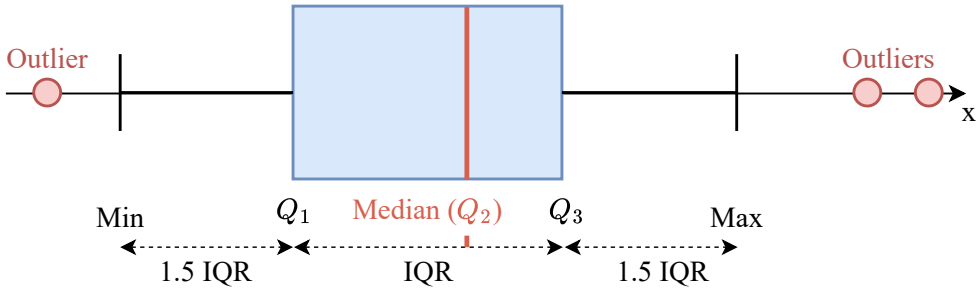


Figure 1.8.: Anatomy of a boxplot: the box spans the IQR from Q_1 to Q_3, the median Q_2 is marked inside, whiskers extend to the most extreme non-outlier observations, and individual outliers are shown beyond.

1.3.4. 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. 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.

[1]This smoothing is termed kernel density estimation (KDE). It is a special case of kernel smoothing in Sec. 6.5. The discussion of this technique is beyond the scope of this chapter.

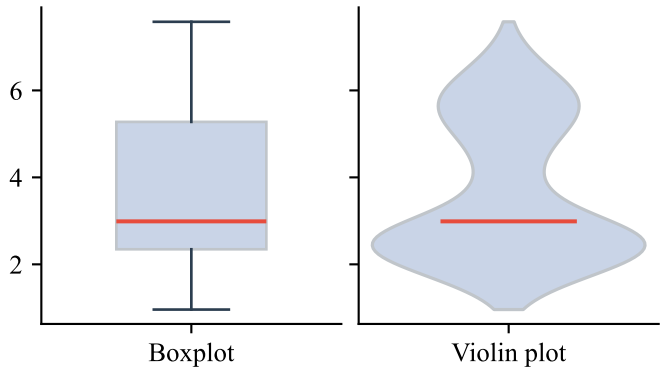


Figure 1.9.: Comparison of a boxplot and a violin plot for the same bimodal data: the boxplot hides the two-group structure, while the violin plot reveals it through its shape.

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.

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

2.1.1. Definition

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

The assumed model underlying the dataset is

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

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

The corresponding **linear model** is

$$\hat{y}_k = f(x_k; w_0, w_1) = w_0 + w_1 x_k, \tag{2.2}$$

where w_0 and w_1 are the model parameters. The corresponding model **error** is

$$e_k = y_k - \hat{y}_k = y_k - w_0 - w_1 x_k. \tag{2.3}$$

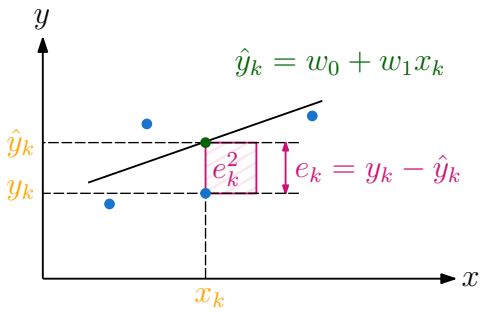


Figure 2.1.: Linear regression visualization. The goal is to minimize SSE, which represents the total area $\sum_k e_k^2$ of the rectangles.

The optimal model parameters are found by minimizing a **loss function**, $\mathcal{L}(\cdot)$. Common examples of loss functions include the sum of squared errors (SSE), mean-square error (MSE), and root-mean-square error (RMSE):

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

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

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

For interpretation, MSE is an estimator of the error variance. Because constant multiplication factors and monotonic transforms do not affect the location of the minimum, all these loss functions share the same optimal parameters:

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

For simplicity, SSE is used in the following discussion.

The minimization of the **loss** function (Fig. 2.1),

$$\mathcal{L}(w_0, w_1) = \sum_{k=1}^{M} \Big(y_k - w_0 - w_1 x_k \Big)^2, \tag{2.8}$$

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

$$\begin{cases} \dfrac{\partial}{\partial w_0} \mathcal{L}(w_0, w_1) = 0 \\ \dfrac{\partial}{\partial w_1} \mathcal{L}(w_0, w_1) = 0 \end{cases} \tag{2.9}$$

which yields:

$$\begin{cases} 2\sum_{k=1}^{M}(y_k - w_0 - w_1 x_k)\cdot(-1) = 0 \\ 2\sum_{k=1}^{M}(y_k - w_0 - w_1 x_k)\cdot(-x_k) = 0. \end{cases} \tag{2.10}$$

Finally, with some basic algebra:

$$\begin{cases} w_0 M + w_1 \sum_{k=1}^{M} x_k = \sum_{k=1}^{M} y_k, \\ w_0 \sum_{k=1}^{M} x_k + w_1 \sum_{k=1}^{M} x_k^2 = \sum_{k=1}^{M} x_k y_k. \end{cases} \tag{2.11}$$

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

2.1.2. Mean and Variance

The goal of this section is 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:

$$\hat{y} = w_0 \tag{2.12}$$

The corresponding MSE loss function is

$$\mathcal{L}(w_0) = \frac{1}{M}\sum_{k=1}^{M}(y_k - w_0)^2, \tag{2.13}$$

with the minimum at the mean of y_k values,

$$w_0 = \frac{1}{M}\sum_{k=1}^{M} y_k = \bar{y} \tag{2.14}$$

The corresponding MSE of the model (substituting (2.14) into (2.13)) is

$$\text{MSE} = \frac{1}{M}\sum_{k=1}^{M}\left(y_k - \frac{1}{M}\sum_{k=1}^{M} y_k\right)^2 = \sigma_y^2 \tag{2.15}$$

which is the *sample variance* of the y_k values [1].

To summarize, $\hat{y} = \bar{y}$ with $\text{MSE} = \sigma_y^2$.

2.2. Normal Equations with Statistical Terms

Goal: To rewrite the normal equations (2.11) using statistical terms.

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

2.2.1. Sample Covariance

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

$$s_{xy} = \frac{1}{M} \sum_{i=1}^{M} (x_i - \bar{x})\,(y_i - \bar{y}) \tag{2.16}$$

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. Normal Equations

It is numerically convenient to rewrite the normal equations (2.11) in terms of sample moments

$$\begin{aligned} w_1 &= \frac{s_{xy}}{s_x^2} = \frac{\sum_k (x_k - \bar{x})(y_k - \bar{y})}{\sum_k (x_k - \bar{x})^2}, \\ w_0 &= \bar{y} - w_1 \bar{x} = \bar{y} - \frac{s_{xy}}{s_x^2} \bar{x}, \end{aligned} \tag{2.17}$$

and the predictive form

$$\hat{y} = \bar{y} + w_1 \Big(x - \bar{x} \Big). \tag{2.18}$$

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.
- If both variables are centered ($\bar{x} = \bar{y} = 0$), then (2.17) reduces to

$$\begin{aligned} w_0 &= 0, \\ w_1 &= \frac{\sum_k x_k y_k}{\sum_k x_k^2} \end{aligned} \tag{2.19}$$

2.3. Correlation Coefficient

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

$$r_{xy} = \frac{s_{xy}}{s_x\, s_y} \tag{2.20}$$

$$= \frac{\sum_{i=1}^{M}(x_i - \bar{x})\,(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{M}(x_i - \bar{x})^2}\,\sqrt{\sum_{i=1}^{M}(y_i - \bar{y})^2}} \tag{2.21}$$

and the slope in (2.17) can be re-written as

$$w_1 = r_{xy}\,\frac{s_y}{s_x}. \tag{2.22}$$

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

2.3.1. 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:

$$z_i = \frac{x_i - \bar{x}}{s_x}, \qquad t_i = \frac{y_i - \bar{y}}{s_y}, \qquad i = 1, \ldots, M \tag{2.23}$$

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 [2]:

$$s_x^2 = \frac{1}{M}\sum_{k=1}^{M}(x_k - \bar{x})^2, \qquad s_y^2 = \frac{1}{M}\sum_{k=1}^{M}(y_k - \bar{y})^2. \tag{2.24}$$

For the means

$$\begin{aligned} \sum_{k=1}^{M} z_k &= \frac{1}{s_x}\sum_{k=1}^{M}(x_k - \bar{x}) \\ &= \frac{1}{s_x}\left(\sum_{k=1}^{M} x_k - M\bar{x}\right) \\ &= \frac{1}{s_x}(M\bar{x} - M\bar{x}) = 0 \end{aligned} \tag{2.25}$$

[2] For unbiased evaluation the factor $M - 1$ is to be used.

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

$$s_z^2 = \frac{1}{M}\sum_{k=1}^{M} z_k^2 = \frac{1}{M}\sum_{k=1}^{M} \frac{(x_k - \bar{x})^2}{s_x^2} = \frac{1}{s_x^2}\left[\frac{1}{M}\sum_{k=1}^{M}(x_k - \bar{x})^2\right] = \frac{s_x^2}{s_x^2} = 1. \tag{2.26}$$

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

Normal Equations The linear model in normalized space,

$$\hat{t}_i = w_0^* + w_1^* z_i \tag{2.27}$$

yields (applying (2.19))

$$w_0^* = 0, \tag{2.28}$$

$$w_1^* = \frac{s_{zt}}{s_z^2} = \frac{\frac{1}{M}\sum_{i=1}^{M} z_i t_i}{\frac{1}{M}\sum_{i=1}^{M} z_i^2} = \frac{1}{M}\sum_{i=1}^{M} z_i t_i = \frac{1}{s_x s_y}\frac{1}{M}\sum_{i=1}^{M}(x_i - \bar{x})(y_i - \bar{y}) = r_{xy} \tag{2.29}$$

Thus, the normalized prediction is

$$\hat{t}_i = r_{xy}\, z_i. \tag{2.30}$$

where

$$r_{xy} = \frac{s_{xy}}{s_x\, s_y} \tag{2.31}$$

is the correlation coefficient. Returning to the original scale gives

$$\frac{\hat{y} - \bar{y}}{s_y} = r_{xy}\frac{x - \bar{x}}{s_x}$$

or the familiar form:

$$\hat{y} = \bar{y} + r_{xy}\frac{s_y}{s_x}\Big(x - \bar{x}\Big).$$

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 corresponding dot product is

  $$\mathbf{z} \cdot \mathbf{t} = \|\mathbf{z}\|\|\mathbf{t}\| \cos(\theta) \tag{2.32}$$

  Using (2.29),

  $$\mathbf{z} \cdot \mathbf{t} = \sum_{i=1}^{M} z_i t_i = M r_{xy}. \tag{2.33}$$

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

  $$\|\mathbf{z}\|\|\mathbf{t}\| = \sqrt{M} \cdot \sqrt{M} = M \tag{2.34}$$

  Finally, $r_{xy} = \cos\theta$, where θ is the angle between the centered and standardized data vectors $\mathbf{z}$ and $\mathbf{t}$ (Fig. 2.2).
  Consequences:

- Range: $-1 \leq r_{xy} \leq +1$.
- Special case of perfect linear fit (zero error)

$$r_{xy} = \pm 1 \implies \theta = 0^\circ \text{ or } 180^\circ$$
$$r_{xy} = 0 \implies \theta = 90^\circ$$

- $r_{xy} = +1$ indicates perfect positive linear association.
- $r_{xy} = -1$ indicates perfect negative linear association.
- $r_{xy} = 0$ indicates no linear association.

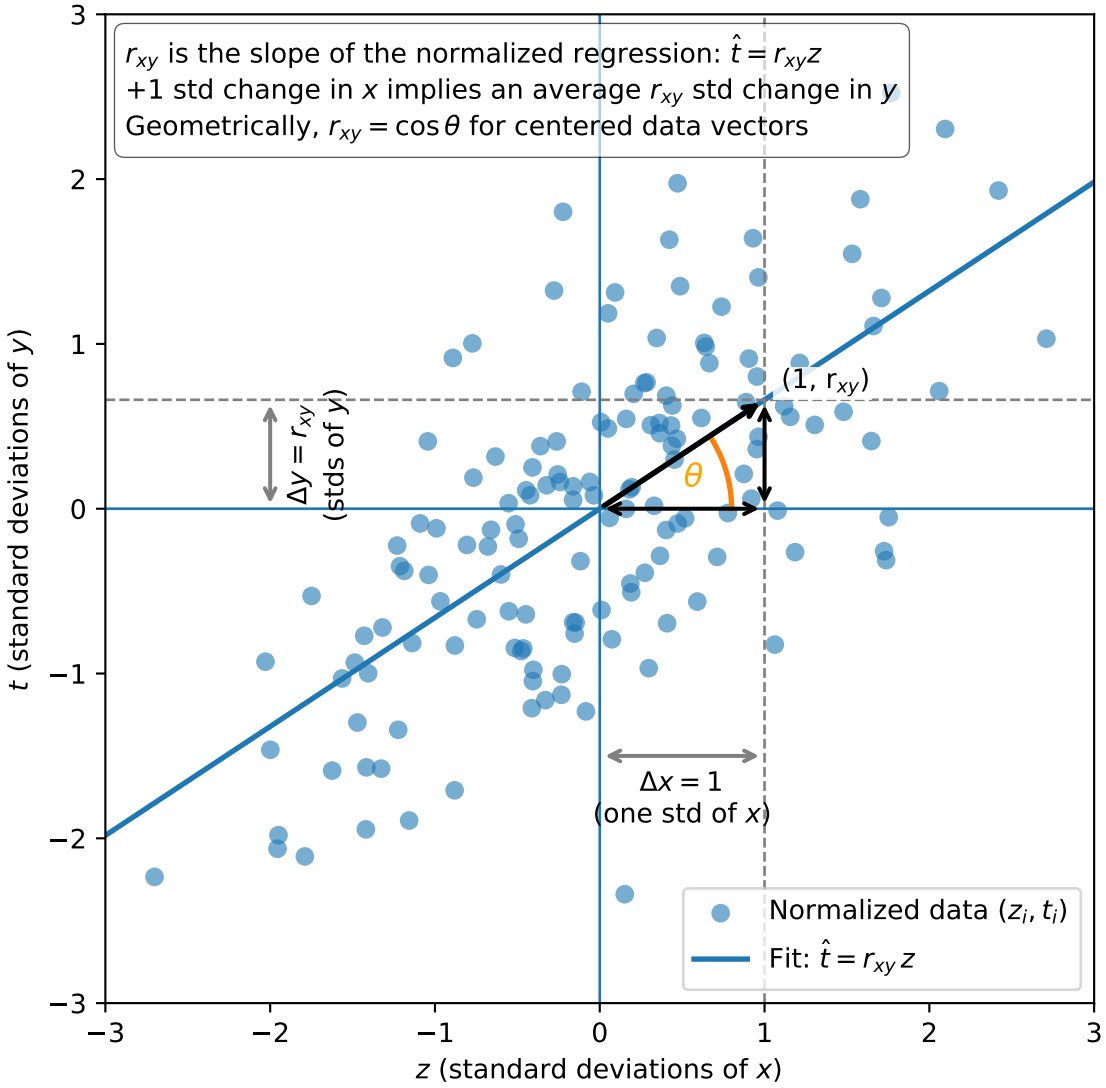


Figure 2.2.: Geometric interpretation of r_{xy} in a normalized LR.

Regression error as a function of r_{xy}

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

$$e_i = t_i - \hat{t}_i = t_i - r_{xy} z_i \tag{2.35}$$

with

$$\text{MSE} = 1 - r_{xy}^2 = s_e^2. \tag{2.36}$$

Proof.

$$\begin{aligned} s_e &= \frac{1}{M}\sum_{i=1}^{M} e_i^2 = \frac{1}{M}\sum_{i=1}^{M}\left(t_i - r_{xy} z_i\right)^2 \\ &= \frac{1}{M}\sum_{i=1}^{M} t_i^2 - 2r_{xy}\frac{1}{M}\sum_{i=1}^{M} t_i z_i + r_{xy}^2 \frac{1}{M}\sum_{i=1}^{M} z_i^2 \\ &= s_t^2 - 2r_{xy}\left(r_{xy}\right) + r_{xy}^2 s_z^2 \\ &= 1 - r_{xy}^2 \end{aligned} \tag{2.37}$$

Consequences:

- $r_{xy} = 0 \Rightarrow$ the best linear predictor is $\hat{y} = \bar{y}$ and the MSE equals the sample variance of y. There is no linear association and the MSE is maximal.
- $|r_{xy}| = 1 \Rightarrow$ perfect fit, zero residual error. It means that the data points lie perfectly on a straight line.
- Typically, the residual errors are (approximately) Gaussian distributed.

Note, the unbiased estimator of s_e^2 requires $1/(M-2)$ factor instead $1/M$.

2.3.2. Coefficient of Determination Metric

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

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

$$\underbrace{\sum_{k=1}^{M}(y_k - \bar{y})^2}_{\text{SST}} = \underbrace{\sum_{k=1}^{M}(\hat{y}_k - \bar{y})^2}_{\text{SSR}} + \underbrace{\sum_{k=1}^{M} e_k^2}_{\text{SSE}}, \tag{2.38}$$

where:

- **SST**: Total sum of squares.
- **SSR**: Sum of squares due to regression.
- **SSE**: Sum of squared errors (or residual sum of squares).

Proof. 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 $\sum_{k=1}^{M} \hat{y}_k\, e_k = 0$.

The R^2 metric is defined as:

$$R^2 = \frac{\text{SSR}}{\text{SST}} = 1 - \frac{\text{SSE}}{\text{SST}}, \tag{2.39}$$

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).

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

$$R^2 = r_{xy}^2. \tag{2.40}$$

Proof. Starting with $\hat{y}_i = w_0 + w_1 x_i$:

$$\begin{aligned} SSR &= \sum_{i=1}^{M}(\hat{y}_i - \bar{y})^2 \\ &= \sum_{i=1}^{M}(w_0 + w_1 x_i - \bar{y})^2 \\ &= \sum_{i=1}^{M}(\bar{y} - w_1\bar{x} + w_1 x_i - \bar{y})^2 \\ &= w_1^2 \sum_{i=1}^{M}(x_i - \bar{x})^2 \\ &= w_1^2 M s_x^2 = M\left(\frac{s_{xy}}{s_x^2}\right)^2 s_x^2 = M\frac{s_{xy}^2}{s_x^2}, \\ SST &= \sum_{k=1}^{M}(y_k - \bar{y})^2 = M s_y^2 \end{aligned} \tag{2.41}$$

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

The general (biased) MSE of unnormalized values is

$$\begin{aligned} \text{MSE} &= \frac{1}{M}\sum_{k=1}^{M}\Big(y_k - \hat{y}_k\Big)^2 \\ &= s_y^2\Big(1 - r_{xy}^2\Big) \\ &= \frac{1}{M}\text{SSE} = \frac{1}{M}\left(\text{SST} - \text{SSR}\right). \end{aligned} \tag{2.42}$$

3. Linear Least-Squares

Goals:re-write LS in the matrix notation and extend it to the multi-variate.

Contents

3.1. Uni-variate LS

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 corresponding SSE loss function (2.4) is

$$\begin{aligned} \mathcal{L}(\mathbf{w}; \mathbf{X}, \mathbf{y}) &= (\mathbf{y} - \hat{\mathbf{y}})^T (\mathbf{y} - \hat{\mathbf{y}}) = \|\mathbf{y} - \hat{\mathbf{y}}\|^2 \\ &= (\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w}) = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 \end{aligned} \tag{3.3}$$

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

$$\nabla_{\mathbf{w}}\mathcal{L}(\cdot) = -\mathbf{X}^T(\mathbf{y} - \mathbf{X}\mathbf{w}) = 0 \tag{3.4}$$

and is given by

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

Finally,

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

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

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

The model error is

$$\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}} \tag{3.6}$$

such that SSE is $\mathcal{L}(\mathbf{w}) = \mathbf{e}^T\mathbf{e}$.

3.2. Multivariate LS

3.2.1. Notation

For the multivariate N-dimensional formulation,

$$\mathbf{X} = \begin{bmatrix} \mathbf{1} & \mathbf{x}_1 & \cdots & \mathbf{x}_N \end{bmatrix} \in \mathbb{R}^{M\times(N+1)} \tag{3.7}$$

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

Each vector $\mathbf{x}_i$ is also termed *feature* vector.

All the LS discussion on $\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}$ is the same independently from the number of feature vectors.

Dataset

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

3.2.2. Adjusted Coefficient of Determination

Goal: Provide a comparison metric for multivariate linear models that accounts for different numbers or subsets of features.

While Section 2.3.2 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 use a penalty term based on the number of features N relative to the number of samples M. It is given by:

$$\bar{R}^2 = 1 - \left(1 - R^2\right) \frac{M-1}{M-N-1} \tag{3.9}$$

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.

3.3. Properties

3.3.1. Moore–Penrose inverse (pseudo-inverse)

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:

$$\mathbf{X}^+ = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T \in \mathbb{R}^{(N+1)\times M}, \tag{3.10}$$

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 thata 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 for LS can be compactly written as:

$$\mathbf{w}_{opt} = \mathbf{X}^+\mathbf{y} \tag{3.11}$$

3.3.2. Projection matrix

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

$$\begin{aligned}\hat{\mathbf{y}} &= \mathbf{X}\mathbf{w} = \mathbf{X}\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y} \\ &= \mathbf{X}\mathbf{X}^+\mathbf{y} = \mathbf{P}\mathbf{y}\end{aligned} \tag{3.12}$$

where

$$\mathbf{P} = \mathbf{X}\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T \in \mathbb{R}^{M\times M} \tag{3.13}$$

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.
$$\begin{aligned}\mathbf{P}^2 &= \mathbf{X}\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{X}\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T \\ &= \mathbf{X}\left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T = \mathbf{P}\end{aligned} \tag{3.14}$$
- 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 $\mathbf{X}$
$$\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}} = \mathbf{y} - \mathbf{P}\mathbf{y} = (\mathbf{I} - \mathbf{P})\,\mathbf{y}. \tag{3.15}$$

3.3.3. Error properties

Error and feature orthogonality

$$\mathbf{e} \perp \mathbf{X} \Rightarrow \mathbf{X}^T\mathbf{e} = \mathbf{0}_{N+1} \tag{3.16}$$

Proof. Substituting the projection matrix notation:

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

Error and prediction orthogonality

$$\mathbf{e} \perp \hat{\mathbf{y}} \Rightarrow \hat{\mathbf{y}}^T\mathbf{e} = \mathbf{e}^T\hat{\mathbf{y}} = 0 \tag{3.17}$$

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,

$$\|\mathbf{y}\|^2 = \|\hat{\mathbf{y}}\|^2 + \|\mathbf{e}\|^2 \tag{3.18}$$

Proof.

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

Average error

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

$$\begin{aligned}\bar{\mathbf{e}} &= \frac{1}{M}\sum_{k=1}^{M} e_k \\ &= \sum_{k=1}^{M} e_k = \mathbf{1}^T\mathbf{e} = 0\end{aligned} \tag{3.19}$$

Proof. From Eq. (3.16), 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),

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

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

$$\begin{aligned}\bar{\mathbf{y}} &= \bar{\hat{\mathbf{y}}} \\ &= w_0 + w_1\bar{\mathbf{x}}_1 + \cdots + w_N\bar{\mathbf{x}}_N\end{aligned} \tag{3.20}$$

Proof.

$$\begin{aligned}\bar{\mathbf{y}} &= \overline{\hat{\mathbf{y}} + \mathbf{e}} \\ &= \bar{\hat{\mathbf{y}}} + \bar{\mathbf{e}}\end{aligned} \tag{3.21}$$

Following this property, $\bar{\mathbf{y}} = 0$ follows $w_0 = 0$.

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 need in ML, but it is important for statistical analysis for small values of M.

Minimum SSE

The reduced expression for the resulting minimal loss is

$$\begin{aligned}\mathcal{L}_{min} &= \sum_{k=1}^{M} y_k^2 - \sum_{j=0}^{N} w_j \mathbf{y}^T \mathbf{x}_j \\ &= \mathbf{y}^T\mathbf{y} - \mathbf{y}^T\mathbf{X}\mathbf{w}\end{aligned} \tag{3.22}$$

Proof.

$$\begin{aligned}sse_{min} &= \mathbf{e}^T\mathbf{e} \\ &= (\mathbf{y} - \hat{\mathbf{y}})^T\mathbf{e} \\ &= \mathbf{y}^T\mathbf{e} - \hat{\mathbf{y}}^T\mathbf{e} \qquad \leftarrow (3.17) \\ &= \mathbf{y}^T\mathbf{e} \\ &= \mathbf{y}^T(\mathbf{y} - \mathbf{X}\mathbf{w})\end{aligned} \tag{3.23}$$

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: Find the minimum of the function:

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

Preface

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

- 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,

$$f(x + \epsilon) \approx f(x) + \epsilon f'(x) \tag{3.24a}$$
$$f(x - \epsilon) \approx f(x) - \epsilon f'(x) \tag{3.24b}$$

- Given the sign of the derivative,

$$\begin{aligned} f(x-\epsilon) < f(x), \quad f'(x) > 0 \\ f(x+\epsilon) < f(x), \quad f'(x) < 0 \\ f(x+\epsilon) > f(x), \quad f'(x) > 0 \\ f(x-\epsilon) > f(x), \quad f'(x) < 0 \end{aligned} \tag{3.25}$$

- For sufficiently small ϵ,

$$f\left(x - \epsilon \operatorname{sign}\left(f'(x)\right)\right) \leq f(x) \tag{3.26}$$

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 (3.26).

add illustration

3.4.1. Definition

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

$$x_{n+1} = x_n - \alpha f'(x_n) \tag{3.27}$$

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

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \alpha \nabla_{\mathbf{x}} f(\mathbf{x}_n). \tag{3.28}$$

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 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,

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \alpha \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}) \tag{3.29}$$

GD for LS

Optimal values of $\mathbf{w}$ may be found by substitution of the gradient (Eq. (3.4) into (3.29)),

$$\begin{aligned}\mathbf{w}_{n+1} &= \mathbf{w}_n - \alpha \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}) \\ &= \mathbf{w}_n - \frac{\alpha}{M} \mathbf{X}^T \left(\mathbf{X}\mathbf{w}_n - \mathbf{y}\right) \\ &= \mathbf{w}_n \left(\mathbf{I} - \frac{\alpha}{M} \mathbf{X}^T \mathbf{X}\right) + \frac{\alpha}{M} \mathbf{X}^T \mathbf{y}\end{aligned} \tag{3.30}$$

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

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

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

Lets define each row of the dataset as $\mathbf{X}_i$, such that 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,

$$\mathcal{L}(\mathbf{w}; \mathbf{X}, \mathbf{y}) = \sum_{k=1}^{M} \ell_k(\mathbf{w}; \mathbf{X}_k, y_k), \tag{3.31}$$

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

For the LS,

$$\ell_k(\mathbf{w}) = \left(y_k - \mathbf{X}_k^T \mathbf{w}\right)^2. \tag{3.32}$$

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 (3.30).

Mini-batch GD

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 $\mathcal{B}_n$,

$$\mathbf{X}_{\mathcal{B}_n} \in \mathscr{R}^{B \times (N+1)}, \qquad \mathbf{y}_{\mathcal{B}_n} \in \mathscr{R}^B, \tag{3.33}$$

The mini-batch loss is

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

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

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

where B is called the *batch size*.

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.

3.4.3. Practical aspects

Theoretical relation to the full gradient

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. 3.1.

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 (with B between a few tens and a few hundreds) is the most commonly used compromise between speed and stability.

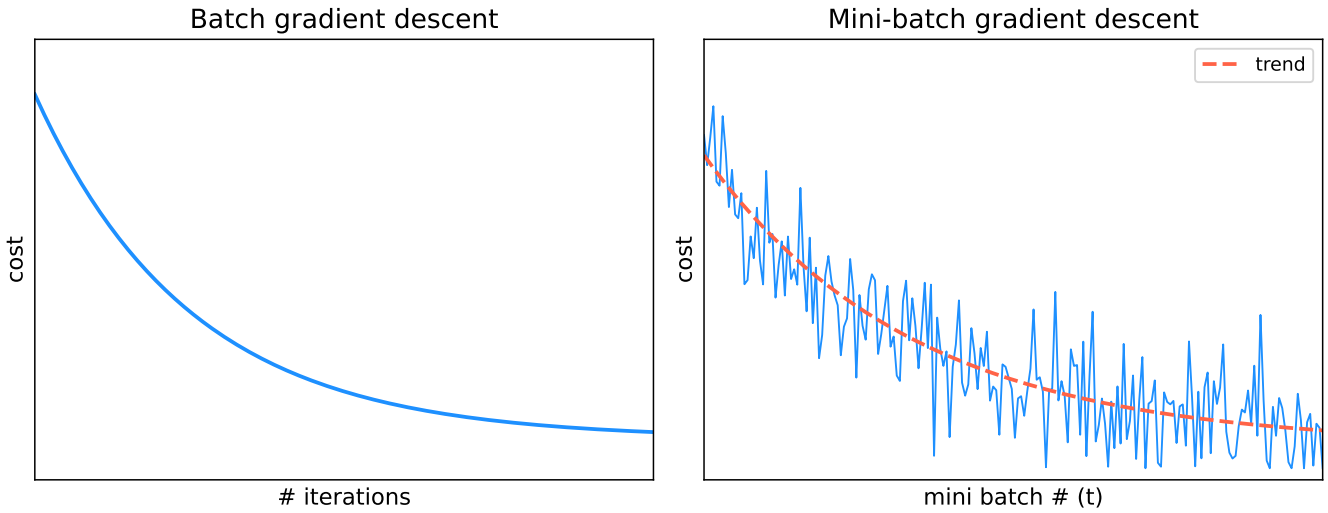


Figure 3.1.: Mini-batch gradient noise: smaller batches produce noisier loss trajectories but can escape local minima.

4. Model Characterization

Contents

4.1. Uni-variate Polynomial Model

The goal is to extend a linear model "engine" to polynomial models. The polynomial model is very flexible, e.g. due to the Taylor expansion theorem.

The L-degree uni-variate polynomial regression model is

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

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),

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

The weights vector values are straightforward.

Hyper-parameter The value of L is termed hyper-parameter of the model.

Examples of questions of the dramatic importance:
- Is a model $\hat{y} = f(x; \mathbf{w})$ is appropriate?
- How do we evaluate a model performance?
- What are the optimal hyper-parameter values, e.g. L of the polynomial model?

4.2. Generalization

Goal: The goal is:

- Estimate some metrics of the model and understand the limitations on this estimate.
- Trial and error approach.
- Understand the limitations on the effectiveness of the model's ability to make inferences on unfamiliar 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 performance: The difference between:

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

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 $\bar{J}_{theory}$.

Notes:

- Better generalization means smaller difference between model performance and generalization performance.
- Can be evaluated theoretically only on very simple models and datasets.

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.

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

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.

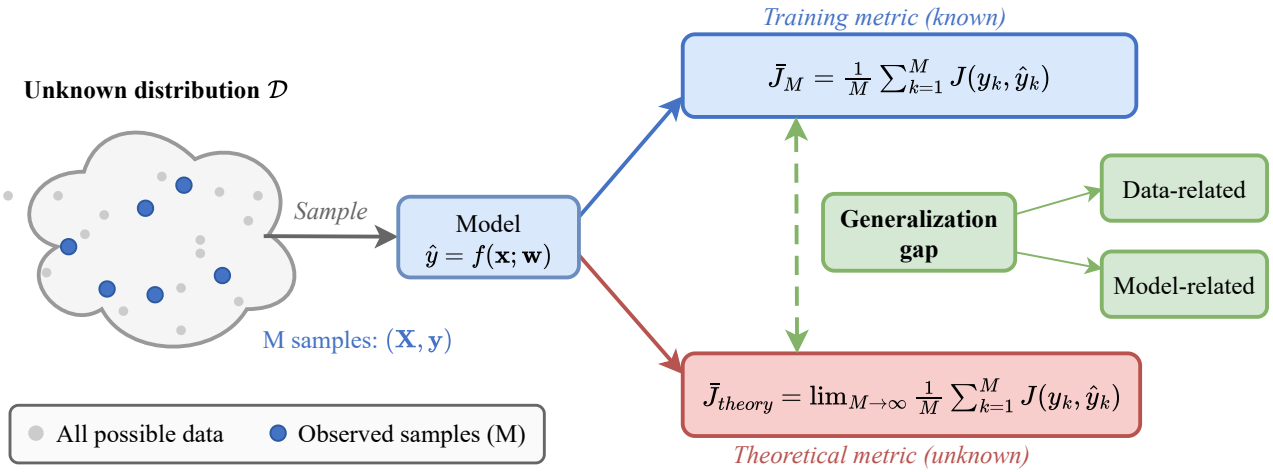


Figure 4.1.: Generalization: the gap between the training metric $\bar{J}_M$ (computed on M observed samples) and the theoretical metric $\bar{J}_{theory}$ (over the entire unknown distribution $\mathcal{D}$ of possible inputs). The gap has data-related and model-related sources.

4.3. 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.

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

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

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

Medium datasets $\left(10^2 \lesssim M \lesssim 10^4\right)$: k-fold

First step is to resample the dataset into the **random order**. Then, apply the following steps:

- *Data Splitting*: First, the available dataset is divided into k subsets of approximately equal size. These subsets are often referred to as "folds".

- *Model Training and Evaluation*: The model is trained k times. In each iteration, one of the subsets is used as the test set, and the remaining $k-1$ subsets are used as the training/validation sets. This means that in each iteration, the model is trained and validated on a different combination of training and test data.
- *Performance Evaluation*: After training the model k times, the performance of the model is evaluated by averaging the performance metrics obtained in each iteration.

Usually, k is defaulted to 5 or 10.

Very small datasets $\left(M \lesssim 10^2\right)$: one-hold-out

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

The cross-validation methods are illustrated in Fig. 4.2.

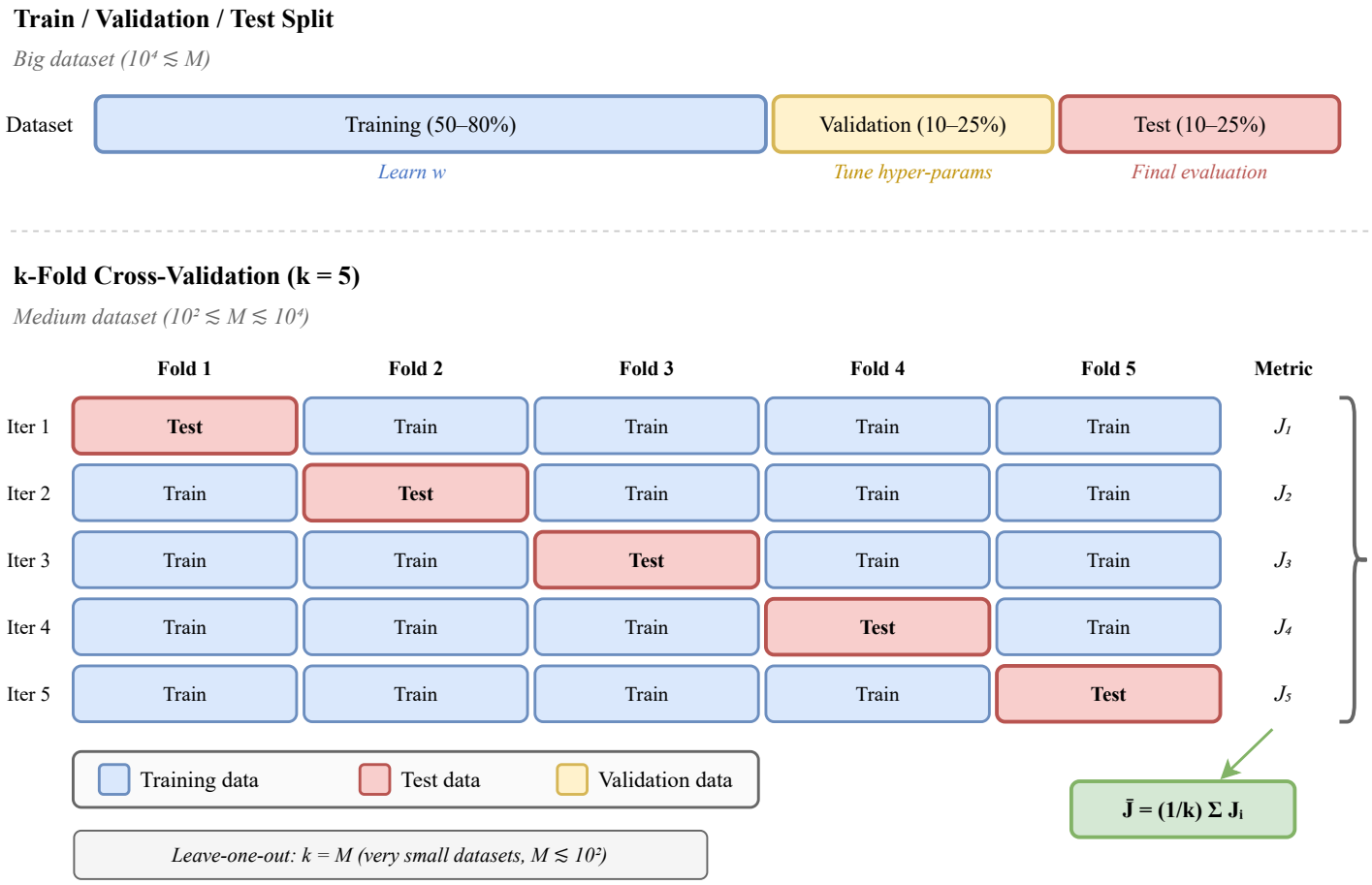


Figure 4.2.: Cross-validation illustration: train/validation/test split for big datasets (top) and k-fold cross-validation for medium datasets (bottom).

4.4. Basic Workflow

The basic ML/DL workflow is presented in Fig. 4.3. The workflow parts are:

- Goal definition
- Data: available data
- Pre-processing: preliminary dataset exploration and validation of dataset integrity (e.g., same physical units for all values of the same feature).
- Model: basic assumptions about the hidden pattern within the data
- Model training: minimization of the loss functions to derive the most appropriate parameters.
- Hyper-parameter optimization: tuning the model sub-type.
- Performance assessment according predefined metrics.

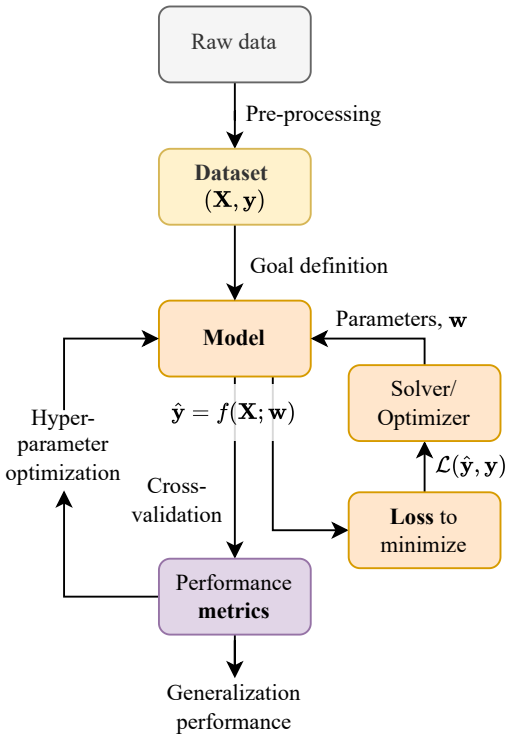


Figure 4.3.: Basic workflow of ML/DL solution.

Baseline The basic end-to-end workflow implementation is called baseline.

4.4.1. Goal definition

Typical related goals:

- Prediction or regression, $\mathbf{y}$ is quantitative (Fig. 4.4a).
- Classification, $\mathbf{y}$ is categorical (Fig. 4.4b).
- Clustering, no $\mathbf{y}$ is provided - it is learned from dataset (Fig. 4.4c).
- Semi-supervised learning, combination of labeled and unlabeled data (Fig. 4.4d).
- Anomaly detection, somewhere between classification and clustering (Fig. 4.4e).
- Segmentation
- Simulation
- Signal processing tasks: noise removal, smoothing (filling missing values), event/condition detection.

Note, there is possible to have two or more goals for the same dataset.

4.4.2. Model

We assume that there is an underling problem (e.g., regression and classification) formulation is of the form

$$y = h(\mathbf{x}) + \epsilon \tag{4.5}$$

where $h(\mathbf{x})$ is the true unknown function and ϵ is some irreducible noise. Sometimes, zero-mean noise is assumed. The values of $\mathbf{x}$ (scalar or vector) and y are known (it is the dataset).

The goal is to find the function $f(\cdot; \mathbf{w})$ that approximates $h(\mathbf{x})$. The way to define the $f(\cdot; \mathbf{w})$ is termed *model* that depends on some model parameters vector $\mathbf{w}$. The process of finding parameters $\mathbf{w}$ is called learning, such as the resulting model can provides predictions

$$\hat{y}_0 = f(\mathbf{x}_0; \mathbf{w}) \tag{4.6}$$

for some new data $\mathbf{x}_0$.

Parameters vs hyper-parameters

There is a conceptual difference between parameters and hyper-parameters.

Model parameters: Model parameters are learned directly from a dataset.

Hyper-parameters: Model parameters that are not learned directly from a dataset are called hyper-parameters. They are learned in in-direct way during cross-validation process in the follow.

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

Parametric vs non-parametric models

There are two main classes of models: parametric and non-parametric, summarized in Table 4.1. The modern trend is to bridge the gap between interpretable and non-parameteric modeling.

4.4.3. Loss Function

Loss (or cost) function is a function that relates between dataset outputs $\mathbf{y}$ and model outputs $\hat{\mathbf{y}}$. The parameters $\mathbf{w}$ are minimum of that function,

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) \tag{4.7}$$

The minimization of the loss function is also termed *training*.

Loss Function Minimization

Goal: Minimum of the loss function for a given model.

Table 4.1.: Comparison of parametric and non-parametric models.

Aspect	Parametric	Non-parametric
Dependence on 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

Closed-form solution A closed-form solution for $\mathbf{w}$ is a solution that is based on basic mathematical functions. For example, a "normal equation" is a solution for linear regression/classification.

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

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

Global optimizers The goal of global optimizers is to find a global minimum of non-convex function. These algorithms may be gradient-free, first-derivative or second-derivative. The complexity of these algorithms is significantly higher than the local optimizer and can be prohibitive for more than a few hundred variables in $\mathbf{X}$.

4.4.4. Metrics

Metrics are quantitative performance indicator $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$ of the model that relate between $\mathbf{y}$ and $\hat{\mathbf{y}}$. Sometimes, the minimum of the loss function is also a metric, e.g. mean squared error (MSE).

4.5. Project Steps

A widely adopted framework for ML/DL projects is the Cross-Industry Standard Process for Data Mining (CRISP-DM), which defines six iterative phases:

1. **Business understanding** – define the problem, success criteria, and project plan.

2. **Data understanding** – collect initial data and perform exploratory analysis.
3. **Data preparation** – clean, transform, and construct the final dataset.
4. **Modeling** – select and train candidate models.
5. **Evaluation** – assess model performance against business objectives.
6. **Deployment** – integrate the model into production.

The process is iterative: findings in any phase may require revisiting earlier phases.

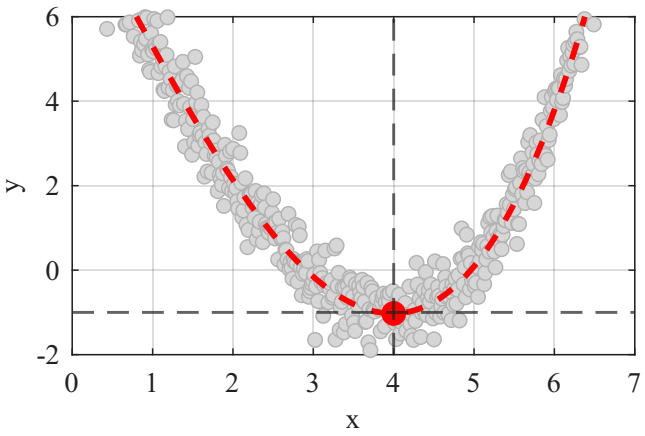
Practical guidelines

- *Understanding the problem*: domain knowledge is essential.
  - Define performance metric and performance goal.
  - Evaluate human-level performance, if relevant.
- *Data collection and preparation*: typically the most time-consuming step.
  - Ensure representative distribution.
  - Identify outliers.
  - Compute basic statistics and create visualizations.
  - Verify sufficient dataset size.
- *Model engineering*:
  - Start with a standard, pre-implemented model.
  - Avoid overly complex models for the baseline.
- *Baseline implementation*:
  - Build an end-to-end pipeline: data $\rightarrow$ model $\rightarrow$ loss $\rightarrow$ metrics.
  - Debug and verify a sufficient level of performance.
- *Performance evaluation*:
  - Identify overfitting and underfitting (bias/variance analysis).
  - Investigate errors and validate dataset integrity.
- *Performance improvement*:
  - Hyper-parameter optimization.
  - Iteratively refine: revisit the problem, add data, or improve the model until sufficient performance is achieved.
- *Model deployment*:
  - Deploy the model for the target application.

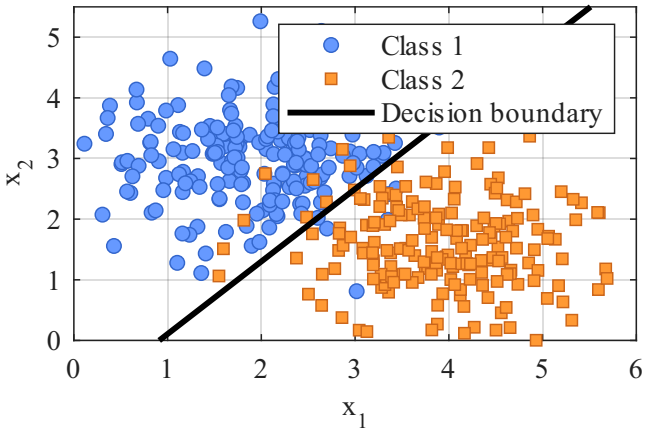
4.5.1. MLOps

Machine Learning Operations (MLOps) extends DevOps principles to ML systems, bridging the gap between model development and production deployment.

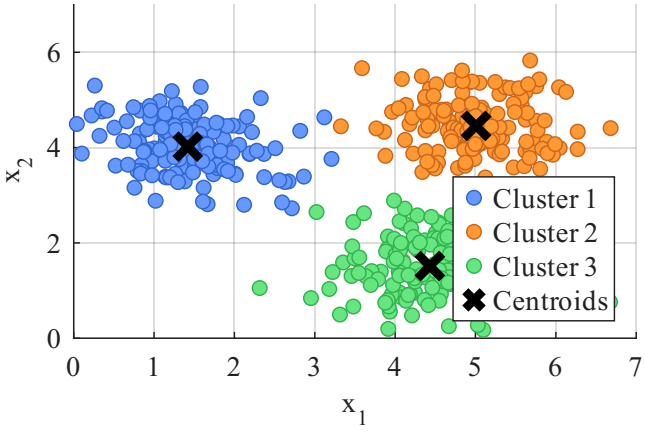
- **Develop**: algorithm training and testing, ETL / data pipelines, continuous integration and deployment (CI/CD).
- **Operate**: continuous delivery, model inference, monitoring and management.



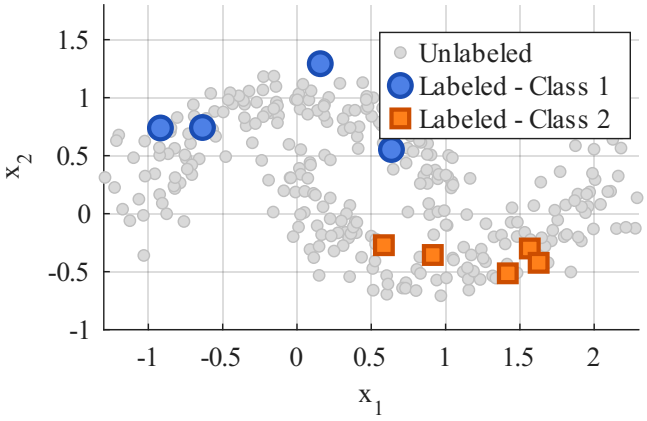
(a) Regression: what is the value of y for given x?



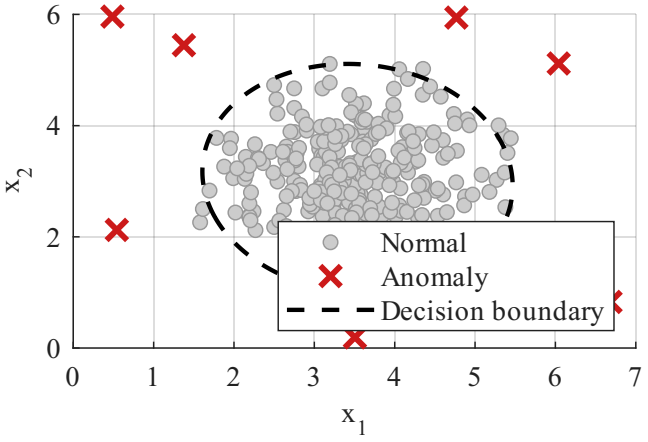
(b) Classification: what is the class of a new sample?



(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 4.4.: Examples of common ML goal types.

5. Overfitting and underfitting

Goal:
- Interpret cross-validation results.
- Identify fundamental trade-offs and/or relations between:
  - Model complexity and generalization performance.
  - Model complexity and the size of dataset.

5.1. Definition

Overfitting when 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 datasets.

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 complimentary 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. 5.1

The numerical example of underfitting and overfitting trade-off from the polynomial model (Sec. 4.1) is presented in Fig. 5.2.

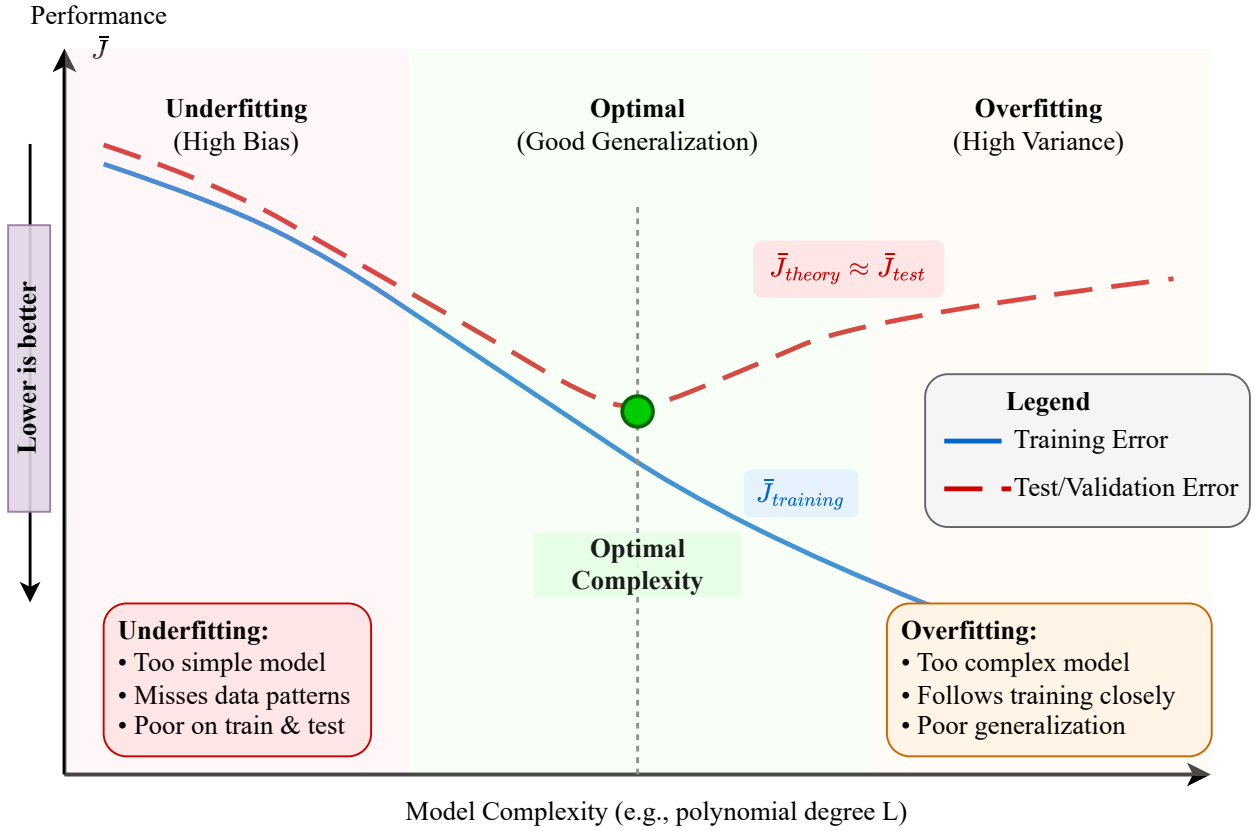


Figure 5.1.: Increasing model complexity improves the prediction error for training data, but from a certain point of transitioning from underfitting to overfitting, it increases the error for test data.

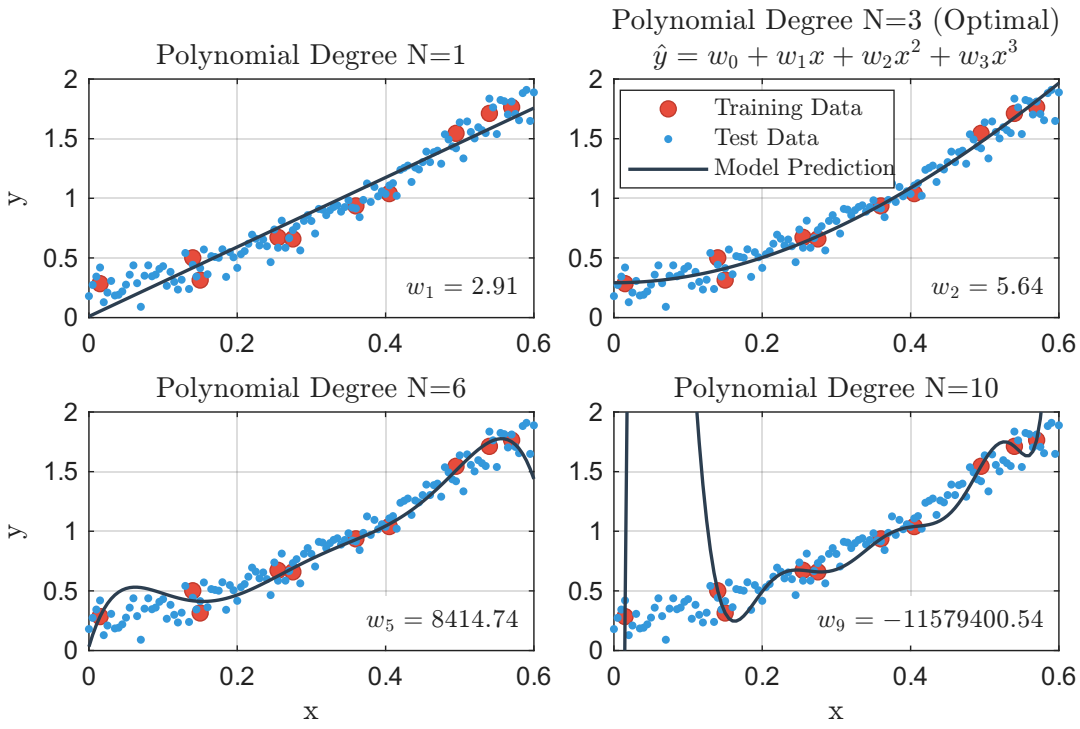


Figure 5.2.: Overfitting and underfitting polynomial example.

5.2. Bias-Variance Trade-off

Goal: Inherent underfitting and overfitting trade-off.

5.2.1. Noisy Deterministic Function Interpretation

The general model beneath the dataset model is

$$y_i = h(\mathbf{x}_i) + \epsilon_i, \tag{5.1}$$

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

A model predicts outputs via $\hat{y}_i = f(\mathbf{x}_i; \boldsymbol{\theta})$.

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

$$\mathrm{Bias}[\hat{y}] = \mathbb{E}[\hat{y}] - \mathbb{E}[y] = \mathbb{E}\big[f(\mathbf{x}; \mathbf{w})\big] - h(\mathbf{x}) \tag{5.2}$$

where the expectation is taken over all possible training samples.

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

$$\mathrm{Bias}_{\text{empirical}} = \frac{1}{M}\sum_{i=1}^{M} \hat{y}_i - \frac{1}{M}\sum_{i=1}^{M} h(\mathbf{x}_i) \tag{5.3}$$

The bias and variance are illustrated in Fig. 5.3.

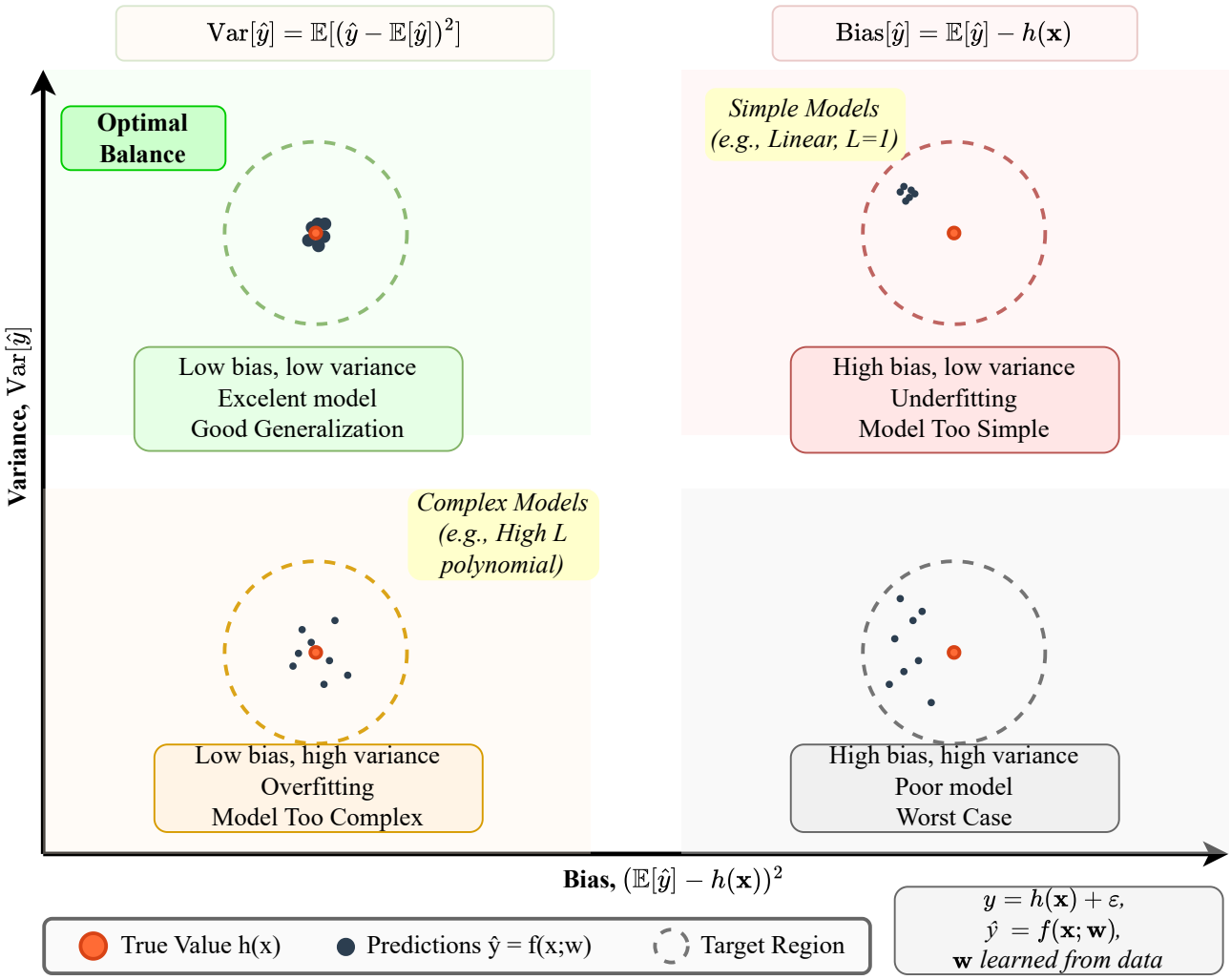


Figure 5.3.: Illustration of bias and variance.

5.2.2. Bias and Variance Trade-off

The expected prediction error (MSE) for a new point can be decomposed as:

$$
\begin{aligned}
\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) &= \mathbb{E}\Big[(\hat{y} - y)^2\Big] = \frac{1}{M}\sum_{i=1}^{M}(\hat{y}_i - y_i)^2 \\
&= \mathbb{E}\Big[\hat{y}^2\Big] - 2\mathbb{E}[y]\,\mathbb{E}[\hat{y}] + \mathbb{E}\Big[y^2\Big] \\
\mathbb{E}\Big[\hat{y}^2\Big] &= \mathrm{Var}[\hat{y}] + \mathbb{E}^2[\hat{y}] \\
\mathbb{E}[y] &= \mathbb{E}\big[h(\mathbf{x})\big] = h(\mathbf{x}) \\
\mathbb{E}\Big[y^2\Big] &= \mathbb{E}\Big[(h(\mathbf{x}) + \epsilon)^2\Big] = \frac{1}{M}\sum_{i=1}^{M}\big(h(\mathbf{x}_i) + \epsilon_i\big)^2 \\
&= \mathbb{E}\Big[h^2(\mathbf{x})\Big] + 2\mathbb{E}\big[h(\mathbf{x})\big]\,\cancelto{0}{\mathbb{E}[\epsilon]} + \mathbb{E}\Big[\epsilon^2\Big] \\
&= \mathbb{E}\Big[h^2(\mathbf{x})\Big] + \sigma^2 \\
&= h^2(\mathbf{x}) + \sigma^2 \\
\mathcal{L} &= \mathrm{Var}[\hat{y}] + \mathbb{E}^2[\hat{y}] - 2h(\mathbf{x})\mathbb{E}[\hat{y}] + h^2(\mathbf{x}) + \sigma^2 \\
&= \underbrace{\big(\mathbb{E}[\hat{y}] - h(\mathbf{x})\big)^2}_{bias^2} + \underbrace{\mathrm{Var}[\hat{y}]}_{variance} + \underbrace{\sigma^2}_{noise}
\end{aligned}
\tag{5.4}
$$

This decomposition shows that the total prediction error consists of three components: squared bias, variance, and irreducible noise. The inherent bias-variance trade-off is presented in Fig. 5.4. Underfitting is low variance and high bias, and overfitting is high variance and low bias. The best model performance has inherent bias-variance trade-off. However, some less appropriate models can have high bias and high variance simultaneously

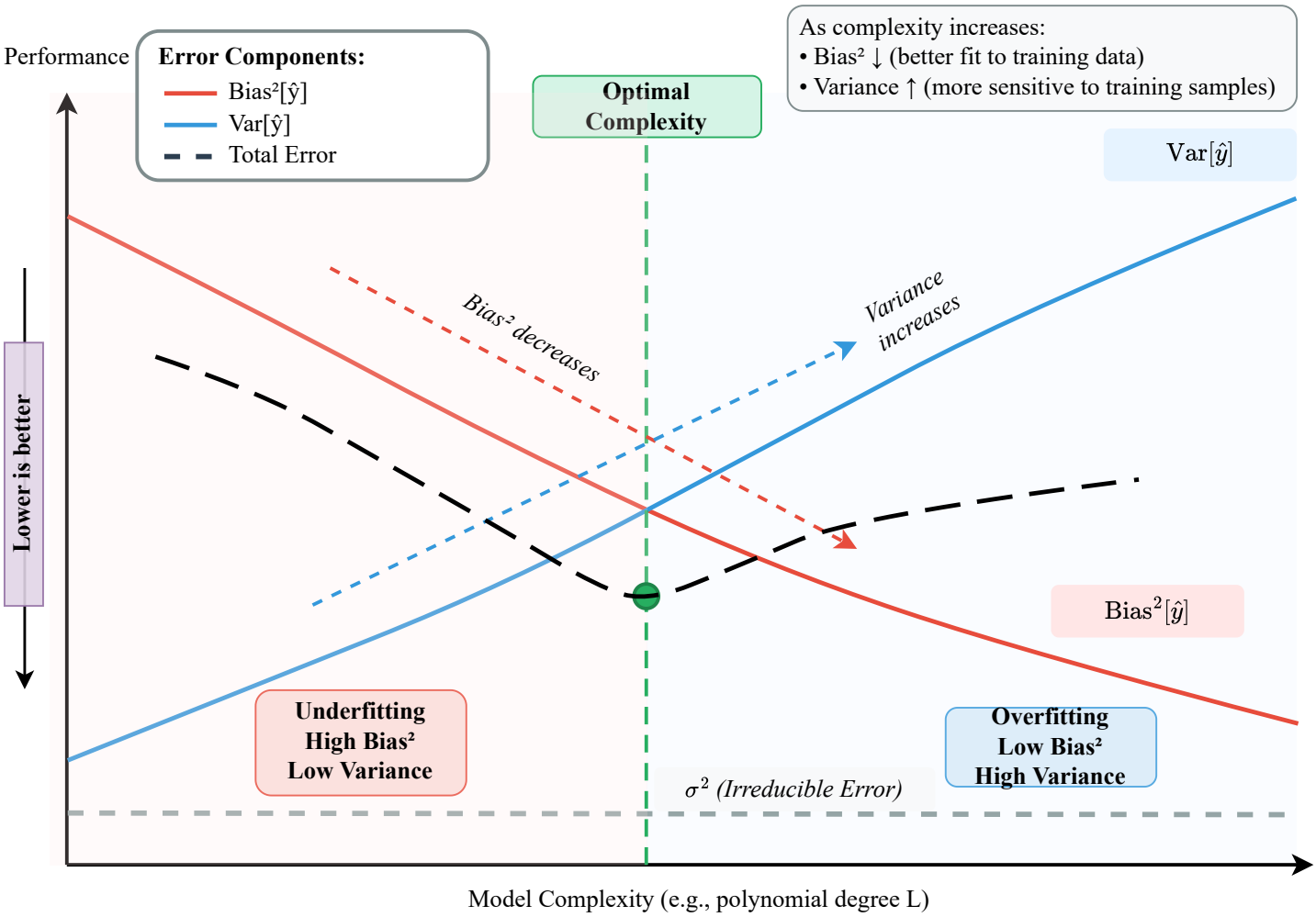


Figure 5.4.: Illustration of bias and variance.

5.3. Normalization and Standardization

Goal: Data pre-processing to reduce possible overfitting.

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

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

Standardization (z-score normalization)

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

$$z_{std} = \frac{\mathbf{x} - \bar{\mathbf{x}}}{\sigma_{\mathbf{x}}}, \tag{5.5}$$

where

$$\bar{\mathbf{x}} = \frac{1}{M}\sum_{j=1}^{M} x_j$$
$$\sigma_{\mathbf{x}}^2 = \frac{1}{M}\sum_{j=1}^{M} \left(x_j - \bar{\mathbf{x}}\right)^2$$

Implementation steps:

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, $\hat{\mathbf{y}} = \hat{\mathbf{y}}_{std}\sigma_{\mathbf{y}} + \bar{\mathbf{y}}$.

Example: `zscore` command in Matlab and `sklearn.preprocessing.StandardScaler` in Python.

Normalization

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

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{5.6}$$

Implementation steps for normalization are similar to standardization.

> Beware, normalization and standardization are used interchangeably.

Pros:

[1] In Python: `sklearn.preprocessing.MinMaxScaler`

- 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.

5.4. Dataset size

Goal: Dataset size influence on underfitting-overfitting trade-off.

Adding data usually improves the performance for an overly complex system as presented in Fig. 5.5.

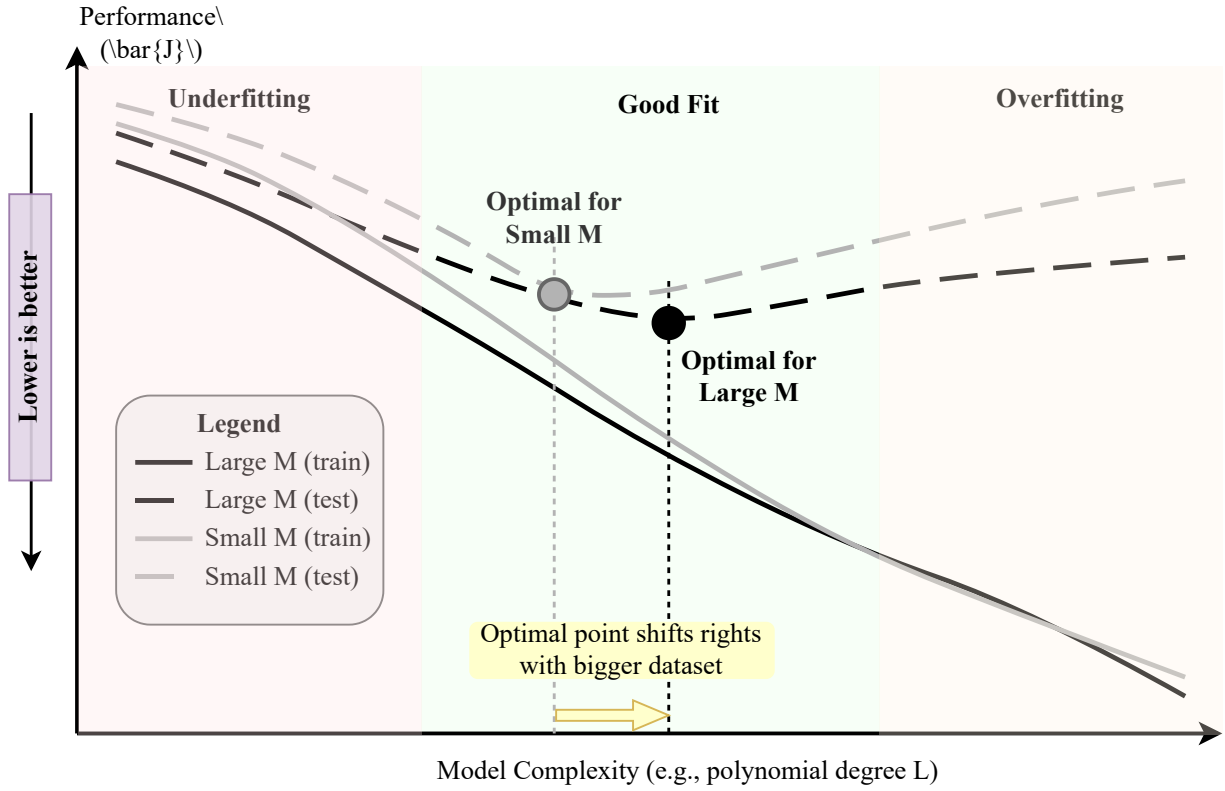


Figure 5.5.: Increase in dataset size improves the over-complex model performance.

5.5. Regularization

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

Regularization: Penalty to the loss function

$$\mathcal{L}_{\text{new}} = \mathcal{L} + \lambda g(\mathbf{w}) \tag{5.7}$$

where λ is termed *regularization parameter*.

L_2-regularization: Special case of

$$g(\mathbf{w}) = \frac{1}{2M} \|\mathbf{w}\|^2 = \frac{\lambda}{2M} \sum_{i=1}^{N} w_i^2 \tag{5.8}$$

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. 5.2). Reducing them results in smooth $\hat{y}$ prediction. The illustration of λ influence is presented in Fig. 5.6. The corresponding underfitting-overfitting λ-related trade-off is presented in Fig. 5.7.

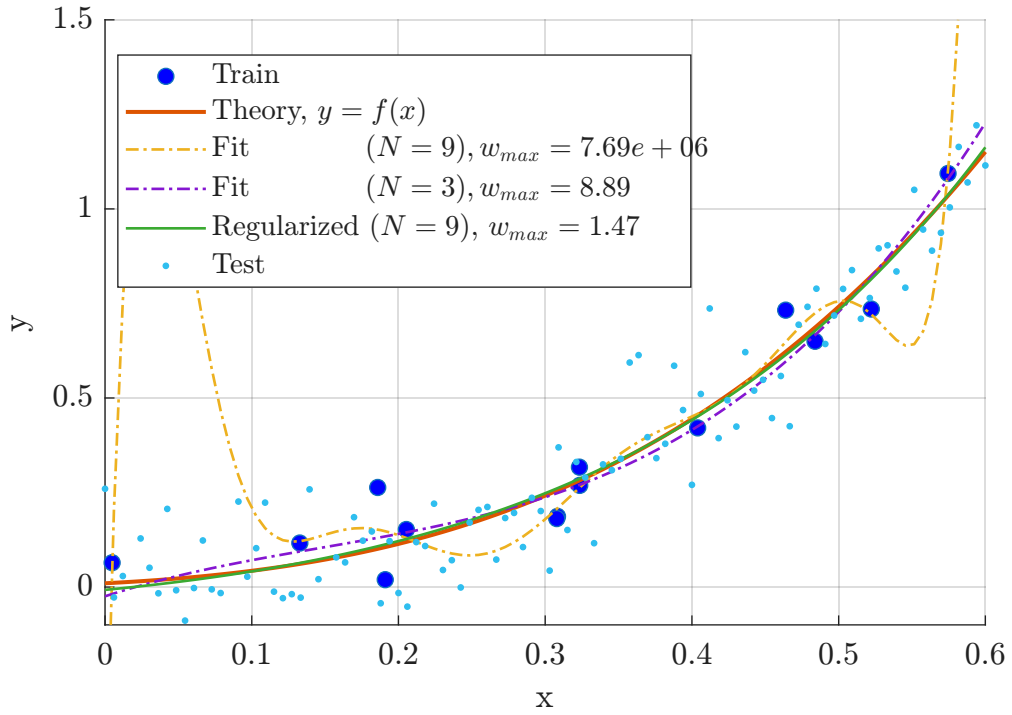


Figure 5.6.: Illustration of regularization influence on the polynomial model.

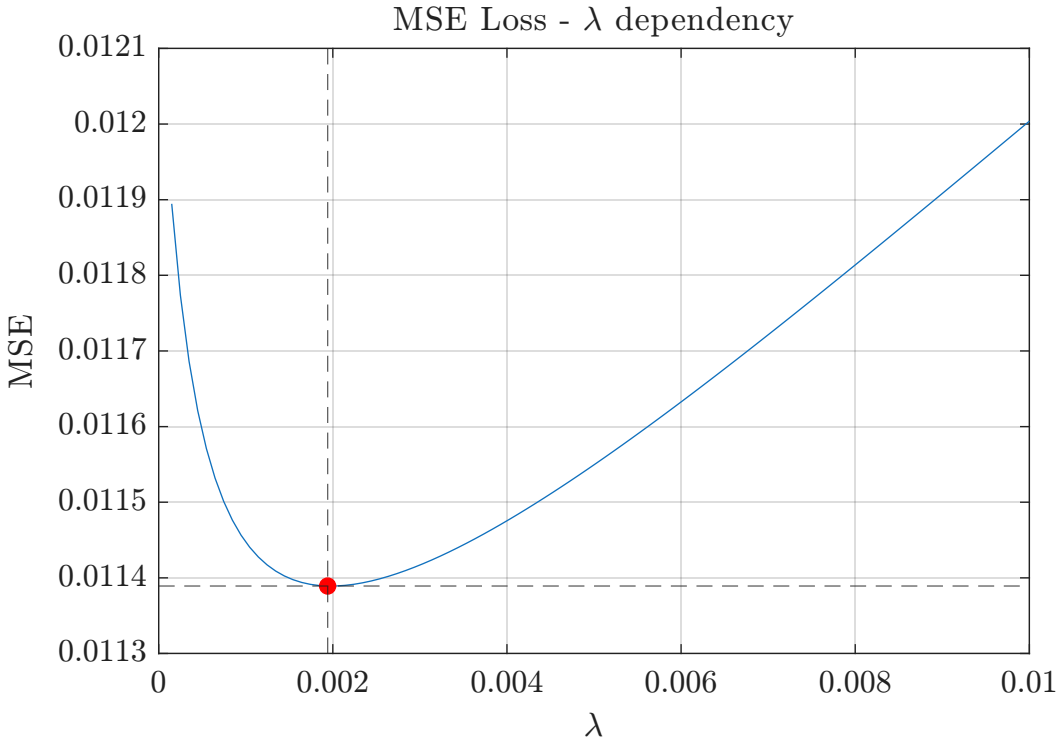


Figure 5.7.: Dependence of MSE on λ for regularization in Fig. 5.6 (polynomial regression). Overfitting is on the left ($\lambda \to 0$) and undefitting on the right ($\lambda \to \infty$).

5.5.1. Ridge Regression

Ridge regression: L_2-regularized linear regression.

$$\begin{aligned}\mathcal{L}(\mathbf{w}) &= \frac{1}{2M}\left\|\mathbf{y}-\mathbf{X}\mathbf{w}\right\|^2 + \frac{\lambda}{2M}\underbrace{\left\|\mathbf{w}\right\|^2}_{\sum_{i=1}^N w_i^2} \\ &= \frac{1}{2M}\left(\mathbf{y}-\mathbf{X}\mathbf{w}\right)^T\left(\mathbf{y}-\mathbf{X}\mathbf{w}\right) + \frac{\lambda}{2M}\mathbf{w}^T\mathbf{w}\end{aligned} \tag{5.9}$$

Derivative

$$\nabla\mathcal{L}(\mathbf{w}) = \frac{1}{M}\left(-\mathbf{X}^T\left(\mathbf{y}-\mathbf{X}\mathbf{w}\right)+\lambda\mathbf{w}\right) = 0 \tag{5.10}$$

Solution

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

Finally, the regularized weights are given by

$$\mathbf{w} = \left(\lambda\mathbf{I}+\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y} \tag{5.11}$$

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

$$\begin{aligned}\tilde{y} &= \tilde{\mathbf{X}}\mathbf{w} \\ \begin{bmatrix}\mathbf{y}\\0\\\vdots\\0\end{bmatrix} &= \begin{bmatrix}- & \mathbf{X} & - \\ \sqrt{\lambda} & & 0 \\ & \ddots & \\ 0 & & \sqrt{\lambda}\end{bmatrix}\begin{bmatrix}w_1\\\vdots\\w_N\end{bmatrix} + w_0\end{aligned} \tag{5.12}$$

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 is illustrated in Fig. 5.8.

GD Solution

$$\begin{aligned}\mathbf{w}_{n+1} &= \mathbf{w}_n - \frac{\alpha}{M}\left[\mathbf{X}^T\left(\mathbf{X}\mathbf{w}_n-\mathbf{y}\right)+\lambda\mathbf{w}_n\right] \\ &= \mathbf{w}_n\left(1-\frac{\alpha}{M}\lambda\right) - \frac{\alpha}{M}\mathbf{X}^T\left(\mathbf{X}\mathbf{w}_n-\mathbf{y}\right)\end{aligned} \tag{5.13}$$

The α value is typically chosen such that

$$0.99 \gtrsim \left(1-\frac{\alpha}{M}\lambda\right) \gtrsim 0.95 \tag{5.14}$$

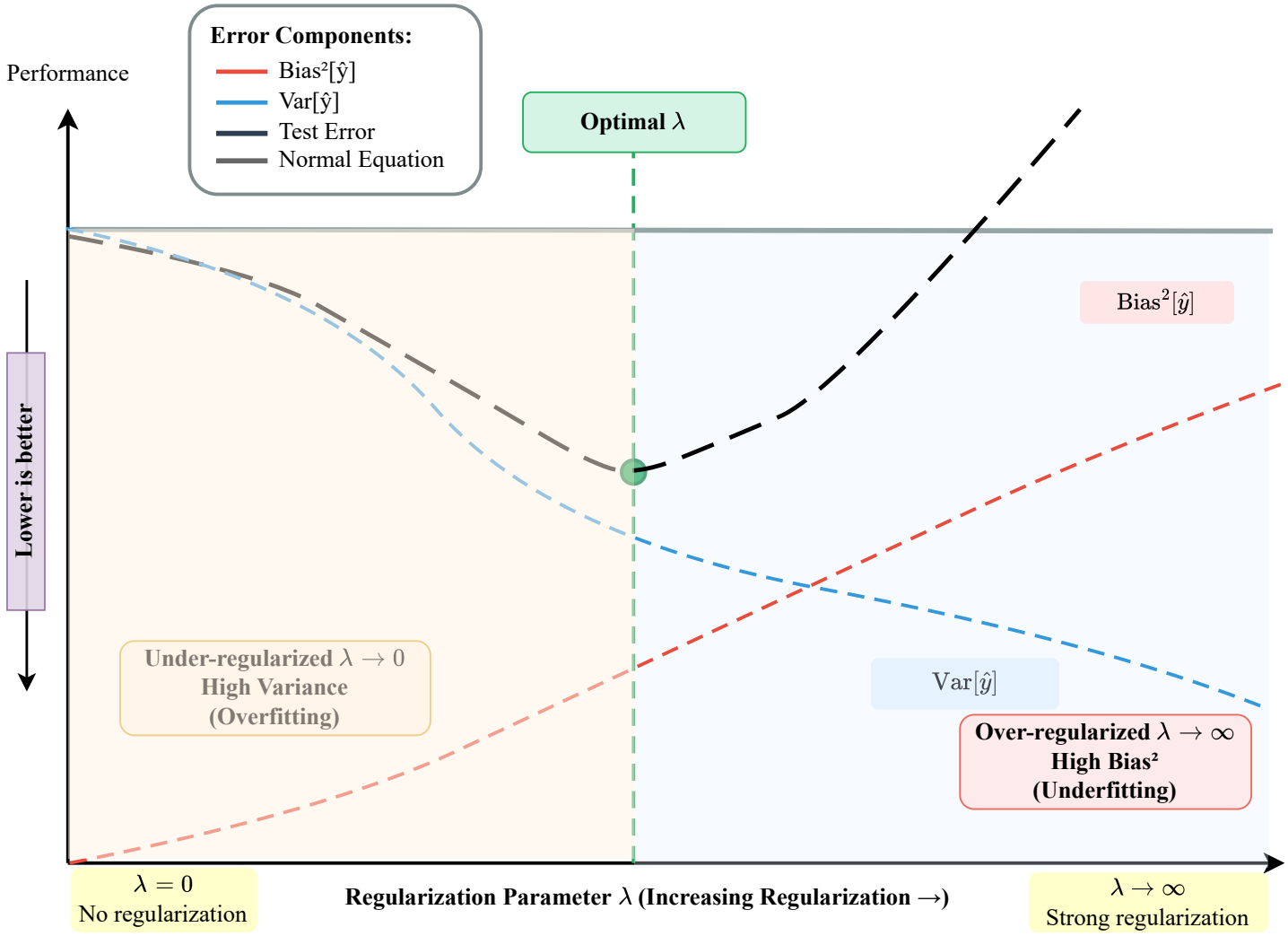


Figure 5.8.: Regularization trade-off for ridge regression: Test error $\bar{J}_{\text{test}}$ is minimized at optimal λ. Note, the trends are opposite to Fig. 5.4.

6. Kernels

The goal is to effectively apply non-linear models.

Contents

6.1. Uni-variate Polynomial Model

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

$$\begin{aligned}\hat{y} = f(x; \mathbf{w}) &= w_0 + w_1 x + w_2 x^2 + \cdots + w_L x^L \\ &= \sum_{j=0}^{L} w_j x^j \end{aligned} \tag{6.1}$$

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

$$\hat{y} = \sum_{j=0}^{L} w_j z_j \tag{6.2}$$

[1] The model is also presented in Sec. 4.1. The selection of L is discussed in Ch. 5.

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),

$$\mathbf{X} = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^L \\ 1 & x_2 & x_2^2 & \cdots & x_2^L \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_M & x_M^2 & \cdots & x_M^L \end{bmatrix} \tag{6.3}$$

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

6.2. Mapping function

The data-linearization results (e.g., in (6.3)) can be formulated by a *mapping function*.

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

$$\phi(x) = \begin{bmatrix} 1 & x & x^2 & \cdots & x^L \end{bmatrix} \tag{6.4}$$

Applying ϕ to each value in $\{x_k\}_{k=1}^M$ builds the matrix $\mathbf{X}$ in (6.3).

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

$$\phi(\mathbf{x}) = \phi(x_1, x_2) = \begin{bmatrix} 1, x_1, x_2, x_1 x_2, x_1^2, x_2^2 \end{bmatrix} \tag{6.5}$$

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

$$\phi(\mathbf{X}) = \mathbf{X}_\phi = \begin{bmatrix} 1 & x_{1_1} & x_{2_1} & x_{1_1} x_{2_1} & x_{1_1}^2 & x_{2_1}^2 \\ 1 & x_{1_2} & x_{2_2} & x_{1_2} x_{2_2} & x_{1_2}^2 & x_{2_2}^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 1 & x_{1_M} & x_{2_M} & x_{1_M} x_{2_M} & x_{1_M}^2 & x_{2_M}^2 \end{bmatrix} \tag{6.6}$$

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 $\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 \end{bmatrix} \in \mathbb{R}^{M\times 3}$ (using (3.7) vector notation) the resulting $\mathbf{X}^T\mathbf{X} \in \mathbb{R}^{3\times 3}$,

$$\mathbf{X}^T\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \mathbf{x}_3^T \end{bmatrix} \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 \end{bmatrix} = \begin{bmatrix} \mathbf{x}_1^T\mathbf{x}_1 & \mathbf{x}_1^T\mathbf{x}_2 & \mathbf{x}_1^T\mathbf{x}_3 \\ \mathbf{x}_2^T\mathbf{x}_1 & \mathbf{x}_2^T\mathbf{x}_2 & \mathbf{x}_2^T\mathbf{x}_3 \\ \mathbf{x}_3^T\mathbf{x}_1 & \mathbf{x}_3^T\mathbf{x}_2 & \mathbf{x}_3^T\mathbf{x}_3 \end{bmatrix}, \tag{6.7}$$

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

6.3. 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 $\phi(\mathbf{x})$.

6.3.1. LS Identity

The base of the method is an identity

$$\begin{aligned} \mathbf{w} &= \mathbf{X}^T \left(\mathbf{X}\mathbf{X}^T\right)^{-1} \mathbf{y} \\ &= \left(\mathbf{X}^T\mathbf{X}\right)^{-1} \mathbf{X}^T\mathbf{y} \end{aligned} \tag{6.8}$$

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 $\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 \end{bmatrix} \in \mathbb{R}^{M\times 3}$ (using (3.7) vector notation) the resulting $\mathbf{X}\mathbf{X}^T \in \mathbb{R}^{M\times M}$,

$$\mathbf{X}\mathbf{X}^T = \begin{bmatrix} \mathbf{x}_1 & \mathbf{x}_2 & \mathbf{x}_3 \end{bmatrix} \begin{bmatrix} \mathbf{x}_1^T \\ \mathbf{x}_2^T \\ \mathbf{x}_3^T \end{bmatrix} = \mathbf{x}_1\mathbf{x}_1^T + \mathbf{x}_2\mathbf{x}_2^T + \mathbf{x}_3\mathbf{x}_3^T \tag{6.9}$$

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

6.3.2. 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 (6.6))

$$\mathbf{X}_\phi = \phi(\mathbf{X}) \tag{6.10}$$

By using the identity

$$\phi\left(\mathbf{X}^T\right) = \phi\left(\mathbf{X}\right)^T \tag{6.11}$$

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

$$\mathbf{K}(\mathbf{X}, \mathbf{X}) = \phi\left(\mathbf{X}\right)\phi\left(\mathbf{X}\right)^T \tag{6.12}$$

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.

6.3.3. Handling Overfitting

Large L values, e.g. in the polynomial model, result in a high potential for overfitting. Therefore, ridge regression (5.11) is applied together with a kernel trick,

$$\mathbf{w} = \mathbf{X}^T\left(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I}\right)^{-1}\mathbf{y} \tag{6.13}$$

6.3.4. Kernel-Based Model

The calculation of the kernel trick relation in Eq. (6.13) is performed by the following steps:

1. Evaluate the kernel matrix $\mathbf{K}$ using (6.12).
2. Calculate $\boldsymbol{\alpha}$,
$$\boldsymbol{\alpha} = (\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{y} \tag{6.14}$$
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
$$\mathbf{w} = \varphi\left(\mathbf{X}\right)^T\boldsymbol{\alpha} = \sum_{j=1}^{M}\alpha_j\varphi\left(\tilde{\mathbf{x}}_j\right) \tag{6.15}$$
but is not required for prediction.
3. **Model inference:** The regression for a new test point $\tilde{\mathbf{x}}_0$ is evaluated by
$$\begin{aligned}\hat{y}_0 &= \varphi(\tilde{\mathbf{x}}_0)^T\mathbf{w}\\ &= \sum_{j=1}^{M}\alpha_j\varphi(\tilde{\mathbf{x}}_0)^T\varphi\left(\tilde{\mathbf{x}}_j\right)\\ &= \varphi(\tilde{\mathbf{x}}_0)^T\varphi\left(\mathbf{X}\right)\boldsymbol{\alpha}\\ &= K\left(\tilde{\mathbf{x}}_0, \mathbf{X}\right)\boldsymbol{\alpha}\end{aligned} \tag{6.16}$$

where

$$\begin{aligned} \varphi(\tilde{\mathbf{x}}_0)^T \in \mathbb{R}^{1\times L}, \varphi(\tilde{\mathbf{x}}_0) \in \mathbb{R}^{L\times 1} \\ \varphi\left(\mathbf{X}\right)^T \in \mathbb{R}^{L\times M}, \varphi\left(\mathbf{X}\right) \in \mathbb{R}^{M\times L} \\ K\left(\tilde{\mathbf{x}}_0, \mathbf{X}\right) \in \mathbb{R}^{1\times M} \end{aligned} \tag{6.17}$$

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

In summary, the order is:

1. Kernel matrix, (6.12).
2. $\boldsymbol{\alpha}$, (6.14).
3. For test data $\tilde{\mathbf{X}}_0$, the prediction (6.16) becomes

$$\hat{\mathbf{y}}_0 = K\left(\tilde{\mathbf{X}}_0, \mathbf{X}\right)\boldsymbol{\alpha} \tag{6.18}$$

6.4. Kernel Types

6.4.1. Polynomial Kernel

$$K(\mathbf{x}, \mathbf{y}) = \left(\mathbf{x}^\mathsf{T}\mathbf{y} + c\right)^d \tag{6.19}$$

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:

$$\begin{aligned} K(\mathbf{a}, \mathbf{b}) &= \left(\mathbf{a}^T\mathbf{b} + 1\right)^2 \\ &= \left(\mathbf{a}^T\mathbf{b}\right)^2 + 2\mathbf{a}^T\mathbf{b} + 1 \end{aligned} \tag{6.20}$$

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

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.

6.4.2. Gaussian Radial Basis Kernel (RBK)

The kernel definition is

$$
K(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|^2}{2b}\right) \tag{6.21}
$$

Due to Taylor expansion,

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

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. 5.6) is presented in Fig. 6.1.

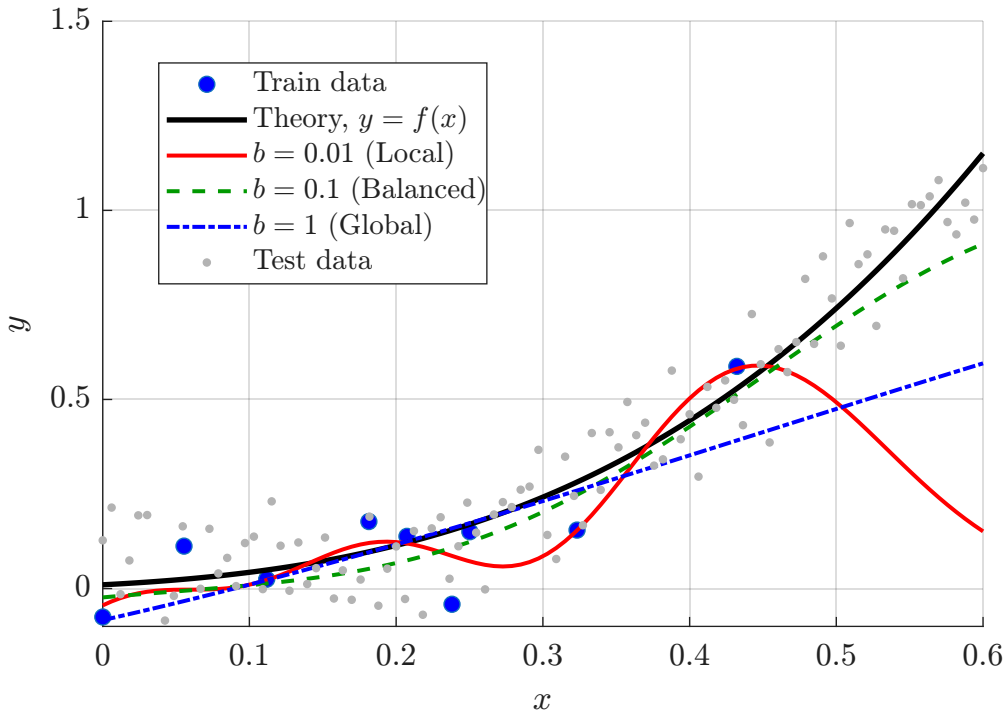


Figure 6.1.: Kernel regression example with polynomial kernel. Direct comparison of RBF kernel regression with three bandwidth values. The medium bandwidth ($b = 0.1$, green) achieves the best balance between fitting the data and maintaining smoothness.

6.4.3. 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 $\mathbf{x}_j$ and inter-data $\mathbf{x}_j, \mathbf{x}_k$ relations are unknown.

6.5. Kernel Smoothing

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

$$\hat{y}_0 = \sum_{j=1}^{M} w_j y_j \tag{6.22}$$

where the weights are

$$\tilde{\mathbf{w}} = K(\tilde{\mathbf{x}}_0, \mathbf{X}) \tag{6.23}$$

with normalization

$$w_j = \frac{\tilde{w}_j}{\sum_{j=1}^{M} \tilde{w}_j} \tag{6.24}$$

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

$$\tilde{\mathbf{W}} = K(\mathbf{X}_{\text{test}}, \mathbf{X}_{\text{train}}) \qquad \in \mathbb{R}^{M_{\text{test}} \times M_{\text{train}}} \tag{6.25}$$

$$\mathbf{W} = \tilde{\mathbf{W}} / \tilde{\mathbf{W}} \mathbf{1}_{M_{\text{train}}} \tag{6.26}$$

$$\hat{\mathbf{y}}_{\text{test}} = \mathbf{W} \mathbf{y}_{\text{train}} \tag{6.27}$$

The numerical example is presented in Fig. 6.2.

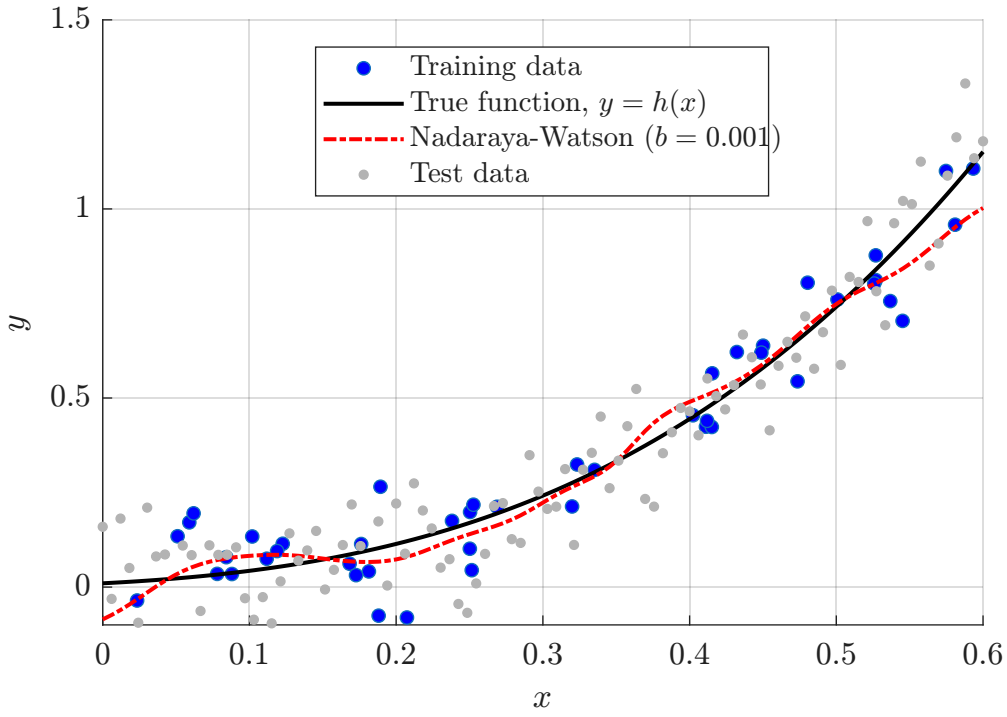


Figure 6.2.: Kernel smoothing example with exponential kernel. Incorrect selection of b parameters results in overfitting. The kernel smoother provides a non-parametric estimate by computing locally weighted averages of nearby training points, where weights decay exponentially with distance.

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.

7. Regression Losses and Metrics

Goal: Quantify the difference between predicted and target values.

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

Contents

7.1. Preface

The predicted range of $\hat{y}_i$ results from the particular applied model.

Metric A performance metric is a function $J : \hat{\mathbf{y}}, \mathbf{y} \to \mathscr{R}$ 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 $L : \hat{\mathbf{y}}, \mathbf{y} \to \mathscr{R}$ that is calculated over the test set, such that optimal parameters that correspond to the minimum loss

$$\boldsymbol{\theta} = \arg\min_{\boldsymbol{\theta}} \mathcal{L}(\hat{\mathbf{y}}, \mathbf{y}) \tag{7.1}$$

can be evaluated (Sec. 4.4.3).

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.

7.2. 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 $\boldsymbol{\theta}$. 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,

$$\left|f(x_1) - f(x_2)\right| \leq K|x_1 - x_2|. \tag{7.2}$$

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 Lipschitz continuous.

This property is particularly important in NNs, which are based on the back-propagation principle.

Convexity and strict convexity: Each segment lies above the graph between two points (Fig. 7.1). Formally, it means that the line 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 $\forall x_1, x_2 \in \mathbb{R}$,

$$f\left(tx_1 + (1-t)x_2\right) \leq tf(x_1) + (1-t)f(x_2) \tag{7.3}$$

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, the meaning of convexity is the presence of one global minimum.

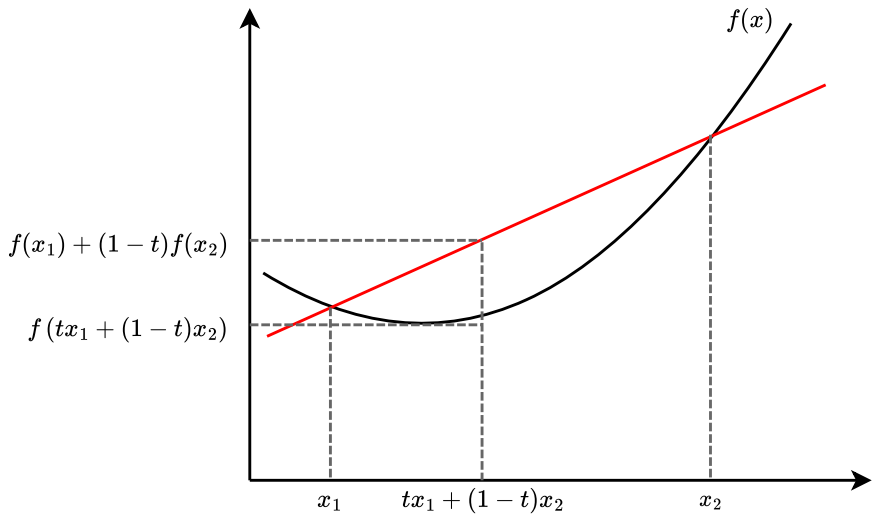


Figure 7.1.: Convexity property visualization.

7.3. Losses

Per-sample error notation is, $e_i = y_i - \hat{y}_i, i = 1, \ldots, M$. The corresponding vector notation is $\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}}$. Note, the order of subtraction is not important in the context of the following material.

Some of the following losses are also used as metrics.

7.3.1. Mean-squared error (MSE)

- Continuous, differentiable, convex

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

$$\begin{aligned} J(\mathbf{y}, \hat{\mathbf{y}}) &= \frac{1}{M}\|\mathbf{y} - \hat{\mathbf{y}}\|^2 = \frac{1}{M}\mathbf{e}^T\mathbf{e} \\ &= \frac{1}{M}\sum_{i=1}^{M}(y_i - \hat{y}_i)^2 = \frac{1}{M}\sum_{i=1}^{M} e_i^2 \end{aligned} \tag{7.4}$$

When used as loss, sometimes factor 2 is applied to "compensate" for the derivative,

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{2M}\sum_{i=1}^{M} e_i^2 \tag{7.5}$$

Sum-of-squared error (SSE) ((2.4)) is also used.

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.

7.3.2. RMSE

- Continuous, differentiable, convex

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

$$\begin{aligned} J(\mathbf{y}, \hat{\mathbf{y}}) &= \frac{1}{\sqrt{M}} \|\mathbf{y} - \hat{\mathbf{y}}\| \\ &= \sqrt{\frac{1}{M} \sum_{i=1}^{M} (y_i - \hat{y}_i)^2} = \sqrt{\frac{1}{M} \sum_{i=1}^{M} e_i^2} \end{aligned} \tag{7.6}$$

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

7.3.3. Mean absolute error (MAE)

MAE is used both as loss and metric.

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{M} \sum_{i=1}^{M} |y_i - \hat{y}_i| = \frac{1}{M} \sum_{i=1}^{M} |e_i| \tag{7.7}$$

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 7.1 and in Fig. 7.2.

Table 7.1.: Example of MSE, RMSE and MAE. Small change in error results in significant change in MSE and less dramatic change in MAE.

True Values	Predicted Values	MSE	RMSE	MAE
(**40**,30)	(30,25)	$\frac{(40-30)^2}{2} + \frac{(30-25)^2}{2} = 62.5$	$\sqrt{62.5} = 7.91$	$\frac{\lvert 40-30 \rvert}{2} + \frac{\lvert 30-25 \rvert}{2} = 7.5$
(**50**,30)	(30,25)	$\frac{(50-30)^2}{2} + \frac{(30-25)^2}{2} = 212.5$	$\sqrt{212.5} = 14.6$	$\frac{\lvert 50-30 \rvert}{2} + \frac{\lvert 30-25 \rvert}{2} = 12.5$

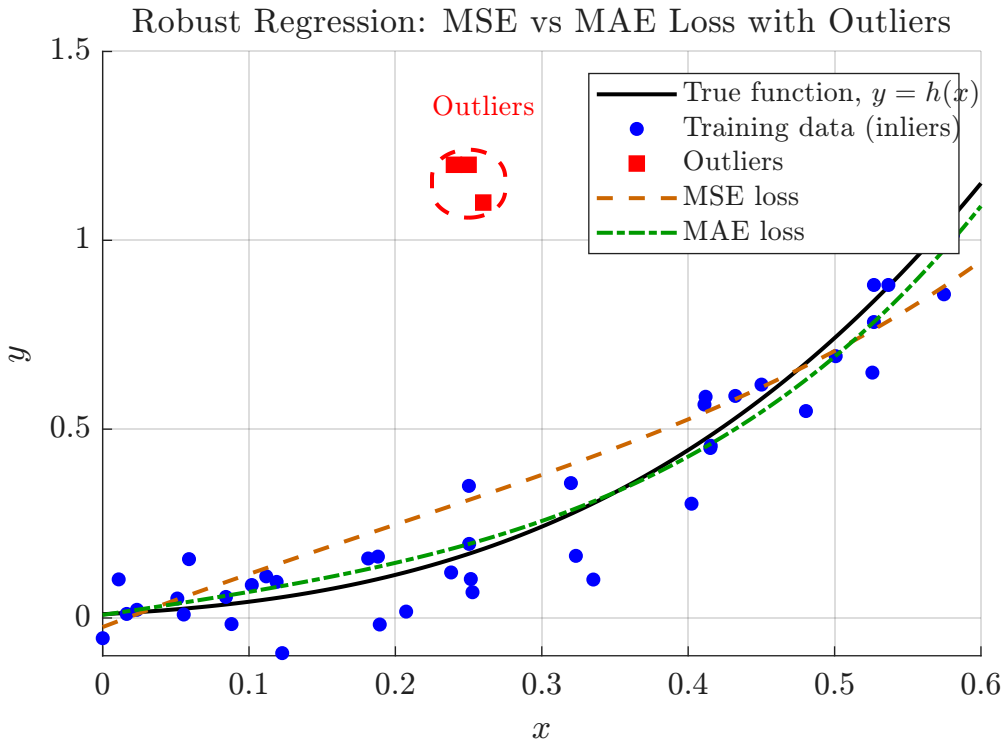


Figure 7.2.: The model is $N = 4$ polynomial with 40 inliers and three outliers at significantly higher values. Both MSE and MAE were used with λ-optimized L_2-regularization. MSE regression is pulled toward outliers due to squaring large errors, while MAE regression provides more robust fitting.

7.3.4. Huber loss

- L-Continuous, differentiable, s-convex

Goal: Hybrid between MAE and MSE.

$$\mathcal{L}(e_i) = \begin{cases} \frac{1}{2}e_i^2 & |e_i| \leq \delta \\ \delta\left(|e_i| - \frac{1}{2}\delta\right) & \text{otherwise} \end{cases} \tag{7.8}$$

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.

7.3.5. Log-cosh loss

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

$$\mathcal{L}(e_i) = \log\cosh e_i \tag{7.9}$$

Properties:

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

$$L(e_i) \approx \begin{cases} \frac{e_i^2}{2} & e_i \text{ small} \\ |e_i| - \log(2) & e_i \text{ large} \end{cases} \tag{7.10}$$

- 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.

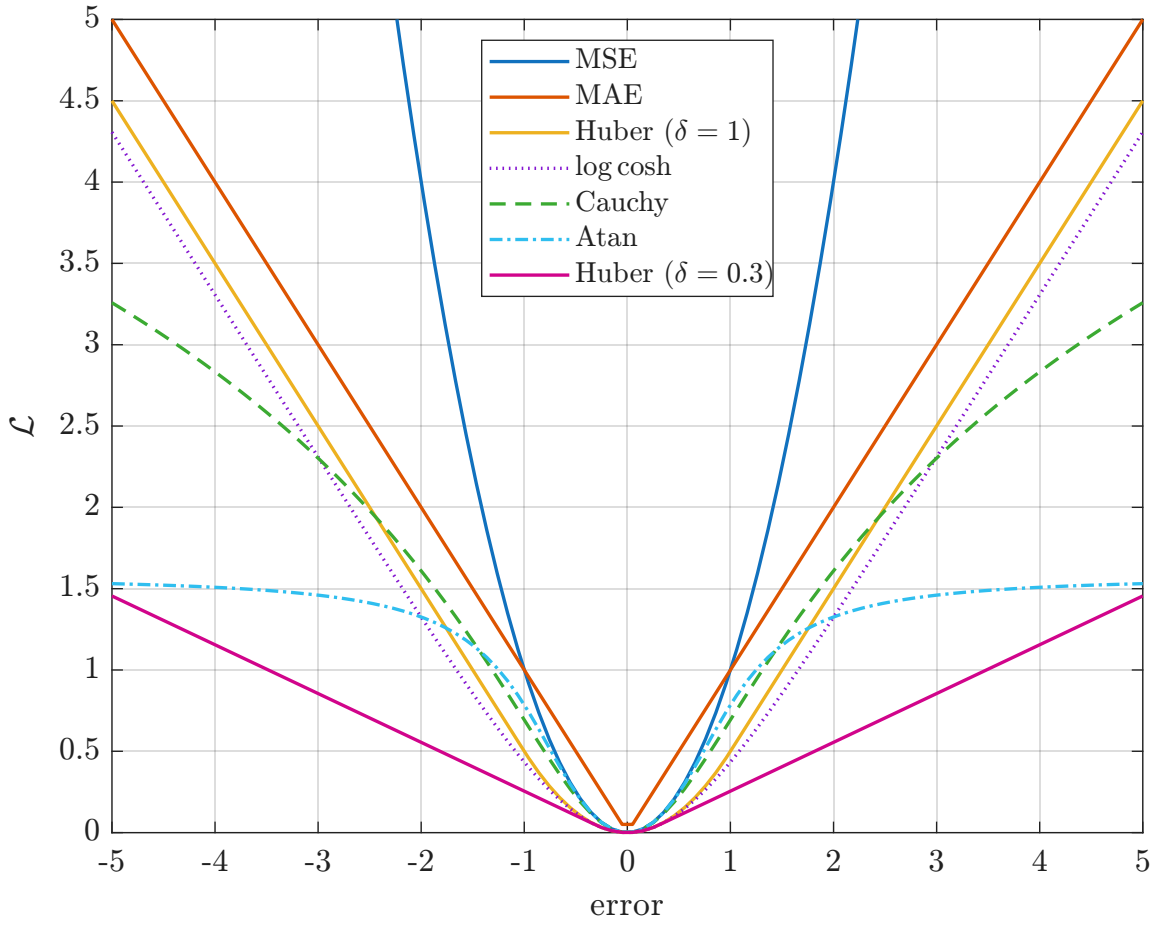


Figure 7.3.: Visualization of different loss functions. MSE is the steepest.

7.3.6. Cauchy

Goal: More robust to outliers than MAE.

$$\mathcal{L}(e_i) = \log\left(1 + \left(\frac{e_i}{d}\right)^2\right) \tag{7.11}$$

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

7.3.7. Atan

Goal: More robust to outliers than Cauchy.

$$\mathcal{L}(e_i) = \arctan\left(e_i^2\right) \tag{7.12}$$

- 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).

7.3.8. 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.

$$\begin{aligned}\mathcal{L}(y_i, \hat{y}_i) &= \frac{1}{M}\sum_{i=1}^{M}\left(\log(y_i+1) - \log(\hat{y}_i+1)\right)^2 \\ &= \frac{1}{M}\sum_{i=1}^{M}\log\left(\frac{y_i+1}{\hat{y}_i+1}\right)^2\end{aligned} \tag{7.13}$$

'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 7.2.
- Penalizes underestimated values more than overestimated values.
- Root MSLE (RMSLE) is also used, e.g. in `scikit`.

Table 7.2.: Example of MSLE

True Values	Predicted Values	MSE Loss	MSLE Loss
40	30	100	0.0782
4000	3000	**1,000,000**	0.0827
20	10	100	**0.4181**
20	30	100	**0.1517**

7.4. 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.

$$J(y_i, \hat{y}_i) = \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \overline{y})^2} = \frac{MSE}{\mathrm{Var}[\mathbf{y}]} = \frac{\|\mathbf{y} - \hat{\mathbf{y}}\|^2}{\|\mathbf{y} - \bar{\mathbf{y}}\|^2} \tag{7.14}$$

Shows the fraction of the explained variance, since $\hat{y}_i = \bar{y}$ is the minimum MSE predictor if the data and $\mathbf{y}$ are statistically independent. Closer to 0 is better.

R^2

The common metric in social sciences,

$$R^2 = 1 - RSE \tag{7.15}$$

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%.

Normalized Root Mean Squared Error Expressed as a percentage, defined as:

$$J(y_i, \hat{y}_i) = 100 \left(1 - \frac{\|\mathbf{y} - \hat{\mathbf{y}}\|}{\|\mathbf{y} - \bar{\mathbf{y}}\|}\right) = 100 \left(1 - \sqrt{RSE}\right) \tag{7.16}$$

Used in Matlab.

Relative absolute error (RAE)

Normalized MAE loss.

$$J(y_i, \hat{y}_i) = \frac{\sum_i |y_i - \hat{y}_i|}{\sum_i |y_i - \overline{y}|} = \frac{MAE}{\sum_i |y_i - \overline{y}|} \tag{7.17}$$

Closer to 0 is better.

Mean Absolute Percentage Error (MAPE)

Scaled error metric.

$$J = \frac{1}{M} \sum_{i=1}^{M} \frac{|y_i - \hat{y}_i|}{y_i} \times 100\% \tag{7.18}$$

- Beware of small denominator!
- Can exceed 100%.
- Asymmetric, as described in Table 7.3.

Table 7.3.: Example of MAPE

True Values	Predicted Values	Absolute Error	MAPE
100	60	40	40%
20	60	40	**300%**

Additional Reading Additional reading on regression metrics: [3]

7.5. Number of Parameters Penalty

Adding parameters improves performance but may result in overfitting. These metrics attempt to resolve this problem by introducing a penalty term for the *number of parameters* in the model with MSE loss.

All these have lengthy theoretical justification that is not provided here.

- N is the number of parameters,
- M is the number of data-points.

Main assumption: residuals have Gaussian distribution.

Akaike's Final Prediction Error (FPE)

Akaike's Final Prediction Error (FPE) criterion provides a measure of model quality.

$$FPE = MSE \frac{1 + N/M}{1 - N/M} = MSE \frac{M + N}{M - N} \tag{7.19}$$

Akaike's Information Criterion Penalizes the number of parameters,

$$\Delta AIC = 2N + M \ln(MSE) \tag{7.20}$$

AICc is AIC with a correction for small sample sizes

$$AICc = AIC + 2N \frac{N + 1}{M - N - 1} \tag{7.21}$$

Bayesian Information Criterion (BIC)

$$BIC = M \ln(MSE) + N \ln(M) \tag{7.22}$$

7.6. Summary

7.6.1. Loss Selection Guidelines

- **MSE**: Default choice for clean data; analytical gradients enable fast convergence.
- **MAE**: Prefer when outliers are present; requires adaptive learning rate.
- **Huber**: Best of both worlds when δ can be tuned via cross-validation.
- **MSLE**: Use for high dynamic range targets where relative error matters.
- **Cauchy/Atan**: Heavy outlier contamination; require careful initialization.

7.6.2. Metric Selection Guidelines

- **RMSE**: Same units as target; intuitive for stakeholders.
- R^2: Explains variance reduction vs. mean baseline; use for model comparison.
- **MAPE**: Percentage interpretation; avoid when y approaches zero.

7.6.3. 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.
- R^2 **misuse**: High R^2 does not guarantee good predictions; inspect residuals and error magnitudes.
- **Non-convex robust losses**: Without careful initialization/scaling, they may converge to poor local minima.
- **MAE with fixed learning rate**: May oscillate near the optimum; use learning-rate decay or adaptive optimizers.

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

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 $\{\mathbf{x} : \mathbf{w}^T\mathbf{x} \lesseqgtr thr\}$ 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.

8.2. Basic Linear Model

Goal: Use linear regression for classification as a baseline approach.

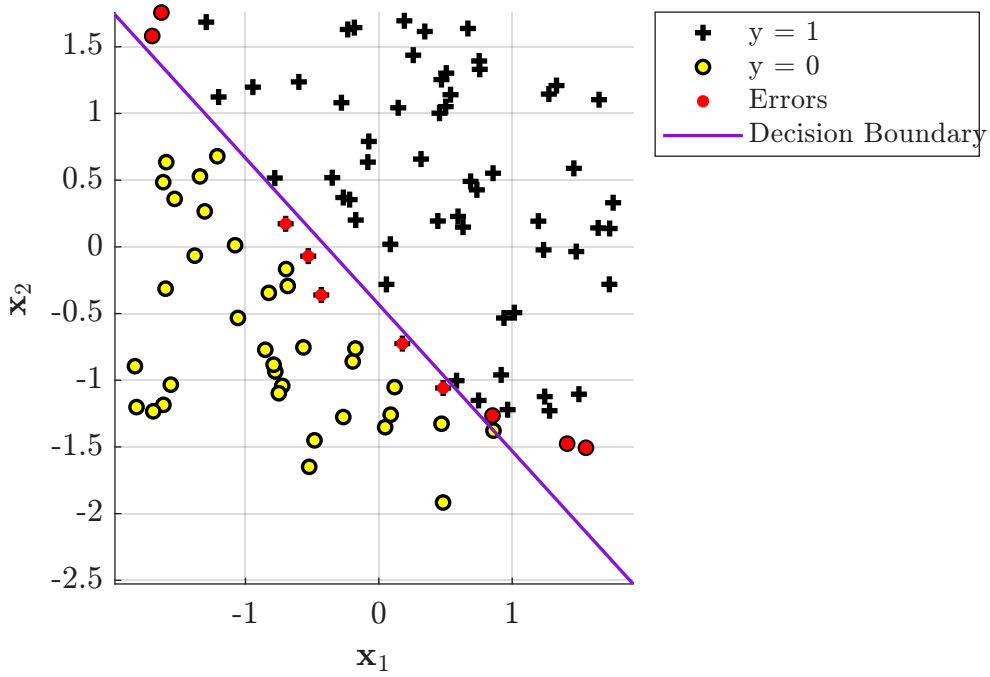


Figure 8.1.: An example of linear classification boundary.

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

- **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.

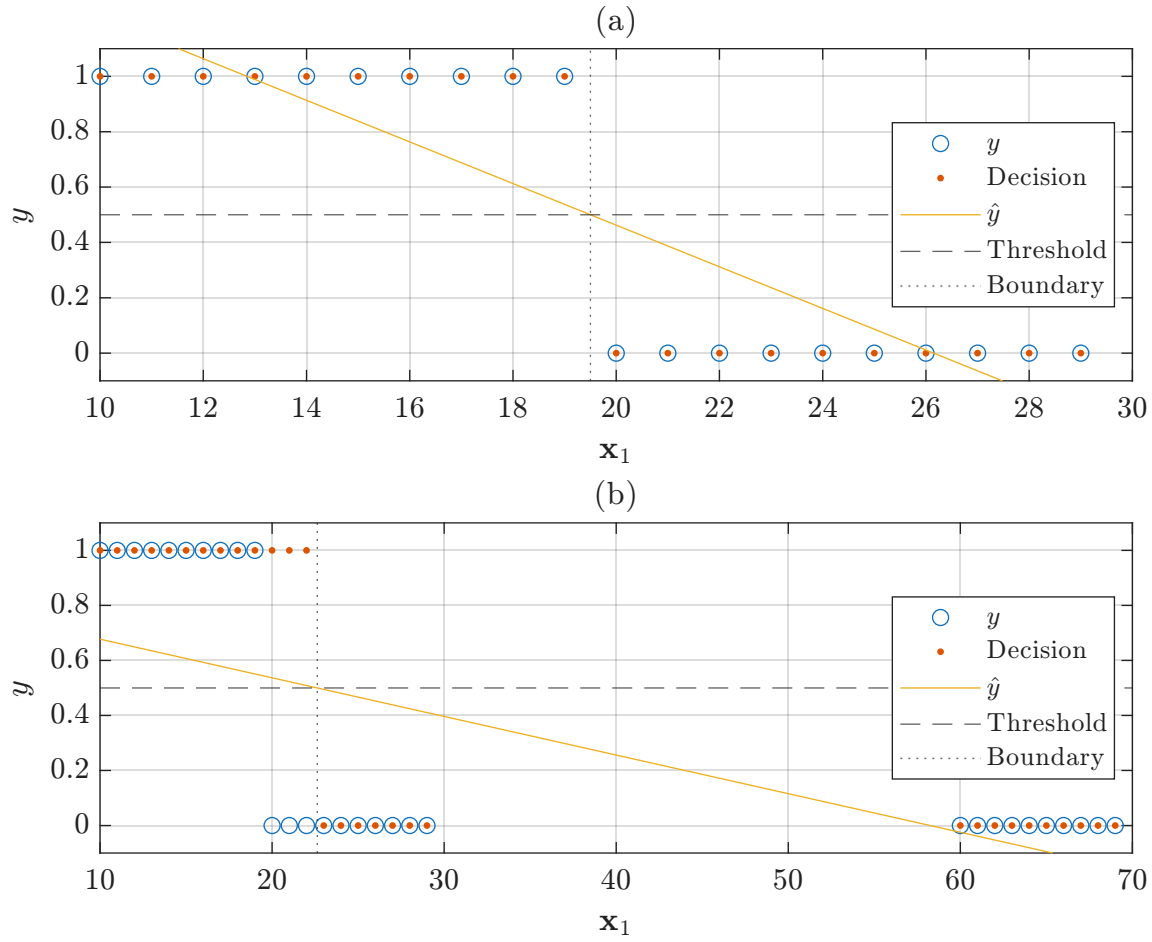


Figure 8.2.: Linear regression for 1D binary classification: (a) least-squares fit with threshold at 0.5 correctly separates the two classes; (b) adding class-0 outliers shifts the decision boundary and causing misclassification.

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.

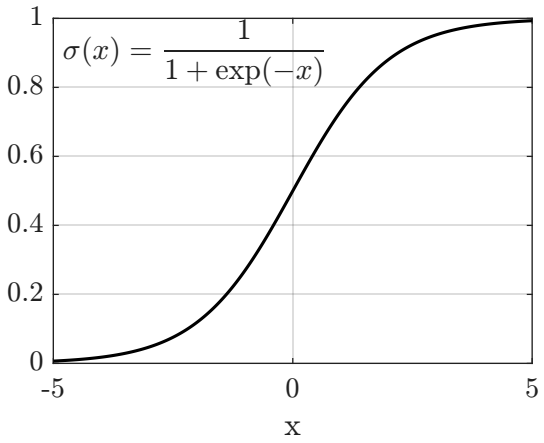


Figure 8.3.: Sigmoid function: bounded output $\sigma(x) \in [0, 1]$ with saturating tails.

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.

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 $\Pr(y = 1 \mid \mathbf{x}, \mathbf{w})$. The loss function should therefore measure how far the predicted distribution is from the target distribution. Cross-entropy, rooted in information theory, provides exactly such a measure.

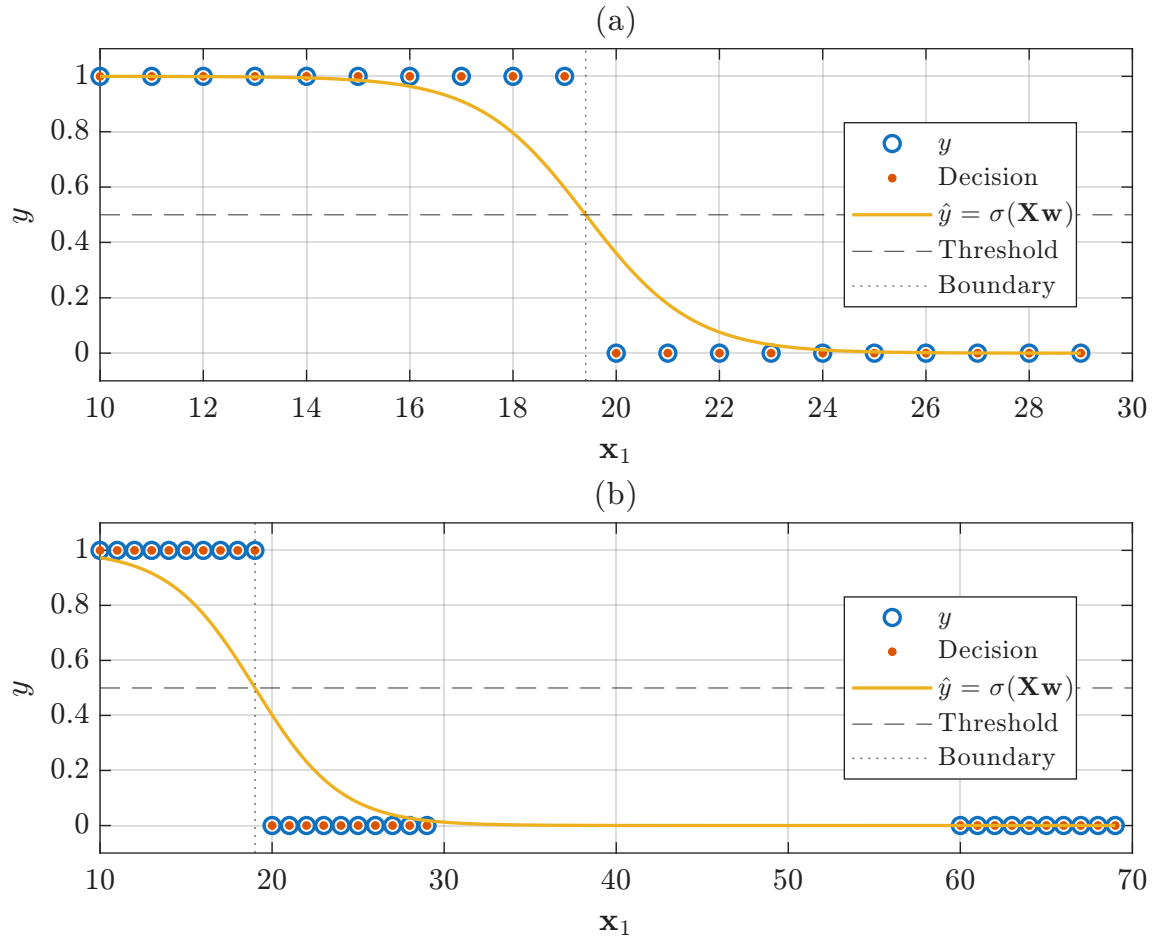


Figure 8.4.: Logistic regression for 1D binary classification: (a) sigmoid fit correctly separates the two classes; (b) adding class-0 outliers does not shift the decision boundary, unlike linear regression (Fig. 8.2).

8.4.1. 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 8.1, 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 8.1.: 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.

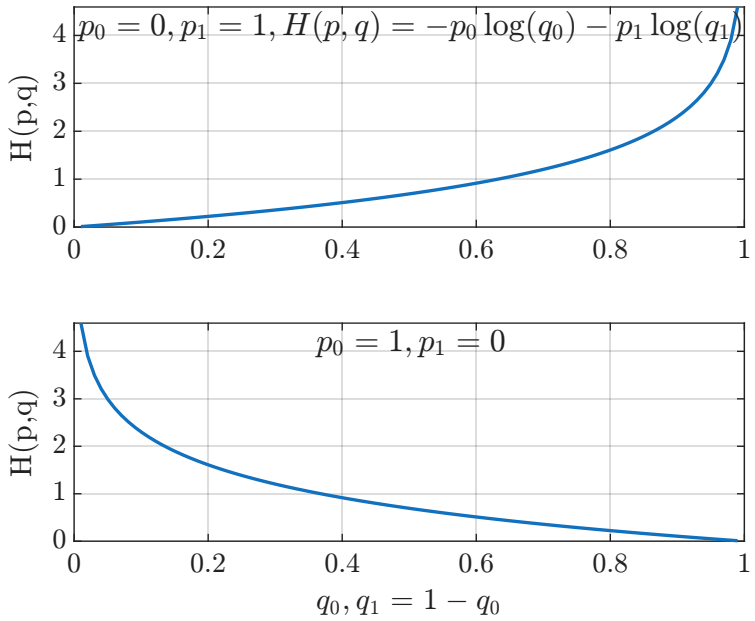


Figure 8.5.: Binary cross-entropy for the two cases: $y = 1$ (left) and $y = 0$ (right) (Eq. (8.10)).

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):

$$H(p, q) = -(1 - y) \log\big(1 - f_{\boldsymbol{\theta}}(\mathbf{x})\big) - y \log\big(f_{\boldsymbol{\theta}}(\mathbf{x})\big)$$

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.

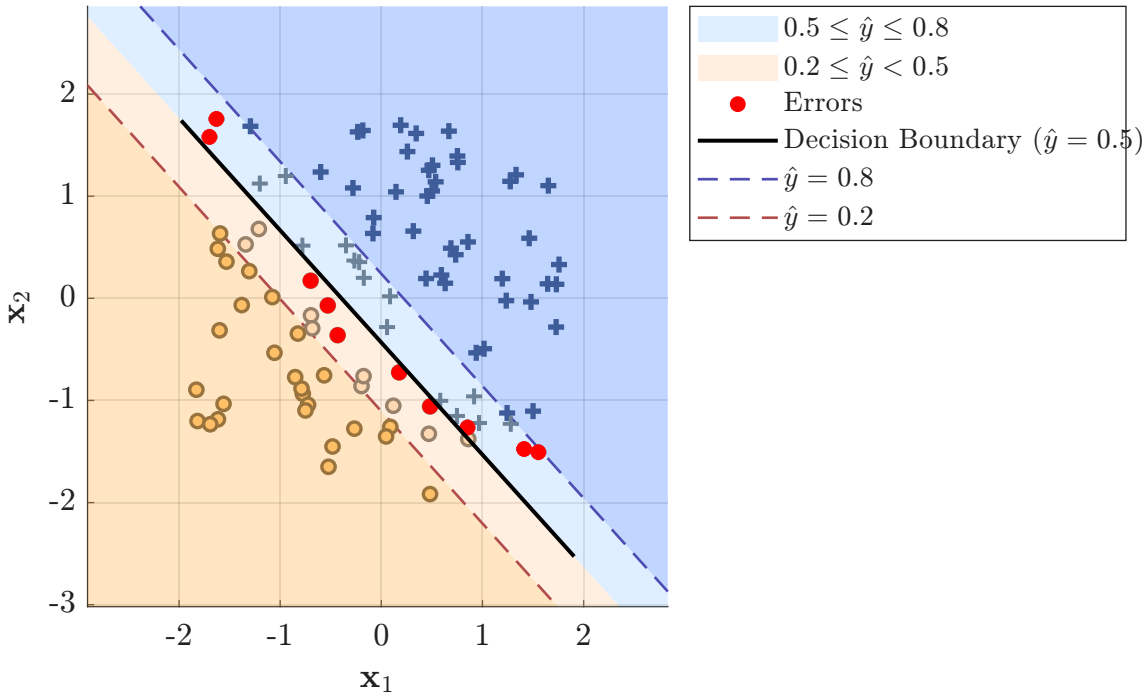


Figure 8.6.: Probabilistic interpretation: regions of predicted probability $\hat{y} = \sigma(\mathbf{w}^T\mathbf{x})$ with boundaries at 0.2 and 0.8. Misclassified points are marked in red (see also Fig. 8.1).

Loss minimization

Substituting $\hat{\mathbf{y}} = \sigma(\mathbf{X}\mathbf{w})$ into the BCE loss gives the logistic regression objective in vector form:

$$\mathcal{L} = \frac{1}{M}\left[-\mathbf{y}^T \log\big(\sigma(\mathbf{X}\mathbf{w})\big) - (1-\mathbf{y})^T \log\big(1-\sigma(\mathbf{X}\mathbf{w})\big)\right] \tag{8.14}$$

Setting $\nabla_{\mathbf{w}}\mathcal{L} = \mathbf{0}$ has no closed-form solution for $\mathbf{w}$. However, the gradient has a compact form:

$$\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}) = \frac{1}{M}\mathbf{X}^T\left(\sigma\left(\mathbf{X}\mathbf{w}\right) - \mathbf{y}\right) \tag{8.15}$$

which enables gradient descent using only matrix–vector operations:

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \alpha\nabla_{\mathbf{w}}\mathcal{L}(\mathbf{w}) \tag{8.16}$$

Decision boundary

From Eq. (8.13), the classification rule is

$$\hat{\mathbf{y}} = \begin{cases} 1 & \mathbf{w}^T\mathbf{x} \geq 0 \\ 0 & \mathbf{w}^T\mathbf{x} < 0 \end{cases} \tag{8.17}$$

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 $\mathbf{w}$.

Regularization and feature mapping

- **Regularization.** L_2 regularization can be applied, similar to Eq. (5.13):

$$\mathcal{L}_{\text{reg}} = \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) + \frac{\lambda}{2M}\sum_{i=1}^{N} w_i^2 \tag{8.18}$$

- **Feature mapping.** Mapping functions or kernels extend the model to non-linear decision boundaries.

Example 8.6: For example, a polynomial mapping

$$\varphi(x_1, x_2) = \langle 1, x_1, x_1^2, x_2, x_2^2, x_1x_2, x_1^2x_2, x_1x_2^2, x_1^2x_2^2\rangle$$

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).

8.6. Odd and Logit

If $\Pr(Y = 1) = p$, than the **odds** (probability of event divided by probability of no event) is defined by ratio

$$\text{odds} = \frac{\Pr(y=1)}{\Pr(y=0)} = \frac{\Pr(y=1)}{1-\Pr(y=1)} = \frac{p}{1-p} \tag{8.19}$$

The odds range from $[0, \infty)$.

Taking the natural logarithm gives the **logit** function (or log-odds):

$$\text{logit}(p) = \ln\left(\frac{p}{1-p}\right) \tag{8.20}$$

The logit maps probabilities from the range $(0, 1)$ to the entire real line $(-\infty, \infty)$.

Logistic Regression As already mentioned in (8.13),

$$\Pr(y = 1) = \sigma(\mathbf{w}^T\mathbf{x}) = \frac{1}{1 + \exp\left(-\mathbf{w}^T\mathbf{x}\right)} = p \tag{8.21}$$

By reformulation,

$$\frac{p}{1-p} = \frac{\Pr(y=1)}{\Pr(y=0)} = \exp\left(\mathbf{w}^T\mathbf{x}\right) \tag{8.22}$$

$$\ln\left(\frac{p}{1-p}\right) = \mathbf{w}^T\mathbf{x} \tag{8.23}$$

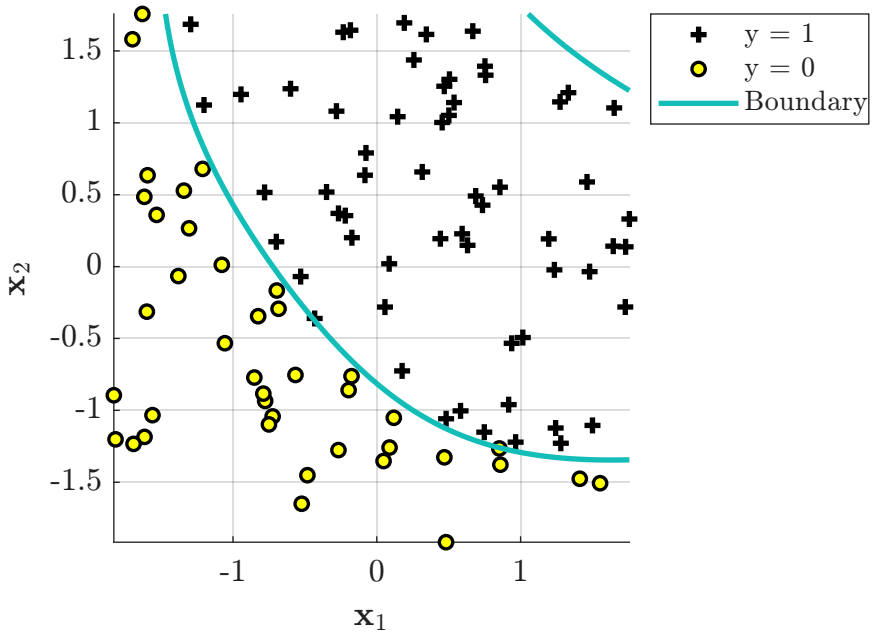
To illustrate the influence of $\tilde{x}_i = x_i + \Delta x$,

$$\begin{aligned}\frac{\text{odds}_{\Delta x}}{\text{odds}} &= \frac{\exp\left(w_0 + w_1x_1 + \cdots + w_i(x_i + \Delta x) + \cdots + w_Nx_n\right)}{\exp(w_0 + w_1x_1 + \cdots + w_ix_i + \cdots + w_Nx_n)} \\ &= \exp\left(w_i(x_i + \Delta x) - w_ix_i\right) = \exp(w_i\Delta x)\end{aligned} \tag{8.24}$$

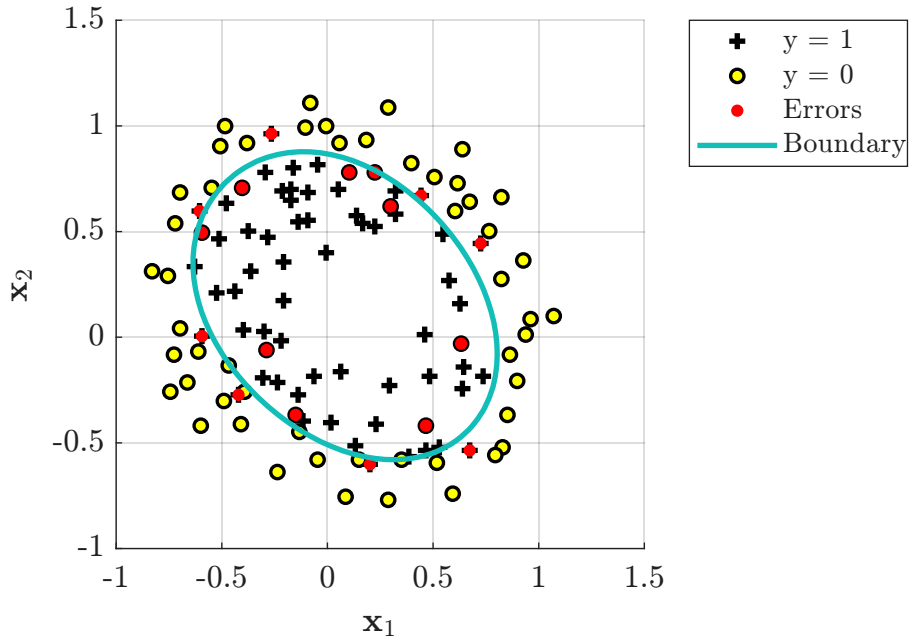
Add numerical example

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.



(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.

9. 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 $\hat{\mathbf{y}} = f_{\boldsymbol{\theta}}(\mathbf{X})$.

Typically, in binary classification, $y_i \in \{0, 1\}$.

Contents

9.1. 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

Positive/negative terminology is rather arbitrary. Typically, the result of interest is termed positive.

9.2. 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 9.1.: 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.

9.3. Performance Metrics

Goal: Characterization is useful to compare classifiers and/or performance on different datasets.

9.3.1. Accuracy

Goal: The most intuitive metric, fraction of $Y = \hat{Y}$, among all the classification results, $\Pr\left(Y = \hat{Y}\right)$.

$$\begin{aligned} \text{Accuracy} &= \frac{\text{correct predictions}}{\text{total predictions}} \\ &= \frac{TP + TN}{TP + TN + FP + FN} \end{aligned} \tag{9.1}$$

Example 9.1: Covid antibody (fast non-PCR) test performance. The example includes test statistics of 239 participants [9], as presented below.

		Predicted	
		Yes	No
Actual	Yes	141	67
	No	0	31

The resulting accuracy is

$$\text{Accuracy} = \frac{141 + 31}{239} = 0.7196652 \approx 72.0\% \tag{9.2}$$

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.

9.3.2. Precision

Goal: Proportion of positive classification that is actually correct, $\Pr\Big(Y = 1 | \hat{Y} = 1\Big)$.

From the probability theory,

$$\Pr\Big(Y = 1 | \hat{Y} = 1\Big) = \frac{\Pr\Big(Y = 1, \hat{Y} = 1\Big)}{\Pr\Big(\hat{Y} = 1\Big)} \tag{9.3}$$

$$\Pr\Big(\hat{Y} = 1\Big) = \Pr\Big(\hat{Y} = 1, Y = 0\Big) + \Pr\Big(\hat{Y} = 1, Y = 1\Big) \tag{9.4}$$

$$\text{Precision} = \frac{TP}{FP + TP} = \frac{\text{Correctly predicted 1's}}{\text{All predicted 1's}} \tag{9.5}$$

TP = Correctly predicted 1's
TP + FP = All predicted 1's

Example 9.1: Back to the previous example,

$$\text{Precision} = \frac{141}{0 + 141} = 1 = 100\% \tag{9.6}$$

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.

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 9.1.: Radar interpretation of the classification metrics.

9.3.3. Recall (sensitivity)

Goal: Proportion of positives identified correctly, $\Pr\left(\hat{Y}=1|Y=1\right)$.

From the probability theory,

$$\Pr\left(\hat{Y}=1|Y=1\right)=\frac{\Pr\left(Y=1,\hat{Y}=1\right)}{\Pr(Y=1)} \tag{9.7}$$

$$\Pr(Y=1)=\Pr\left(Y=1,\hat{Y}=0\right)+\Pr\left(Y=1,\hat{Y}=1\right) \tag{9.8}$$

$$\text{Recall}=\frac{TP}{TP+FN}=\frac{\text{Correctly predicted 1's}}{\text{Actual 1's}} \tag{9.9}$$

Medical meaning: portion of correctly classified ill among all the ill.

Example 9.1: Back to the previous example,

$$\text{Recall}=\frac{141}{141+67}=0.678=67.8\% \tag{9.10}$$

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.

9.3.4. Specificity

Goal: Proportion of negatives identified correctly, $\Pr\left(\hat{Y}=0|Y=0\right)$.

$$\text{Specificity}=\frac{TN}{FP+TN}=\frac{\text{Correctly predicted 0's}}{\text{Actual 0's}} \tag{9.11}$$

Medical meaning: portion of classified healthy among all the healthy.

Example 9.1: Back to the previous example,

$$\text{Specificity} = \frac{31}{0+31} = 1 = 100\% \tag{9.12}$$

From the medical point of view, all negative results are really negative.

9.3.5. F_1-score

Goal: Combination of precision and recall.

The harmonic mean between precision and recall,

$$F_1 = \frac{2}{\frac{1}{recall} + \frac{1}{precision}} = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \tag{9.13}$$

Example 9.1: Back to the previous example,

$$F_1 = \frac{141}{141 + \frac{1}{2}(0+67)} = 0.808 = 80.8\% \tag{9.14}$$

Example 9.2: The logistic regression classifier from Figs. 8.1 and 8.6 has the following confusion matrix:

		Predicted	
		Yes	No
Actual	Yes	55	5
	No	5	35

The resulting metrics are

$$\begin{aligned}
\text{Accuracy} &= \frac{55+35}{100} = 0.9 = 90\% \\
\text{Precision} &= \frac{55}{5+55} = \frac{55}{60} \approx 91.7\% \\
\text{Recall} &= \frac{55}{55+5} = \frac{55}{60} \approx 91.7\% \\
\text{Specificity} &= \frac{35}{5+35} = \frac{35}{40} = 87.5\% \\
F_1 &= \frac{55}{55 + \frac{1}{2}(5+5)} = \frac{55}{60} \approx 91.7\%
\end{aligned} \tag{9.15}$$

Note, since $FP = FN$, precision, recall, and F_1 are all equal.

9.3.6. Per-class performance

The metrics above are defined with respect to the positive class ($Y = 1$). To assess both classes, the metrics can be computed separately for each class by treating it 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

The per-class metrics are then

$$\text{Precision}_c = \frac{TP_c}{TP_c + FP_c}, \quad \text{Recall}_c = \frac{TP_c}{TP_c + FN_c}, \quad F_{1,c} = \frac{2 \cdot \text{Precision}_c \cdot \text{Recall}_c}{\text{Precision}_c + \text{Recall}_c} \tag{9.16}$$

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 9.1: Back to the Covid antibody test example (Example 9.1, 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.

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 9.2.: Per-class metrics for the Covid antibody test.

Example 9.2: Back to the example 9.2. The classifier performs better on class 1 (the majority class with 60 samples) than on class 0 (40 samples).

9.4. Imbalanced Dataset

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 9.3: 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 $\hat{Y} = 0$?

Metric	Class 1	Class 0
Precision	$\frac{55}{55+5} \approx 91.7\%$	$\frac{35}{35+5} = 87.5\%$
Recall	$\frac{55}{55+5} \approx 91.7\%$	$\frac{35}{35+5} = 87.5\%$
F_1	$\frac{2 \cdot 0.917 \cdot 0.917}{0.917+0.917} \approx 91.7\%$	$\frac{2 \cdot 0.875 \cdot 0.875}{0.875+0.875} = 87.5\%$

Table 9.3.: Per-class metrics for the logistic regression example.

Solution: The resulting confusion matrix is

		Predicted	
		Yes	No
Actual	Yes	0	10
	No	0	990

and the resulting quantities are

$$\begin{aligned} \text{Accuracy} &= \frac{990}{1000} = 0.99 = 99\% \\ \text{Precision} &= \frac{TP}{FP+TP} = \frac{0}{0+0} = \text{Undefined} \\ \text{Recall} &= \frac{TP}{TP+FN} = \frac{0}{0+10} = 0 \\ \text{Specificity} &= \frac{TN}{FP+TN} = \frac{990}{1000} = 0.99 = 99\% \\ F_1 &= \frac{TP}{TP + \frac{1}{2}(FP+FN)} = \frac{0}{\cdots} = 0 \end{aligned} \tag{9.17}$$

- Note, accuracy is insufficient metrics!
- Note, while the convention is to label '1' for the most important class outcome, sometimes it is interchangeable.

Majority classifier

Majority classifier is a classifier that always predicts the most frequent class in the dataset, as in Example 9.3.

Despite achieving high accuracy on imbalanced data, a majority classifier has zero recall and zero F_1 for the minority class. It is commonly used as a baseline: any useful classifier should outperform it on metrics beyond accuracy.

Small dataset problem

In imbalanced datasets, the minority class may contain very few samples. Performance metrics computed on small subsets are subject to high sampling variability, making them unreliable estimates of the true classifier performance.

Example 9.4: Consider the dataset from Example 9.3 with only $n = 10$ positive samples. Assume a classifier with *true* per-sample accuracy of $p = 0.8$ on class '1'. What is the probability that it correctly classifies 6 or fewer of the 10 positive samples?

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

$$\Pr(X \leq 6) = \sum_{k=0}^{6} \binom{10}{k} p^k (1-p)^{10-k} \approx 12.09\%$$

There is a ≈12% chance of observing recall ≤ 60%, despite the true accuracy being 80%. Conversely, $\Pr(X = 10) = 10.74\%$: even perfect recall is plausible by chance for an imperfect classifier.

This is a problem of *confidence* in performance evaluation. While confidence interval analysis is out of the scope of this document, the key takeaway is: metrics computed on small subsets can deviate significantly from the true performance and should be interpreted with caution.

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.

9.5. Multi-class performance

9.5.1. Categorical Encoding

Goal: Convert categorical (non-numeric) features into numeric representations suitable for mathematical models.

Label encoding[1] assigns each category an integer. For a feature with categories $\{A, B, C\}$, label encoding maps $A \mapsto 0$, $B \mapsto 1$, $C \mapsto 2$. This is appropriate for ordinal features where the order is meaningful (e.g., low, medium, high).

One-hot encoding[2] creates a binary indicator column for each category. A feature with C categories is replaced by C binary features, increasing the dimensionality from N to $N + C - 1$.

Example 9.5: Consider a feature "Color" with three categories: $\{\text{Red}, \text{Green}, \text{Blue}\}$ and $M = 4$ samples:

[1] In Python: `sklearn.preprocessing.LabelEncoder`
[2] In Python: `sklearn.preprocessing.OneHotEncoder`

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.

Pros:

- One-hot encoding avoids imposing artificial ordinal relationships between categories.
- Label encoding is memory-efficient and preserves dimensionality.

Cons:

- One-hot encoding increases dimensionality significantly for high-cardinality features.
- Label encoding introduces a spurious ordinal relationship for nominal features.

9.5.2. Performance

For multi-class classification, confusion matrices, accuracy and per-class performance (Sec. 9.3.6) are used.

To get a single number, two averaging strategies are applied.

Macro-averaging

Averaging with *equal weight to each class*, regardless of the number of instances.

Micro-averaging

Equal weight to each instance, regardless of the class label and the number of instance in the class. All TP, FP, etc. are summed across all the classes.

Further reading: Accuracy, precision, and recall in multi-class classification

9.6. Decision threshold

Goal: Quantify the trade-off between confusion matrix elements as a function of the decision threshold.

With a probabilistic classifier, the output is the estimated probability of the positive class,

$$\Pr(\hat{y} = 1 \mid \mathbf{x}) = f_{\boldsymbol{\theta}}(\mathbf{x}) \tag{9.18}$$

The binary decision is obtained by comparing $f_{\boldsymbol{\theta}}(\mathbf{x})$ with a threshold thr:

$$\hat{y} = \begin{cases} 1 & f_{\boldsymbol{\theta}}(\mathbf{x}) \geq \mathsf{thr} \\ 0 & f_{\boldsymbol{\theta}}(\mathbf{x}) < \mathsf{thr} \end{cases} \tag{9.19}$$

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).

9.6.1. Receiver Operating Characteristics (RoC)

Goal: Visualize classifier performance across all thresholds.

The RoC curve plots the following two quantities as thr varies from 0 to 1:

- **True Positive Rate (TPR)**: synonym for recall,

$$TPR = \frac{TP}{TP + FN} = \text{recall} \tag{9.20}$$

- **False Positive Rate (FPR)**: complement of specificity,

$$FPR = \frac{FP}{FP + TN} = 1 - \text{specificity} \tag{9.21}$$

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.

9.6.2. 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. 9.2. 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. 9.3 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:

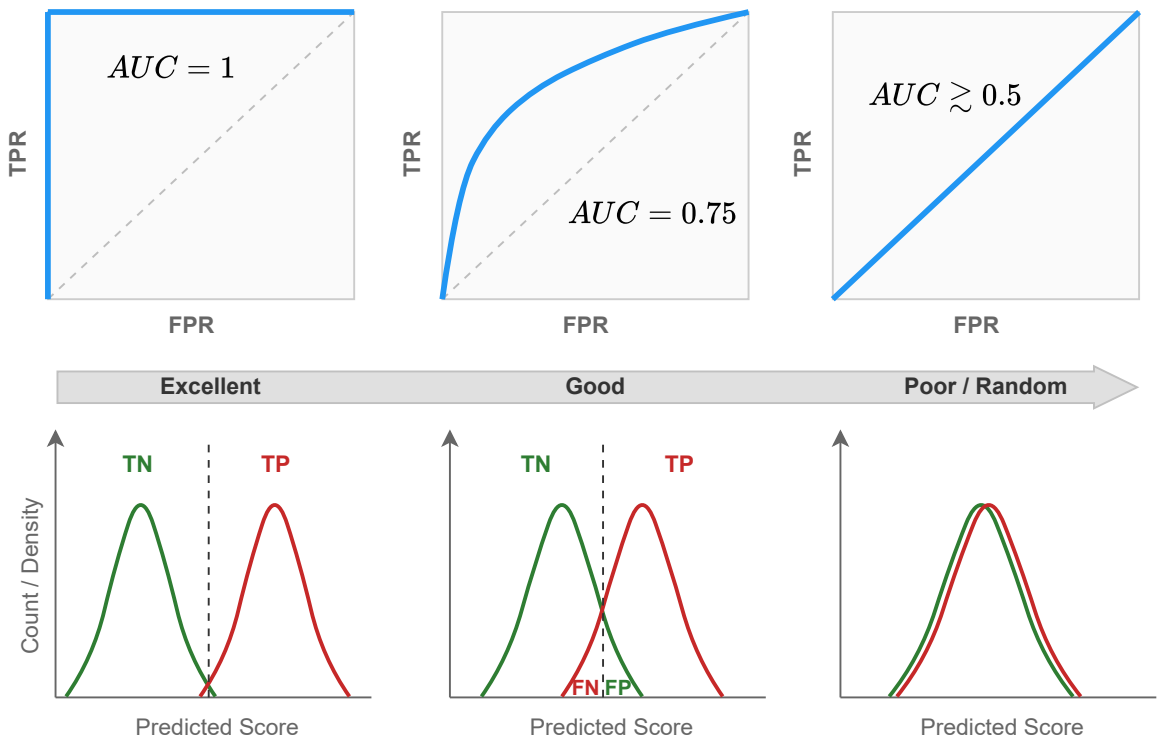


Figure 9.2.: RoC curves (top) and corresponding class distributions (bottom) for three classifiers. Well-separated distributions yield AUC ≈ 1; overlapping distributions produce FN/FP errors and AUC $\to 0.5$.

- **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.

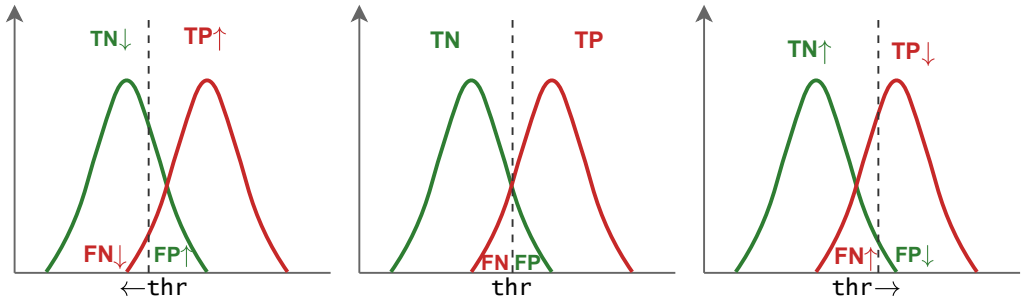


Figure 9.3.: Effect of threshold on classification: lowering thr (left) reduces FN but increases FP; raising thr (right) reduces FP but increases FN. The arrows indicate the direction of change relative to the default threshold (center).

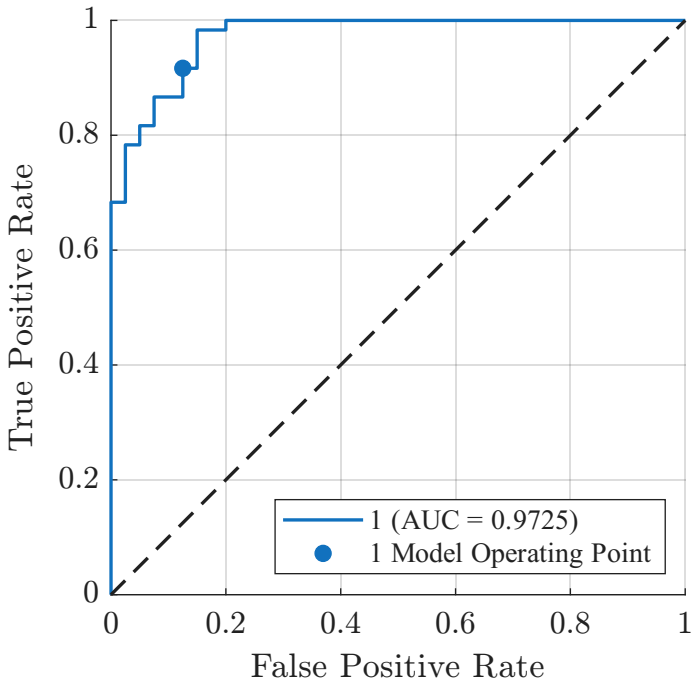


Figure 9.4.: RoC of logistic regression example in Fig. 8.1. The model operation point is thr = 0.5.

10. Classifiers

10.1. Support Vector Machine (SVM)

Goal: Binary classifier that finds the decision boundary maximizing the margin between classes.

The central idea is to find a hyperplane that separates two classes while maximizing the distance (margin) to the nearest training points. Only the data points closest to the boundary, called *support vectors*, determine the classifier (Fig. 10.1).

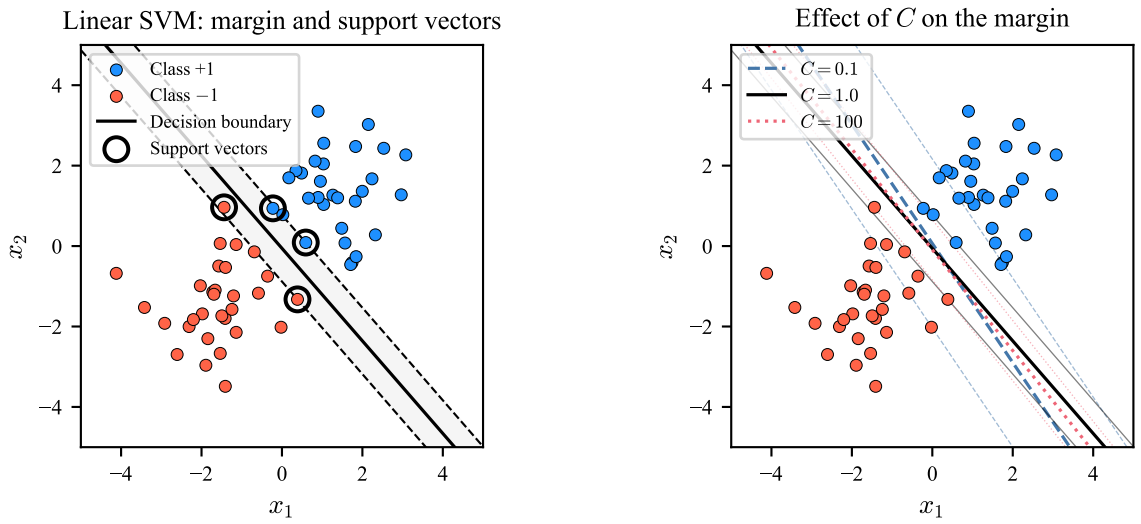


Figure 10.1.: Left: linear SVM with decision boundary (solid), margin boundaries (dashed), and support vectors (circled). Right: effect of the regularization parameter C on margin width — small C yields a wider margin, large C a narrower one.

10.1.1. Linear SVM

Consider a binary classification problem with labeled data:

$$\{(x_i, y_i)\}_{i=1}^{n}, \quad y_i \in \{-1, +1\}$$

A linear classifier is defined as:

$$f(x) = w^\top x + b$$

The decision boundary is given by:

$$w^\top x + b = 0$$

For a hard-margin SVM, the constraints are:

$$y_i(w^\top x_i + b) \geq 1 \quad \forall i$$

The distance from a point x to the hyperplane $w^\top x + b = 0$ is $|w^\top x + b|/\|w\|$. The two margin boundaries $w^\top x + b = \pm 1$ each lie at distance $1/\|w\|$ from the decision boundary, so the total margin width is

$$\text{Margin width} = \frac{2}{\|w\|}.$$

Maximizing the margin is therefore equivalent to minimizing $\|w\|^2$.

Hard margin

When data is perfectly separable, the hard-margin SVM solves:

$$\min_{w,b} \frac{1}{2}\|w\|^2 \qquad \text{s.t.} \quad y_i(w^\top x_i + b) \geq 1 \quad \forall i \tag{10.1}$$

This is a convex quadratic program with a unique global optimum.

Soft margin

Real-world data often contains noise and class overlap. Soft margin SVM introduces slack variables $\xi_i \geq 0$ that allow margin violations:

$$\min_{w,b,\xi} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i \qquad \text{s.t.} \quad y_i(w^\top x_i + b) \geq 1 - \xi_i \tag{10.2}$$

Interpretation of ξ_i:

ξ_i	Interpretation
0	Correctly classified, outside the margin
$0 < \xi_i < 1$	Correctly classified, inside the margin
$= 1$	On the decision boundary
> 1	Misclassified

The parameter $C > 0$ controls the trade-off: large C penalizes violations heavily (narrow margin, risk of overfitting), small C allows more violations (wider margin, better robustness).

Hinge loss

Soft margin SVM is equivalent to minimizing hinge loss with L2 regularization:

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i} \max(0, 1 - y_i f(x_i)) \tag{10.3}$$

Only points violating the margin ($y_i f(x_i) < 1$) contribute to the loss (Fig. 10.2).

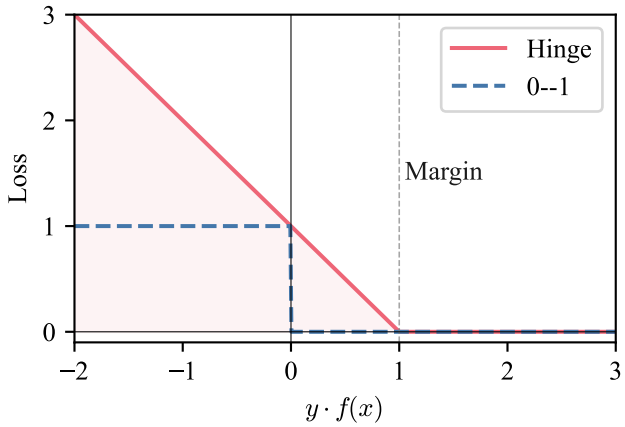


Figure 10.2.: Hinge loss (solid) compared to the 0–1 loss (dashed). The shaded region shows where hinge loss penalizes points inside or violating the margin ($y \cdot f(x) < 1$).

10.1.2. Kernel SVM

When data is not linearly separable, SVM maps inputs into a higher-dimensional feature space $\phi : \mathbb{R}^d \to \mathbb{R}^D$. Kernels compute inner products in this space without explicit mapping:

$$K(x, z) = \phi(x)^\top \phi(z) \tag{10.4}$$

Common kernels: linear, polynomial, radial basis function (RBF), and sigmoid.

Example 10.1: Fig. 10.3 shows SVM with three kernels on concentric-circle data (not linearly separable). The linear kernel fails to separate the classes. The polynomial kernel ($d = 2$) and RBF kernel both produce circular decision boundaries that correctly separate the inner and outer rings. Support vectors are circled.

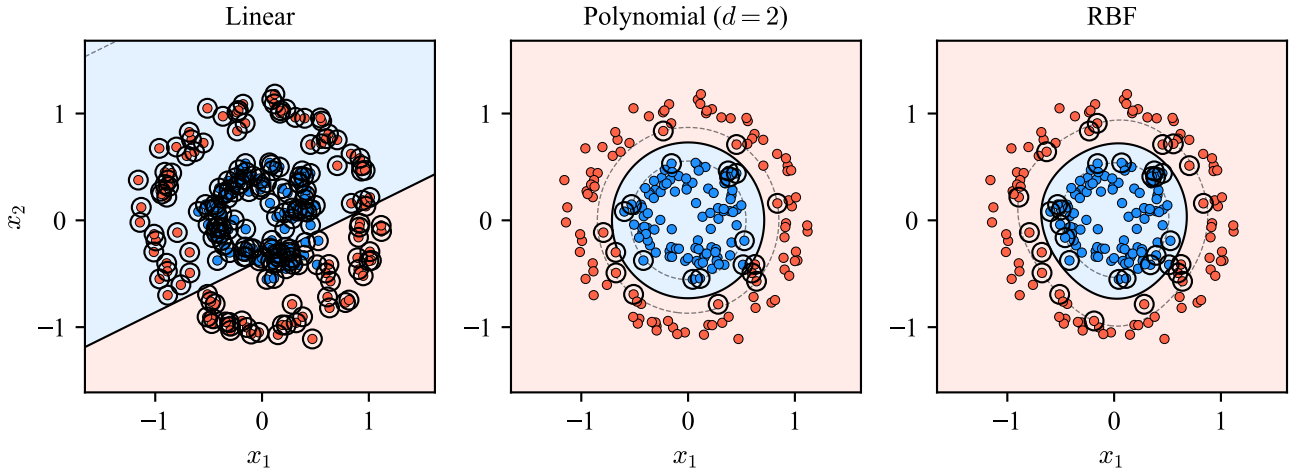


Figure 10.3.: Kernel SVM on non-linearly separable data. Linear kernel (left) cannot separate the classes; polynomial (center) and RBF (right) kernels produce nonlinear boundaries. Dashed lines show the margin.

10.1.3. Summary

Pros:

- Strong generalization via margin maximization.
- Effective in high-dimensional spaces.
- Robust against overfitting (controlled by C).

Cons:

- Not suitable for very large datasets (training scales poorly).
- Non-probabilistic output (no direct class probabilities).
- Kernel and C selection require cross-validation.

Hyperparameters: C (regularization), kernel type, kernel parameters (e.g. γ for RBF). Feature scaling is required.

In Python: `scikit-learn` (`SVC`, `LinearSVC`).

10.2. Decision Trees

Goal: Recursive partitioning classifier that splits the feature space into axis-aligned regions, each assigned a class label.

A decision tree classifies a sample by traversing a binary tree from root to leaf. At each internal node, a single feature x_j is compared to a threshold t: the sample goes left if $x_j \leq t$ and right otherwise. Each leaf assigns a class label.

Example 10.2: Fig. 10.4 compares three classifiers on the same dataset. A shallow tree (depth = 2) produces coarse axis-aligned regions. An unrestricted tree perfectly fits the training data but creates a complex, overfitting boundary. A random forest (100 trees) smooths the boundary by averaging many individual trees, reducing variance.

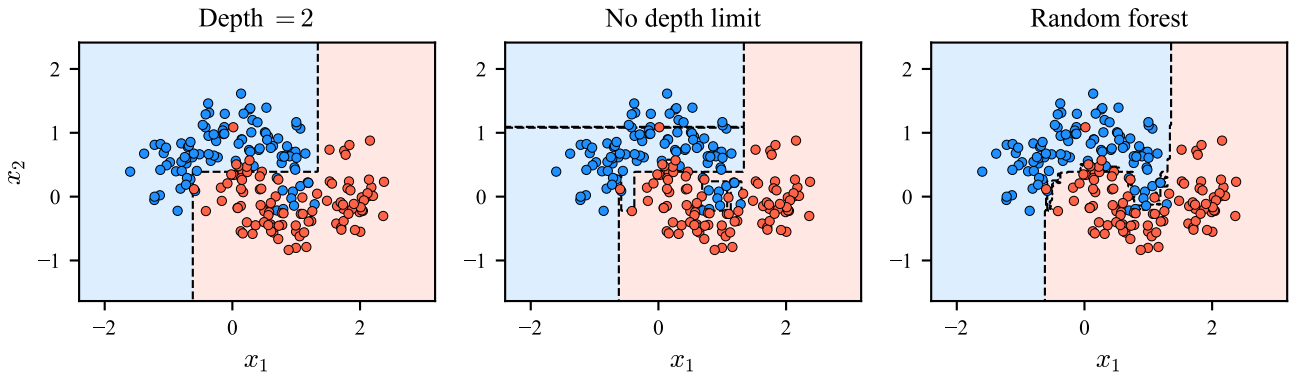


Figure 10.4.: Decision boundaries for a shallow tree (left), an unrestricted tree (center), and a random forest (right). Note the axis-aligned splits characteristic of tree-based methods.

10.2.1. Tree Construction (CART)

The Classification and Regression Tree (CART) algorithm builds the tree greedily:

1. For each node, consider all features and all possible thresholds.
2. Select the split (j, t) that maximizes a purity criterion (see below).
3. Partition the data and recurse on each child node.
4. Stop when a stopping criterion is met (e.g. maximum depth, minimum samples per leaf, or no further purity improvement).

10.2.2. Splitting Criteria

Let p_c denote the fraction of samples belonging to class c at a given node.

Gini impurity

$$G = 1 - \sum_c p_c^2 \tag{10.5}$$

Measures the probability that a randomly chosen sample would be misclassified if labeled according to the class distribution at the node. $G = 0$ for a pure node.

Entropy (information gain)

$$H = -\sum_c p_c \log_2 p_c \tag{10.6}$$

The split that yields the largest reduction in entropy (information gain) is selected. Entropy and Gini often produce similar trees; Gini is computationally cheaper.

10.2.3. Regularization

Without constraints, a tree can grow until every leaf contains a single sample, perfectly fitting the training data but overfitting. Common regularization strategies:

- **Maximum depth**: limit the number of levels.
- **Minimum samples per leaf**: require at least $n_{\min}$ samples in each leaf.
- **Pruning**: grow a full tree, then remove branches that do not improve validation performance.

10.2.4. Random Forest

A random forest is an ensemble of decision trees trained on bootstrap samples of the data. At each split, only a random subset of features is considered. The final prediction is the majority vote across all trees.

This reduces variance compared to a single tree while preserving the ability to model complex boundaries.

10.2.5. Summary

Pros:

- Interpretable: the tree can be visualized and each decision explained.
- No feature scaling required.
- Handles both numerical and categorical features.
- Fast inference: $\mathcal{O}(\text{depth})$ per prediction.

Cons:

- Single trees are prone to overfitting (high variance).
- Axis-aligned splits limit expressiveness for rotated decision boundaries.
- Greedy construction does not guarantee a globally optimal tree.

Hyperparameters: maximum depth, minimum samples per leaf, splitting criterion (Gini or entropy). For random forests: number of trees, number of features per split.

In Python: `scikit-learn` (`DecisionTreeClassifier`, `RandomForestClassifier`).

10.3. 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.

10.3.1. Algorithm

For a new sample $\mathbf{x}$:

1. Compute distance $d(\mathbf{x}, \mathbf{x}_i)$ to all training samples.
2. Select the k samples with smallest distances.
3. Assign the class by majority vote among the k neighbors.

Example 10.3: Fig. 10.5 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.

10.3.2. Distance Metrics

The most popular distance metrics are:

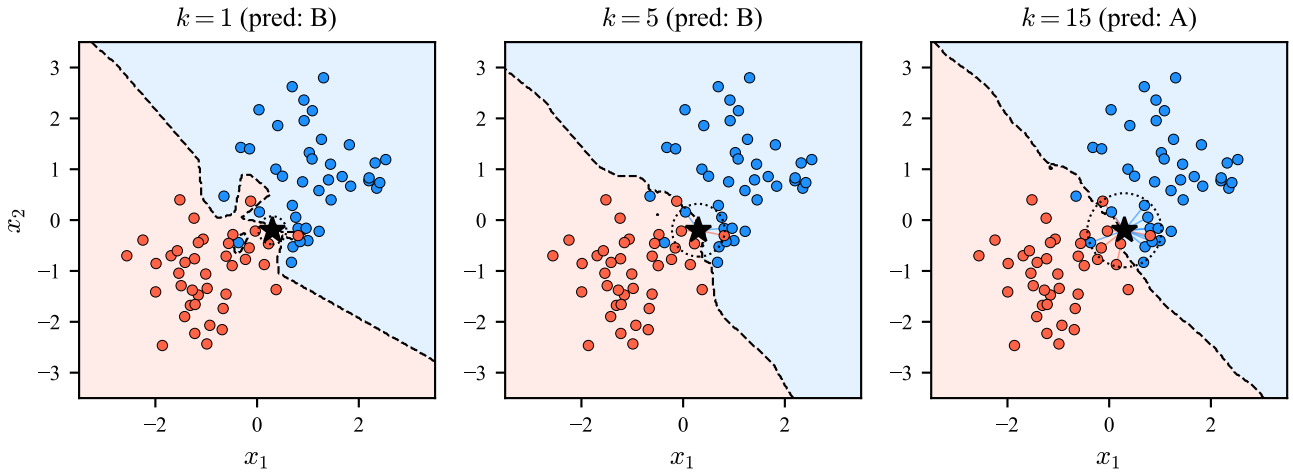


Figure 10.5.: k-NN decision boundaries for different values of k. The query point (star) and its k nearest neighbors (connected by lines) are shown. Larger k smooths the boundary.

Euclidean (L_2)

$$d(\mathbf{a}, \mathbf{b}) = \|\mathbf{a} - \mathbf{b}\| = \sqrt{\sum_{j=1}^{N} (a_j - b_j)^2} \tag{10.7}$$

Manhattan (L_1, city block)

$$d(\mathbf{a}, \mathbf{b}) = \sum_{j=1}^{N} |a_j - b_j| \tag{10.8}$$

Minkowski (generalized, hyperparameter p)

$$d(\mathbf{a}, \mathbf{b}) = \left(\sum_{j=1}^{N} |a_j - b_j|^p \right)^{1/p} \tag{10.9}$$

Special cases: $p = 1$ gives Manhattan, $p = 2$ gives Euclidean.

10.3.3. 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.

10.3.4. Curse of Dimensionality

In high-dimensional spaces, distances become less meaningful[1]:

[1] Illustration by Matlab simulation.

- All points tend to be equidistant from each other.
- The ratio of nearest to farthest neighbor distance approaches 1.
- Most volume of a hypersphere concentrates near its surface.

As N grows, k-NN performance degrades unless M grows exponentially.

10.3.5. Summary

Pros:

- Simple, non-parametric, no training phase.
- Natural probabilistic interpretation (fraction of neighbors).
- Can model complex decision boundaries.

Cons:

- High inference cost: $\mathcal{O}(M \cdot N)$ per prediction (N-times M-dimensional distance).
- Sensitive to irrelevant features and scale (normalization required).
- Sensitive to outliers (especially for small k).
- Performance degrades in high dimensions (curse of dimensionality).

Hyperparameters: k (number of neighbors), distance metric.

11. Feature Engineering and Selection

Goal: Working with feautures:

- Feature engineering (FE) transforms raw features into representations that improve model performance.
- 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

11.1. Feature Engineering

Feature engineering creates or transforms features to improve model performance, while feature selection chooses a subset from existing ones. In a typical pipeline, feature engineering is applied first, followed by feature selection.

11.1.1. Feature Transformations

Goal: Create new features from existing ones to capture non-linear relationships.

Polynomial features[1] 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 is closely related to the polynomial kernel approach (see Kernels chapter).

Example 11.1: For two features x_1, x_2 with degree $d = 2$, the expanded feature set is:

$$[x_1, x_2] \to [1,\ x_1,\ x_2,\ x_1^2,\ x_1x_2,\ x_2^2]. \tag{11.1}$$

[1] In Python: `sklearn.preprocessing.PolynomialFeatures`

The dimensionality increases from $N = 2$ to $N' = 6$ (including the bias term).

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 converts multiplicative relationships into additive ones.

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.

Pros:

- Polynomial features enable linear models to fit non-linear patterns.
- Log transform stabilizes variance and reduces the influence of outliers.
- Binning is robust to outliers and can simplify the model.

Cons:

- Polynomial features cause rapid dimensionality growth: $\binom{N+d}{d}$ terms for degree d.
- Log transform is only applicable to strictly positive values.
- Binning discards information within each bin.

11.1.2. Date and Time Features

Goal: Extract informative features from date and time variables to capture temporal patterns.

A single datetime variable (e.g., a timestamp) carries no direct numerical meaning for most models. Extracting its components creates features that expose cyclic and calendar-based patterns:

- *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:

$$x_{\sin} = \sin\left(\frac{2\pi\, x}{P}\right), \quad x_{\cos} = \cos\left(\frac{2\pi\, x}{P}\right), \tag{11.2}$$

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 11.2: 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

Pros:

- Exposes temporal patterns that are invisible in raw timestamps.
- Cyclical encoding preserves the natural proximity of cyclic values.

Cons:

- Increases dimensionality, especially with cyclical encoding (two features per component).
- Requires domain knowledge to choose which components are relevant.

11.2. Feature Selection

Goal: Select a subset of relevant features to reduce dimensionality and improve model performance.

Motivation:

- Some features are redundant and are very similar to each other.
- Some features are noise, i.e does not contribute to the performance of a model.
- Each feature has its (significant) computation complexity.
- Enhanced generalization by reducing overfitting
- Shorter training times and lesser computational cost

Exhaustive full search over all possible combination of features is typically computationally prohibitive. For example, checking all possible subset combinations of $N = 15$ features includes $\sum_{k=1}^{N} \binom{N}{k} = 32767$ possible combinations.

All of the practical FS methods use sub-optimal search that significantly reduce the number of the examined combinations. An example of feature-number-aware metric is adjusted R^2 in Sec. 3.2.2.

11.3. Classifier-Based

Goal: The influence of each feature is evaluated by its influence on the classifier.

Greedy search

Greedy search methods build a feature subset incrementally by making locally optimal choices at each step. While they do not guarantee a globally optimal solution, they significantly reduce computational complexity from exponential to polynomial.

Forward feature selection starts with an empty feature set and iteratively adds the feature that most improves model performance. At each step, every remaining feature is evaluated, and the one yielding the best performance gain is added. This process continues until adding more features no longer improves performance or a predetermined number of features is reached.

Backward feature selection begins by constructing a model with all available features. It then eliminates the one that produces the highest performing model based on specific evaluation metrics. In subsequent steps, it continues to remove the next least significant feature, improving performance each time. This process is repeated, eliminating one feature at a time, until a predetermined criterion is satisfied.

Forward selection is computationally cheaper when the target subset is small ($N' \ll N$), while backward selection is preferred when only few features need to be removed. Both methods require $\mathcal{O}(N \cdot N')$ model evaluations, compared to $\mathcal{O}(2^N)$ for exhaustive search.

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 greedy nature.
- Computationally expensive: requires retraining the classifier for each candidate subset.
- Hyperparameter tuning may be needed for each set or subset size.

Constant number of the selected features For example, for the pre-defined feature subset size of $N' = 3$ and $N = 15$, it results only in $\binom{N}{N'} = 455$ feature combinations.

11.4. Statistical-Based

There are also statistical FS methods that are not related to a particular classifier.

11.4.1. Variance Threshold

Goal: Eliminates features by their statistical properties.

Variance threshold[2] removes features with variance below a specified threshold. For a given feature, let M be the number of samples, x_i the i-th sample, and $\bar{x}$ the mean. The variance is:

$$\mathrm{Var}[x] = \frac{1}{M} \sum_{i=1}^{M} (x_i - \bar{x})^2. \tag{11.3}$$

Features with $\mathrm{Var}[x] < \tau$ are removed, where τ is the threshold parameter.

This method is useful for removing constant or near-constant features that provide no discriminative information. A common use case is removing features with zero variance (all identical values).

Note that variance threshold is unsupervised and does not consider the target variable.

11.4.2. ANOVA

Goal: Eliminate features by inter-feature statistical properties.

Analysis of Variance (ANOVA)[3] evaluates feature relevance by comparing the variance between class means to the variance within classes. For a given feature, let:

[2] In Python: `sklearn.feature_selection.VarianceThreshold`
[3] 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:

$$SS_B = \sum_{k=1}^{K} M_k(\bar{x}_k - \bar{x})^2. \tag{11.4}$$

The within-class sum of squares measures variability within each class:

$$SS_W = \sum_{k=1}^{K} \sum_{i=1}^{M_k} (x_{k,i} - \bar{x}_k)^2. \tag{11.5}$$

The F-statistic is the ratio of mean squares:

$$F = \frac{SS_B/(K-1)}{SS_W/(M-K)} = \frac{MS_B}{MS_W}, \tag{11.6}$$

where $M = \sum_{k=1}^{K} M_k$ is the total number of samples.

A higher F-value indicates that the feature better separates the classes. Features are ranked by their F-scores, and those with the highest scores are selected. ANOVA assumes features are independent and normally distributed within each class.

Example 11.3: 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.

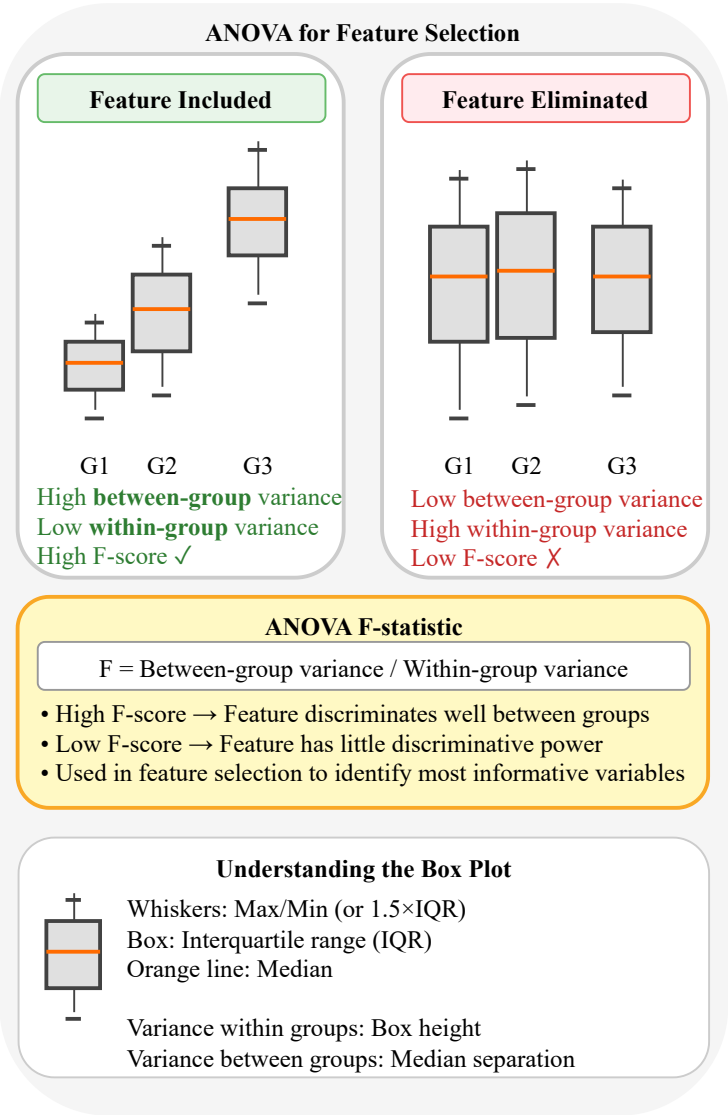


Figure 11.1.: Illustration of ANOVA FS.

11.5. Tree-Based Feature Importance

Decision tree models provide built-in feature importance scores based on how much each feature contributes to reducing impurity (e.g., Gini impurity or entropy) across all splits in the tree.

Pros:

- Fast computation: importance is a byproduct of model training.
- Captures non-linear relationships and feature interactions.

Cons:

- Biased toward high-cardinality features (many unique values).
- Importance is split among correlated features, underestimating individual relevance.

12. Vanilla NN

Goal: One-layer NN.

12.1. Definition

12.1.1. 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. 12.1):

$$a = w_1x_1 + w_2x_2 + \cdots + w_Nx_N + b = \mathbf{w}^T\mathbf{x} + b \tag{12.1}$$

$$z = g(a) \tag{12.2}$$

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 (12.1),
- $g(\cdot)$ 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.

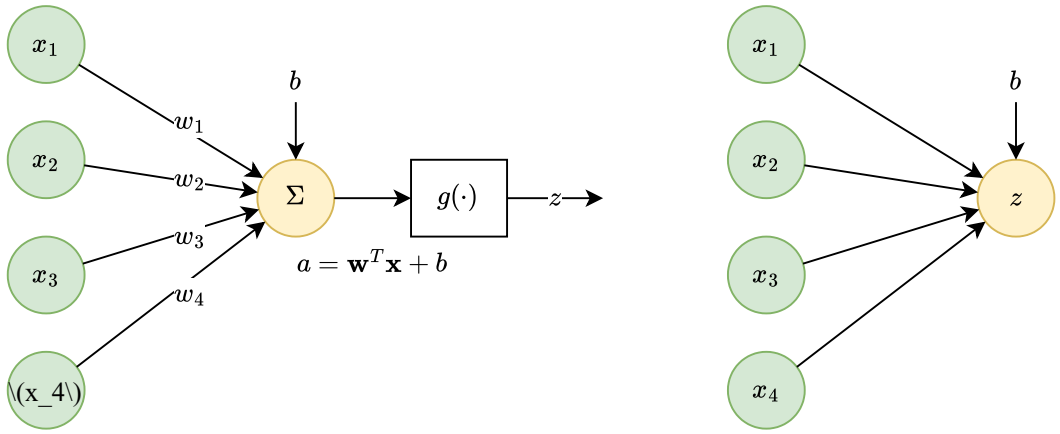
> In the ML part of this book the design matrix $\mathbf{X} \in \mathscr{R}^{M\times N}$ has samples as rows. In DL notation the transposed form $\mathbf{X}^T$ is commonly used, so that samples are columns.

12.1.2. 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:

$$\mathbf{a}^{[k]} = \left(\mathbf{W}^{[k]}\right)^T \mathbf{z}^{[k-1]} + \mathbf{b}^{[k]} \tag{12.3}$$



(a) Fully annotated (b) Compact

Figure 12.1.: Single neuron: fully annotated (left) and compact (right) representation.

$$\mathbf{z}^{[k]} = g_k\left(\mathbf{a}^{[k]}\right) \tag{12.4}$$

For the one-layer network in Fig. 12.2a ($k = 1$, four inputs, five hidden neurons) the pre-activation vector expands as

$$\underbrace{\begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \\ a_5 \end{bmatrix}}_{\mathbf{a}^{[1]}} = \underbrace{\begin{bmatrix} w_{11} & w_{21} & w_{31} & w_{41} \\ w_{12} & w_{22} & w_{32} & w_{42} \\ w_{13} & w_{23} & w_{33} & w_{43} \\ w_{14} & w_{24} & w_{34} & w_{44} \\ w_{15} & w_{25} & w_{35} & w_{45} \end{bmatrix}}_{\left(\mathbf{W}^{[1]}\right)^T} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix}}_{\mathbf{x}} + \underbrace{\begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \end{bmatrix}}_{\mathbf{b}^{[1]}} \tag{12.5}$$

Multi-layer model (single sample):

$$\mathbf{a}^{[k]} = \left(\mathbf{W}^{[k]}\right)^T \mathbf{z}^{[k-1]} + \mathbf{b}^{[k]} \tag{12.6}$$

$$\mathbf{z}^{[k]} = g_k\left(\mathbf{a}^{[k]}\right) \tag{12.7}$$

$$\mathbf{z}^{[0]} = \mathbf{x}_i \tag{12.8}$$

When the entire dataset is processed at once, each column of $\mathbf{Z}^{[k]}$ corresponds to one sample.

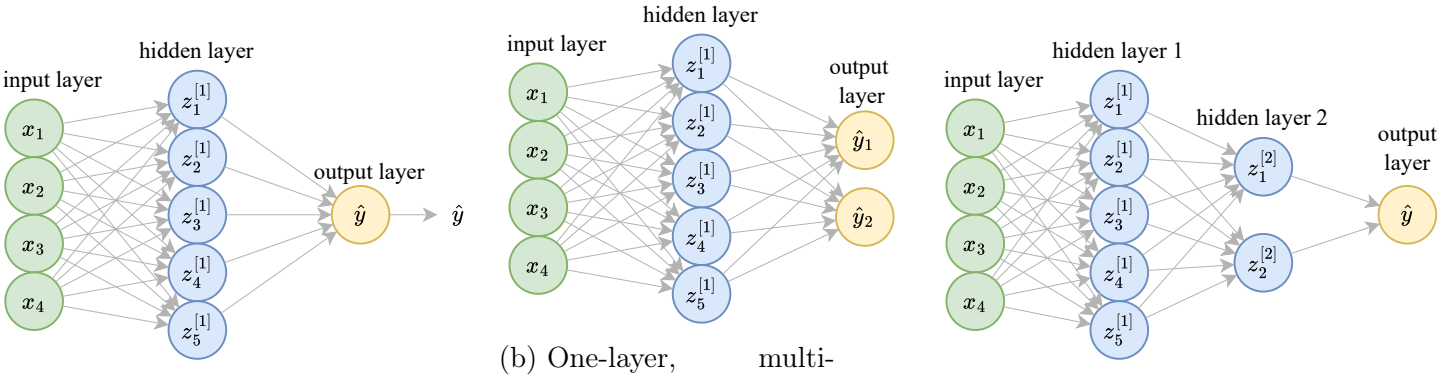
Multi-layer model (all dataset):

$$\mathbf{Z}^{[k]} = g_k\left(\mathbf{W}^{[k]}\mathbf{Z}^{[k-1]} + \mathbf{b}^{[k]}\right) \tag{12.9}$$

$$\mathbf{Z}^{[0]} = \mathbf{X} \tag{12.10}$$

Output is the value of $\mathbf{Z}^{[k]}$ at the output layer.

Fig. 12.2 illustrates three common architectures. A one-layer network with a single output neuron (Fig. 12.2a) performs scalar regression or binary classification. Adding output neurons (Fig. 12.2b) enables multi-target regression or multi-class classification. Stacking hidden layers (Fig. 12.2c) 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 12.2.: 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.

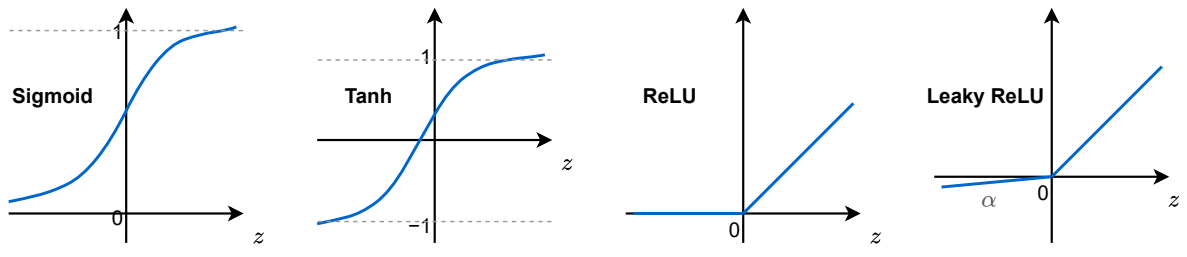
12.2. Activation Functions

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. 12.5 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 12.1 summarizes the common activation functions and their derivatives.



$$\sigma(z) = \frac{1}{1+e^{-z}} \qquad \tanh(z) = \frac{e^{z}-e^{-z}}{e^{z}+e^{-z}} \qquad f(z) = \max(0, z) \qquad f(z) = \max(\alpha z, z)$$

Figure 12.3.: Common activation functions: sigmoid, tanh, ReLU, and Leaky ReLU.

Table 12.1.: 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	$f(z) = \max(\alpha z, z)$	$(-\infty, \infty)$	$f'(z) = \begin{cases} 1 & z \geq 0 \\ \alpha & z < 0 \end{cases}$

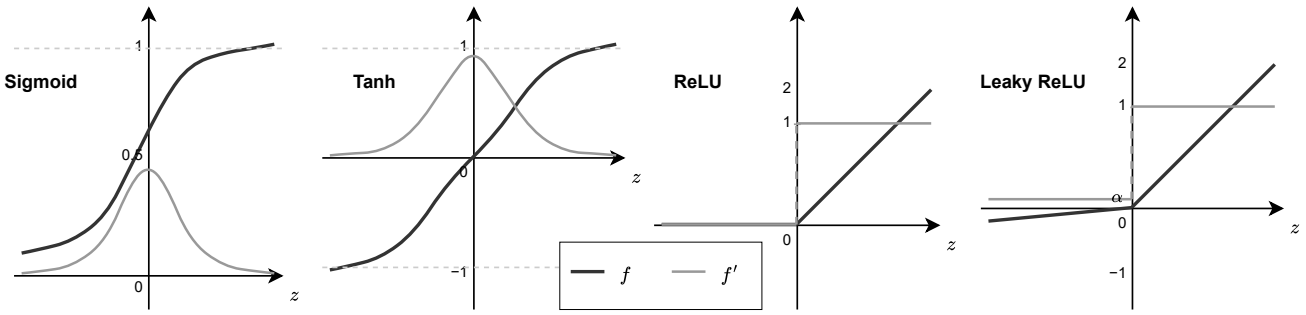


Figure 12.4.: Derivatives of activation functions.

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.

12.3. 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 $c \in \{1, \ldots, K\}$ is encoded as a binary vector $\mathbf{y} \in \{0, 1\}^K$ with exactly one element equal to 1:

$$y_j = \begin{cases} 1 & j = c \\ 0 & j \neq c \end{cases} \qquad j = 1, \ldots, K \tag{12.11}$$

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

$$\hat{s_i} = f_S(s_i) = \frac{\exp(s_i)}{\sum_{j=1}^{K} \exp(s_j)} \quad i = 1, \ldots, K \tag{12.12}$$

The corresponding loss (categorical cross entropy) is given by

$$\mathcal{L} = -\frac{1}{K} \sum_{j=1}^{K} \log(\hat{s}_i) \tag{12.13}$$

For example, $(1, 2, 8) \rightarrow (0.001, 0.002, 0.997)$.

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 $K = 2$.

12.4. 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):

$$\mathcal{L}_{\text{MSE}} = \frac{1}{M}\sum_{i=1}^{M}(y_i - \hat{y}_i)^2 \tag{12.14}$$

  Typically paired with a linear (identity) output activation. See Ch. 7 for derivation and properties.
- **Binary classification** – Binary Cross-Entropy (BCE):

$$\mathcal{L}_{\text{BCE}} = -\frac{1}{M}\sum_{i=1}^{M}\Big[y_i \log \hat{y}_i + (1 - y_i)\log(1 - \hat{y}_i)\Big] \tag{12.15}$$

  Paired with a sigmoid output activation. See Sec. 8.4 for derivation.
- **Multi-class classification** – Categorical Cross-Entropy (CCE):

$$\mathcal{L}_{\text{CCE}} = -\frac{1}{M}\sum_{i=1}^{M}\sum_{j=1}^{K} y_{ij} \log \hat{y}_{ij} \tag{12.16}$$

  Paired with a softmax output activation.

Table 12.2 summarizes the common task–loss–activation combinations.

Table 12.2.: Common task–loss–activation combinations.

Task	Loss	Output activation
Regression	MSE	Linear (identity)
Binary classification	BCE	Sigmoid
Multi-class classification	CCE	Softmax

12.5. Back-propagation

12.5.1. 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:

$$\frac{\partial \mathcal{L}(y, \hat{y})}{\partial w_{ij}^{[k]}} = \frac{\partial \mathcal{L}(y, \hat{y})}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial w_{ij}^{[k]}} \tag{12.17}$$

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.

12.5.2. 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:

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{2M} \sum_{i=1}^{M} (y_i - \hat{y}_i)^2 \tag{12.18}$$

Its derivative with respect to the prediction is

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \hat{y}_i} = -\frac{1}{M} (y_i - \hat{y}_i) \tag{12.19}$$

The output layer applies a sigmoid activation, $\hat{y}_i = \sigma(a_i^{[l]})$, whose derivative is

$$\frac{\partial \hat{y}_i}{\partial a_i^{[l]}} = \sigma(a_i^{[l]}) \Big(1 - \sigma(a_i^{[l]})\Big) \tag{12.20}$$

Combining via the chain rule:

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial a_i^{[l]}} = \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial a_i^{[l]}} \tag{12.21}$$

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

$$\frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial w_{ij}^{[l]}} = \frac{\partial \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \hat{y}_i} \frac{\partial \hat{y}_i}{\partial a_i^{[l]}} \frac{\partial a_i^{[l]}}{\partial w_{ij}^{[l]}} \tag{12.22}$$

This pattern extends to deeper networks: the chain rule propagates the gradient backward through each layer.

12.6. Learning

12.6.1. 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 [15].

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.

12.6.2. 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 12.3):

- the learning rate,
- and the stopping criterion.

Table 12.3.: 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 $\alpha_k = \alpha_0 e^{-\lambda k}$.

12.7. 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 [13]. 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

$$\begin{aligned} \mathbf{W}^{[l]} &\sim N(\mu = 0, \sigma^2 = \frac{1}{n^{[l-1]}}) \\ b^{[l]} &= 0 \end{aligned} \tag{12.23}$$

where $n^{[l-1]}$ is the number of neuron in layer $l-1$ (previous layer).

12.8. 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)

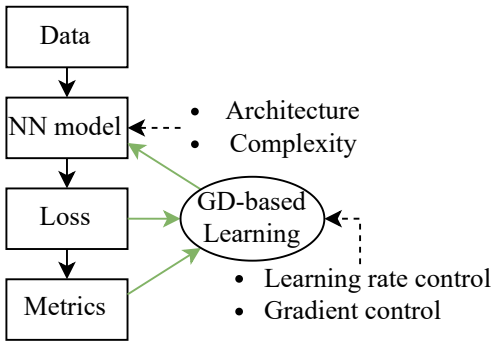


Figure 12.5.: Summary of VNN design choices.

Part II.

Linear Least Squares in Signals & Systems 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 [10]. 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.

13. 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

13.1. Signal Preliminaries

13.1.1. Cosine Signal

A general continuous-time cosine signal can be written as

$$\begin{aligned} x(t) &= A\cos(2\pi F_0 t + \theta), \\ &= A\cos(\Omega_0 t + \theta), \end{aligned} \tag{13.1}$$

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 $F_s = 1/T$,

$$\begin{aligned} x[n] &= x(nT) \\ &= A\cos(\omega_0 n + \theta) \quad n = 0, \ldots, L-1, \end{aligned} \tag{13.2}$$

where

$$\omega_0 = 2\pi F_0 T = 2\pi \frac{F_0}{F_s} \tag{13.3}$$

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.

Energy & power

The energy of the signal $x[n]$ is defined as

$$E_{\mathbf{x}} = \|\mathbf{x}\|^2 = \sum_{n=0}^{L-1} x^2[n], \tag{13.4}$$

where $\mathbf{x}$ is the vector of samples of the signal $x[n]$. The corresponding power is time-averaged energy,

$$P_{\mathbf{x}} = \frac{1}{L} E_{\mathbf{x}} = \frac{1}{L}\|\mathbf{x}\|^2. \tag{13.5}$$

13.1.2. Noise

We define noisy signal

$$y[n] = x[n] + \epsilon[n], \tag{13.6}$$

where $\epsilon(t)$ is additive noise.

The only assumption for the additive noise is that it is zero-mean,

$$\bar{\epsilon} = \frac{1}{L}\sum_{n=0}^{L-1} \epsilon[n] = 0. \tag{13.7}$$

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 noise (unbiased) variance is the power (13.5) of the zero-mean noise signal.

13.1.3. 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. 13.1.

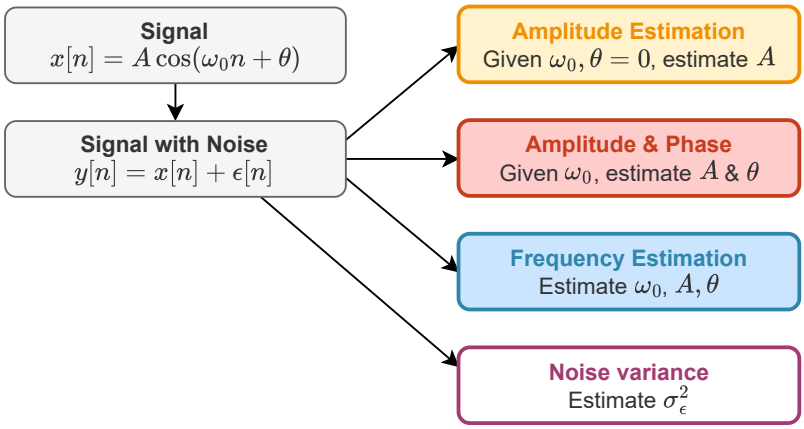


Figure 13.1.: Basic signal analysis goals.

Linear regression

The analysis is based on the LS minimization with model,

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w}, \tag{13.8}$$

where $\mathbf{w}$ are learned weights. SSE loss is

$$\mathcal{L}(\mathbf{w}) = \|\mathbf{y} - \hat{\mathbf{y}}\|^2 = \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 \tag{13.9}$$

with the loss minimum at

$$\hat{\mathbf{w}} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y} \tag{13.10}$$

Due to orthogonality,

$$\|\mathbf{y}\|^2 = \|\hat{\mathbf{y}}\|^2 + \|\mathbf{e}\|^2 \tag{13.11}$$

In the following, for each method, matrices $\mathbf{X}, \mathbf{y}, \mathbf{w}$ and there relation to a sinusoidal signal are to be defined.

13.2. Amplitude estimation

The goal is to find the *amplitude* of a sinusoidal signal in noise that best fits the model in a least squares (LS) sense.

Given a signal model with $\theta = 0$ and a known frequency ω_0,

$$x[n] = A\cos(\omega_0 n) \tag{13.12}$$

the goal is to estimate the amplitude A from

$$y[n] = x[n] + \epsilon[n] \quad n = 0, \ldots, L-1 \tag{13.13}$$

that best fits a provided model,

$$\hat{x}[n] = \hat{A}\cos(\omega_0 n) \tag{13.14}$$

13.2.1. LS Formulation

Technically, we are looking for the value of A that minimizes the squared error,

$$\mathcal{L} = \sum_n (y[n] - \hat{x}[n])^2. \tag{13.15}$$

The corresponding vector form of the SSE loss is

$$\begin{aligned}\mathcal{L}(A) &= \left\| \begin{bmatrix} y[0] \\ y[1] \\ y[2] \\ \vdots \\ y[L-1] \end{bmatrix} - \begin{bmatrix} 1 \\ \cos(\omega_0) \\ \cos(2\omega_0) \\ \vdots \\ \cos\big((L-1)\omega_0\big) \end{bmatrix} A \right\|^2 \\ &= \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2\end{aligned} \tag{13.16}$$

where the required weight is $\mathbf{w} = A$, $\mathbf{X}$ is formed by samples of the signal $\big\{\cos(\omega_0 n)\big\}_{n=0}^{L-1}$ and $\mathbf{y}$ is formed by samples of the signal $\big\{y[n]\big\}_{n=0}^{L-1}$, The resulting minimization is straightforward,

$$\begin{aligned}\hat{A} &= (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \\ &= \frac{\sum_n y[n]\cos(\omega_0 n)}{\sum_n \cos^2(\omega_0 n)}\end{aligned} \tag{13.17}$$

If we substitute the signal model into the resulting solution,

$$\begin{aligned}\hat{A} &= \frac{\sum_n (A\cos(\omega_0 n) + \epsilon[n])\cos(\omega_0 n)}{\sum_n \cos^2(\omega_0 n)} \\ &= A + \frac{\sum_n \epsilon[n]\cos(\omega_0)}{\sum_n \cos^2(\omega_0 n)}\end{aligned} \tag{13.18}$$

it produces a true value of A with some additive noise.

13.2.2. 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 (13.14), the residual error $\|\mathbf{e}\|^2$ is the estimation of the noise,

$$\|\mathbf{y}\|^2 = \|\hat{\mathbf{y}}\|^2 + \|\mathbf{e}\|^2 \Rightarrow \begin{cases} E_{\mathbf{y}} = E_{\hat{\mathbf{y}}} + E_{\mathbf{e}} \\ P_{\mathbf{y}} = P_{\hat{\mathbf{y}}} + P_{\mathbf{e}} \end{cases} \tag{13.19}$$

Moreover, due to zero-mean property of the noise, the estimated power and/or variance of the noise (13.5) is

$$\hat{P}_\epsilon = \hat{\sigma}_\epsilon^2 = \frac{1}{L}\|\mathbf{e}\|^2. \tag{13.20}$$

An interesting interpretation of this result is estimated signal to noise ratio (SNR), defined as

$$\widehat{SNR} = \frac{P_{\hat{\mathbf{y}}}}{P_{\mathbf{e}}} = \frac{\|\hat{\mathbf{y}}\|^2}{\|\mathbf{e}\|^2} \tag{13.21}$$

The following example (Fig. 13.2) 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 $\hat{\sigma}_\epsilon = 4.43$.

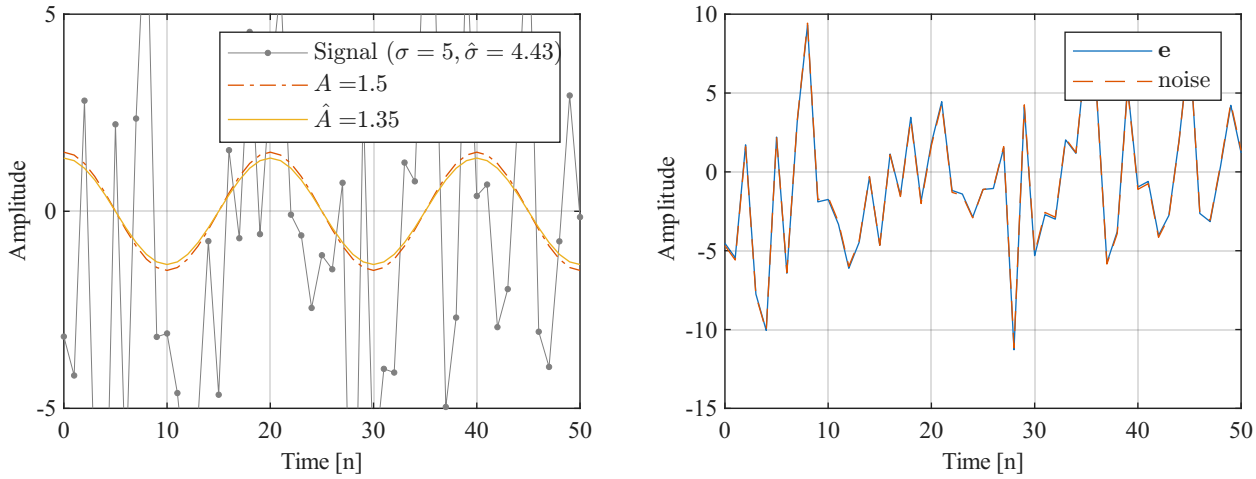
13.3. Amplitude and phase estimation

The goal is to find both the *amplitude* and the *phase* of a sinusoidal signal in noise that best fits the model in a least squares (LS) sense. The following analysis is provided for the more general model,

$$x[n] = A\cos(\omega_0 n + \theta) \quad n = 0, \ldots, L-1, \tag{13.22}$$

The goal is to estimate amplitude A and the phase θ that best fits a provided model,

$$\hat{x}[n] = \hat{A}\cos\left(\omega_0 n + \hat{\theta}\right) \tag{13.23}$$



(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 13.2.: Example of the cosine signal amplitude estimation.

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,

$$A\cos\left(\omega_0 n + \theta\right) = w_c \cos(\omega_0 n) + w_s \sin(\omega_0 n), \tag{13.24}$$

where

$$\begin{aligned} w_c &= A\cos(\theta) \\ w_s &= -A\sin(\theta) \end{aligned} \tag{13.25}$$

and

$$\begin{aligned} A &= \sqrt{w_c^2 + w_s^2} \\ \theta &= -\arctan\left(\frac{w_c}{w_s}\right) \end{aligned} \tag{13.26}$$

This transforms the problem into a two-parameter linear LS problem in terms of w_c and w_s [1]. The resulting LS formulation involves a two-valued vector of linear coefficients, $\mathbf{w} = \begin{bmatrix} w_c & w_s \end{bmatrix}^T$, $\mathbf{y}$ is formed by samples of the signal $\left\{y[n]\right\}_{n=0}^{L-1}$, and the matrix $\mathbf{X}$ of dimensions $L \times 2$ that is given by

$$\mathbf{X} = \begin{bmatrix} 1 & 0 \\ \cos(\omega_0) & \sin(\omega_0) \\ \cos(2\omega_0) & \sin(2\omega_0) \\ \vdots & \vdots \\ \cos\big((L-1)\omega_0\big) & \sin\big((L-1)\omega_0\big) \end{bmatrix}. \tag{13.27}$$

Once $\hat{\mathbf{w}}$ has been found, the amplitude and phase can be recovered from (13.26).

SNR and noise variance interpretations are similar to in the previous model in Eqs. (13.21) and (13.20).

The numerical example is presented in Fig. 13.3. The configuration is similar to the previous figure, expect the lower noise variance, $\sigma = 1.5$. Nevertheless, there is a decrease in performance, since two parameters are estimated simultaneously.

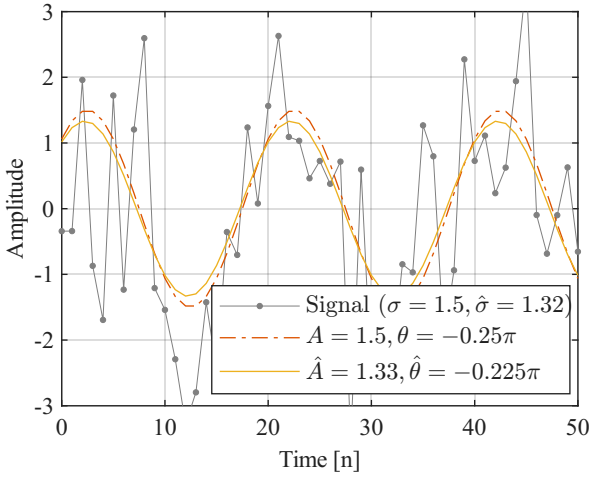


Figure 13.3.: Example of the cosine signal amplitude and phase estimation. Note the lower estimation accuracy compared to Fig. 13.2, since two parameters are estimated simultaneously.

13.3.1. Advanced notes (*)

- 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, 14].
- 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)]

$$\mathrm{Var}\left[\hat{w}_{c,s}\right] \gtrsim \frac{2\sigma^2}{L} \tag{13.28}$$

  This bound is the tightest for the AWGN case and is less accurate for other noise distributions.
- While the derivation of CRLB is non-trivial, the approximated numerical estimation variance can be easily evaluated by Monte-Carlo simulations for any set of parameters and any distribution of interest.

13.4. Frequency estimation

The goal is to deal with the unknown A, θ, ω_0.

If the frequency ω_0 is also unknown, it can be 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 $\mathbf{X}$ may be parameterized as a frequency-dependent one, $\mathbf{X}(\omega)$. Here, the estimated signal is frequency-dependent

$$\hat{\mathbf{y}}(\omega) = \mathbf{X}(\omega)\mathbf{w}(\omega), \tag{13.29}$$

where $\mathbf{w}(\omega)$ are the estimated parameters w_c and w_s at that frequency. The corresponding frequency-dependent residual error is given by (see also (13.19))

$$\mathbf{e}(\omega) = \mathbf{y} - \hat{\mathbf{y}}(\omega). \tag{13.30}$$

Since the error $\mathbf{e}(\omega)$ is orthogonal to $\hat{\mathbf{y}}$,

$$\|\mathbf{y}\|^2 = \left\|\hat{\mathbf{y}}(\omega)\right\|^2 + \left\|\mathbf{e}(\omega)\right\|^2. \tag{13.31}$$

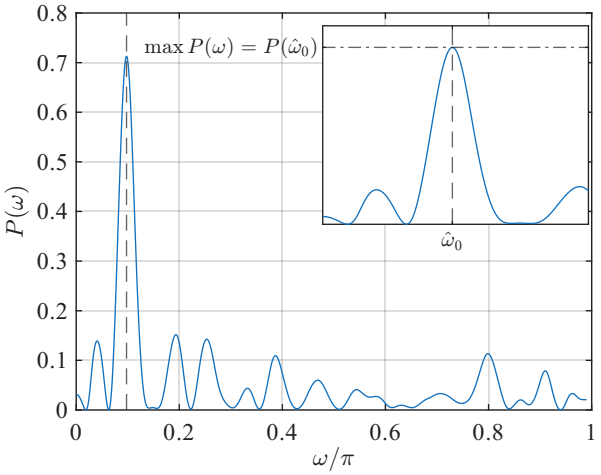
To find the frequency that best represents the data, we seek the one that maximizes the energy of the reconstructed signal (or equivalently minimizes the residual error), as mentioned above

$$\hat{\omega}_0 = \arg\min_{\omega} \left\|\mathbf{e}(\omega)\right\|^2 = \arg\max_{\omega} \left\|\hat{\mathbf{y}}(\omega)\right\|^2. \tag{13.32}$$

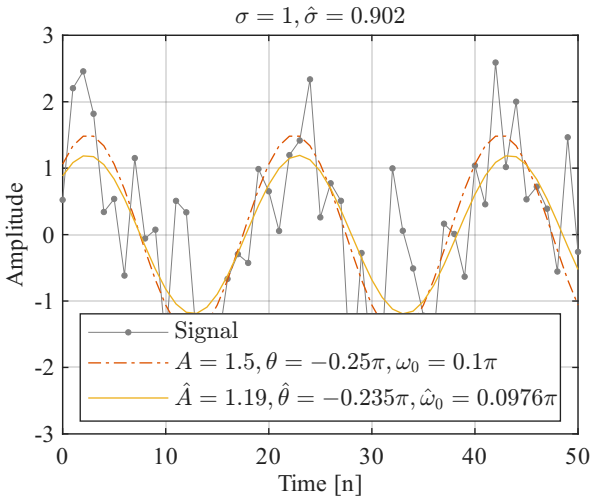
Note, this optimization problem can be challenging because the objective function may exhibit multiple local maxima/minima. Therefore, an appropriate numerical global optimization method is required.

Once $\hat{\omega}_0$ has been found, the amplitude and phase are estimated using the corresponding linear LS solution $\mathbf{w}(\omega_0)$ (Sec. 13.3). This solution also results in SNR and noise variance estimations, as in Eqs. (13.21) and (13.20).

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. 13.4. First, periodogram peak is found (Fig. 13.4a). Than, the subsequent amplitude/phase estimation result is presented (Fig. 13.4b).



(a) The periodogram $P(\omega)$ with a prominent peak at $\omega_0 \approx 0.1\pi$.



(b) Reconstracted signal.

Figure 13.4.: 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. 13.3, since three parameters are estimated.

13.4.1. Advanced notes (*)

Interpretation in Terms of the Periodogram

The function

$$P(\omega) = \frac{1}{L}\left\|\hat{\mathbf{y}}(\omega)\right\|^2 \tag{13.33}$$

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,

$$\hat{\omega}_0 = \arg\min_{\omega} P(\omega) \tag{13.34}$$

The complexity of frequency estimation (without amplitude/phase step) can significantly reduced by noting that

$$\arg\max_{\omega} \left\|\hat{\mathbf{y}}(\omega)\right\|^2 \propto \arg\max_{\omega} \left|\sum_{n=0}^{L-1} [n]\exp(-j\omega n)\right|^2 \tag{13.35}$$

However, this definition involves frequencies in the range $0 \leq \omega < 2\pi$; these has challenging physical interpretation.

Theoretical performance bounds

Under AWGN assumption, theoretical SNR is given by

$$SNR = \frac{A^2}{2\sigma^2} \tag{13.36}$$

and the corresponding CRLB on the estimation variances are [14]

$$\mathrm{Var}\left[\hat{A}\right] \geqslant \frac{2\sigma^2}{L} \quad [V^2] \tag{13.37}$$

$$\mathrm{Var}[\hat{\omega}_0] \geqslant \frac{12}{SNR \times L(L^2-1)} \approx \frac{12}{SNR \times L^3} \quad \left[\left(\frac{rad}{sample}\right)^2\right] \tag{13.38}$$

$$\mathrm{Var}\left[\hat{\theta}\right] \geqslant \frac{2(2L-1)}{SNR \times L(L+1)} \approx \frac{4}{SNR \times L} \quad [rad^2] \tag{13.39}$$

For analog frequency $F_0 = \dfrac{\omega_0}{2\pi}F_s$,

$$\mathrm{Var}[F_0] = \mathrm{Var}[\omega_0]\left(\frac{F_s}{2\pi}\right)^2 \quad [Hz^2] \tag{13.40}$$

13.5. Harmonic Signal Analysis

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.

$$y[n] = A_0 + \sum_{m=1}^{M} A_m \cos(m\omega_0 n + \theta_m) = \sum_{m=\mathbf{0}}^{M} A_m \cos(m\omega_0 n + \theta_m), \tag{13.41}$$

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, $M\omega_0 < \pi$.

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,

$$THD = \frac{\sqrt{\sum_{m=2}^{M} A_m^2}}{A_1}. \tag{13.42}$$

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.

Amplitudes and phases estimation

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. 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. 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,

$$P_h(\omega) = \frac{1}{L} \sum_{m=1}^{M} \left\| \hat{\mathbf{y}}(m\omega) \right\|^2, \tag{13.43}$$

the harmonic amplitudes and phases can be estimated via LS at this frequency [5].

The example is the sampled current of a switch-mode power supply in a 50Hz network sampled at a 50kHz frequency [4]. Figure 13.5a shows a reconstruction of the signal with $M = 250$ harmonics.

The estimated amplitudes $\hat{A}_m$ are shown (Fig. 13.5b) 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 13.5c shows estimated SNR (top) and the noise standard deviation (bottom) vary as the number of harmonics M in the model increases.

13.5.1. Advanced notes (*)

The presented estimator for both known and unknown ω_0 is an effective ML estimator in the case of AWGN, with known analytical CRLB [5].

13.6. Discrete Fourier Transform (DFT)

13.6.1. 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,

$$y[n] = \sum_{k=0}^{N-1} A_k \cos\left(k\frac{2\pi}{N}n + \theta_k\right), \quad n = 0, \ldots, L-1 \tag{13.44}$$

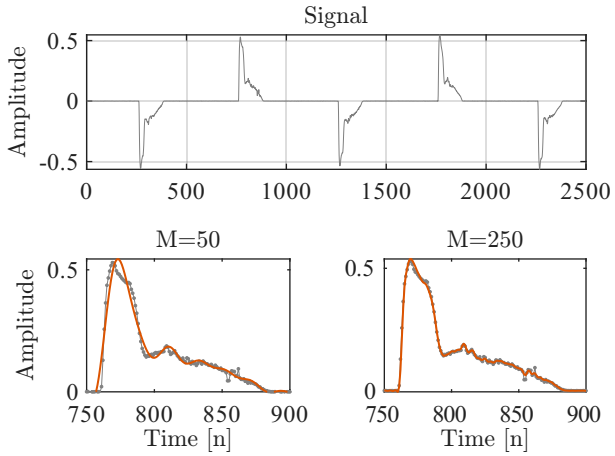
- For $N \geq L$, the DFT allows for *perfect* reconstruction of the signal using its harmonic representation parameters, $\{A_k, \theta_k\}_{k=0}^{N-1}$.

The corresponding LS formulation follows the harmonic model above for known ω_0.

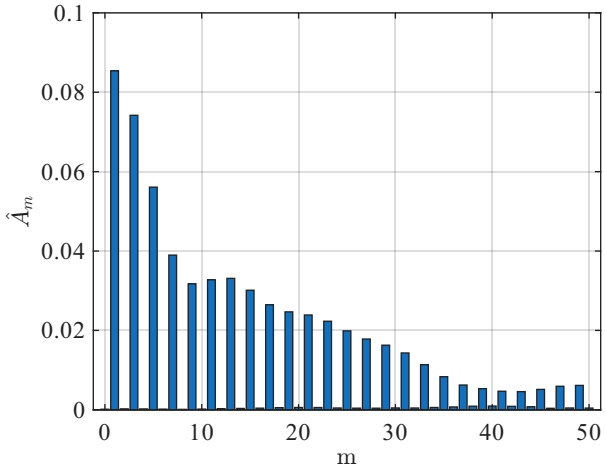
13.6.2. Properties

Zero-padding

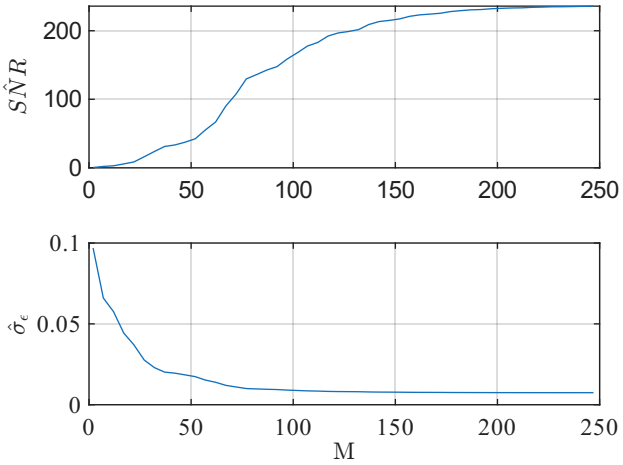
The basic requirement is $N = L$ harmonic signals. Sometimes, denser spectral representation with $N > L$ is required. In this case, the corresponding time-domain signal can be interpreted as the original signal $y[n]$ with $N - L$ zeros padded at the end of this signal. This case it termed *zero padding*.



(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 13.5.: Example of a harmonic signal analysis.

Symmetry

It is also worth noting the symmetry in the DFT,

$$\begin{aligned}\cos\big((N-k)\Delta\omega\big) &= \cos(k\Delta\omega)\\ \sin\big((N-k)\Delta\omega\big) &= -\sin(k\Delta\omega),\end{aligned} \tag{13.45}$$

resulting in redundant information for frequencies $k\omega_0$ above and below π,

$$\begin{aligned}A_k &= A_{N-k}\\ \theta_k &= -\theta_{N-k}\end{aligned}. \tag{13.46}$$

As a result, only frequencies $k\omega_0 \leq \pi$ need to be considered uniquely.

13.6.3. Power Spectral Density

The power of a signal of the form

$$x_k[n] = A_k \cos\left(k\frac{2\pi}{N}n + \theta_k\right) \tag{13.47}$$

is

$$S_{\mathbf{y}}[k] = P_{\mathbf{y}_k} = \frac{1}{L}\|\mathbf{y}_k\|^2 = \frac{A_k^2}{2}. \tag{13.48}$$

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. (13.33)), 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

$$\sum_{k=0}^{N-1} A_k^2 = \frac{1}{L}\|\mathbf{y}\|^2. \tag{13.49}$$

13.6.4. Advanced notes (*)

Fast Fourier transform (FFT) & Goertzel algorithm

The fast Fourier transform (FFT) algorithm efficiently computes $Y[k] = \text{DFT}\big\{y[n]\big\}$, providing $A_k = |Y[k]|/N$ and $\theta_k = \angle(Y[k])$ with significantly lower memory requirements and complexity than direct calculation in Eq. (13.59).

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. (13.32).

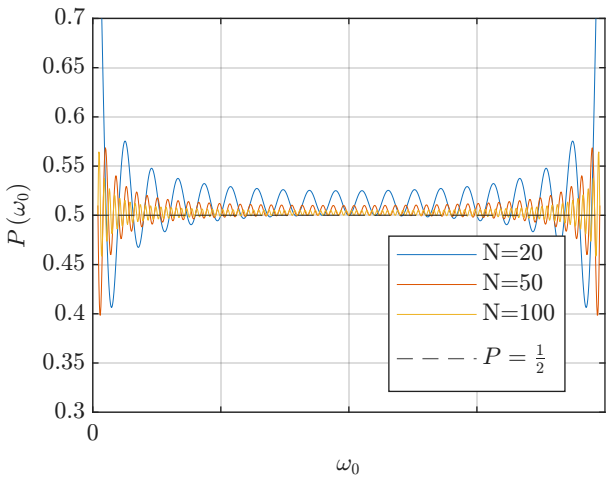


Figure 13.6.: Power of the sinusoidal signal as a function of ω_0 for different values of L.

Parseval's Theorem Using FFT notation,

$$A_k^2 = \left|Y[k]\right|^2/L^2 \Rightarrow \sum_{k=0}^{L-1} \left|Y[k]\right|^2 = L\|\mathbf{y}\|^2. \tag{13.50}$$

Sinusoidal signal power

For a more general case of an arbitrary ω value, the signal of the form

$$y[n] = A\cos(\omega_0 n) \tag{13.51}$$

has the ω_0-dependent power,

$$\begin{aligned} P_{\mathbf{y}} &= \frac{A^2}{L}\sum_{n=0}^{L-1}\cos^2(\omega_0 n) \\ &= \frac{A^2}{4L}\left(1 + 2L - \frac{\sin(\omega_0 - 2L\omega_0)}{\sin(\omega_0)}\right), \end{aligned} \tag{13.52}$$

that results from the time-limited origin of the signal $y[n]$. For the infinite number of samples, the resulting power converges to a continuous-time power expression,

$$\lim_{L\to\infty} P_{\mathbf{y}} \to \frac{A^2}{2} \tag{13.53}$$

$$\sum_n \cos^2(\omega_0 n) \approx \frac{L}{2} \qquad \omega_0 \neq 0, \pi \tag{13.54}$$

The illustration of the (13.52) is presented in Fig. 13.6.

Single frequency analysis

Consider a signal $y_k[n]$ with a discrete frequency $\omega_k = \frac{2\pi}{N}k$ is given,

$$y_n[k] = A\cos(\omega_k n + \theta) = A\cos\left(\frac{2\pi}{N}kn + \theta\right) \tag{13.55}$$

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

$$\mathbf{X}_k^T\mathbf{X}_k = \begin{bmatrix} \mathbf{x}_c^T\mathbf{x}_c & \mathbf{x}_s^T\mathbf{x}_c \\ \mathbf{x}_s^T\mathbf{x}_c & \mathbf{x}_s^T\mathbf{x}_s \end{bmatrix} \tag{13.56}$$

we find that the sine and cosine columns form an orthogonal basis for this single frequency, with (irrespective to k)

$$\begin{aligned} \mathbf{x}_c^T\mathbf{x}_c &= \sum_{n=0}^{N-1}\cos^2\left(2\pi\frac{k}{N}n\right) = \frac{N}{2} \\ \mathbf{x}_c^T\mathbf{x}_s &= \sum_{n=0}^{N-1}\cos\left(2\pi\frac{k}{N}n\right)\sin\left(2\pi\frac{k}{N}n\right) = 0 \\ &= \mathbf{x}_s^T\mathbf{x}_c \\ \mathbf{x}_s^T\mathbf{x}_s &= \sum_{n=0}^{N-1}\sin^2\left(2\pi\frac{k}{N}n\right) = \frac{N}{2} \end{aligned} \tag{13.57}$$

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 [14].

Moreover, since $\left(\mathbf{X}_k^T\mathbf{X}_k\right)^{-1}$ takes a particularly simple diagonal form,

$$\left(\mathbf{X}_k^T\mathbf{X}_k\right)^{-1} = \begin{bmatrix} \frac{2}{N} & 0 \\ 0 & \frac{2}{N} \end{bmatrix} \tag{13.58}$$

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

$$w_{c,k} = \frac{2}{N}\sum_{n=0}^{L-1} y[n]\cos\left(2\pi\frac{k}{N}n\right), \tag{13.59a}$$

$$w_{s,k} = \frac{2}{N}\sum_{n=0}^{L-1} y[n]\sin\left(2\pi\frac{k}{N}n\right). \tag{13.59b}$$

General orthogonality

The orthogonality of $\mathbf{X}^T\mathbf{X}$ in more general form results from

$$\sum_{n=0}^{N-1} \cos\left(2\pi\frac{j}{N}n\right)\cos\left(2\pi\frac{k}{N}n\right) = \frac{N}{2}\delta\left[j-k\right] \tag{13.60}$$

$$\sum_{n=0}^{N-1} \cos\left(2\pi\frac{k}{N}n\right)\sin\left(2\pi\frac{k}{N}n\right) = 0 \quad \forall j,k \tag{13.61}$$

$$\sum_{n=0}^{N-1} \sin\left(2\pi\frac{j}{N}n\right)\sin\left(2\pi\frac{k}{N}n\right) = \frac{N}{2}\delta\left[j-k\right], \tag{13.62}$$

Spectral Spreading and Leakage

In an idealized setting, a pure cosine signal has a perfectly defined frequency representation. For instance, consider the discrete-time signal,

$$x[n] = A\cos\left(k_0\frac{2\pi}{L}n\right),\ k_0 \in \{1,\cdots,L-1\} \tag{13.63}$$

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

$$A_k = \begin{cases} \frac{A}{2} & k = k_0, N-k_0 \\ 0 & \text{otherwise} \end{cases}. \tag{13.64}$$

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

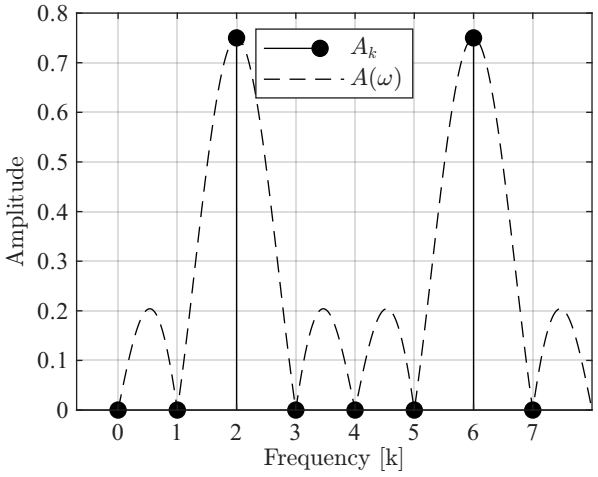


Figure 13.7.: Illustration of a single-frequency cosine signal's spectrum under ideal assumptions (discrete, integer-multiple frequencies) versus practical conditions (denser frequency grid or non-integer frequencies). Note how the ideal single peak broadens and additional low-level components appear, 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. (13.29)), the resulting spectral distribution can differ substantially from the discrete, single-peak ideal (Fig. 13.7). This difference arises because, in general, $\mathbf{X}(\omega)^T\mathbf{X}(\omega)$ is not orthogonal as in Eq. (13.57). 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."

13.6.5. 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.

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

$$y_\ell[n] = w[n] \cdot y[\ell H + n], \quad n = 0, \ldots, N_w - 1, \tag{13.65}$$

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,

$$A_k[\ell], \quad \theta_k[\ell], \quad k = 0, \ldots, N_w - 1, \tag{13.66}$$

providing the amplitude and phase of each frequency component at each time frame ℓ.

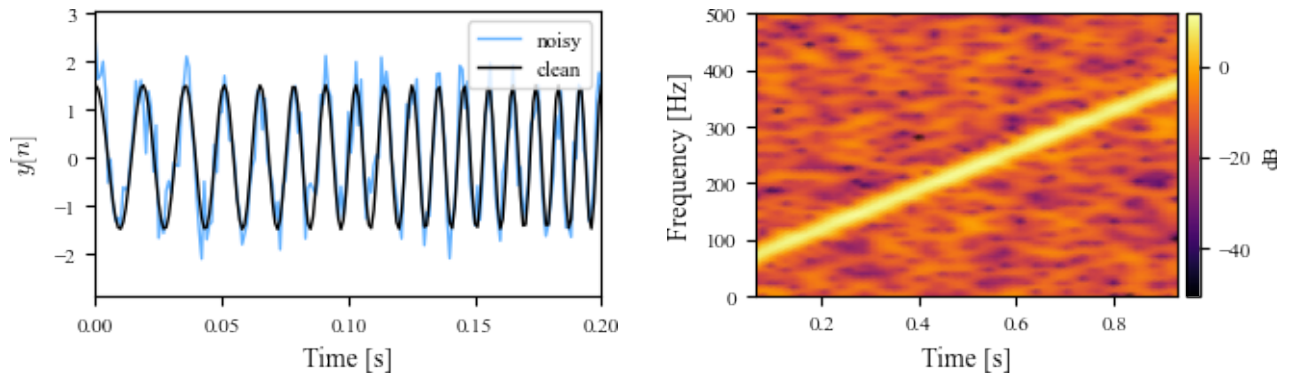
The following example uses a chirp signal whose instantaneous frequency sweeps linearly from 50 Hz to 400 Hz over 1 s, sampled at $F_s = 1000$ Hz, with additive Gaussian noise ($\sigma = 0.5$). The time-domain signal is shown in Fig. 13.8a and its spectrogram ($N_w = 128$, $H = 16$, Hann window) in Fig. 13.8b.

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.

This time-frequency trade-off is inherent to STFT and cannot be resolved by the choice of window function alone. Figure 13.9 illustrates this trade-off: a short window ($N_w = 32$) provides sharp



(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.

Figure 13.8.: Example of STFT analysis of a chirp signal.

time localization but blurred frequency content, while a long window ($N_w = 256$) yields precise frequency resolution at the cost of temporal smearing.

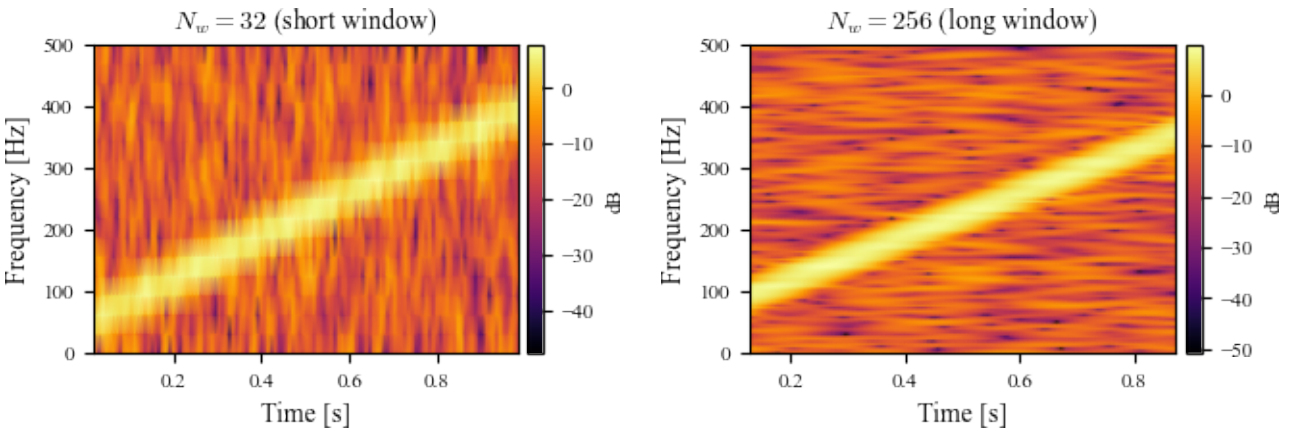


Figure 13.9.: Time-frequency trade-off. Left: short window ($N_w = 32$) — good time resolution, poor frequency resolution. Right: long window ($N_w = 256$) — good frequency resolution, poor time resolution.

Spectrogram

The power spectral density of each segment defines a time-frequency representation known as the spectrogram,

$$S[\ell, k] = \frac{A_k^2[\ell]}{2}, \tag{13.67}$$

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.

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.

13.7. Summary

The summary of the presented approach is shown in Table 13.1. 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.

Table 13.1.: Comparison and summary of different signal estimation methods.

Task	Parameters	Matrix $\mathbf{X}$	SNR
Amplitude only	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: 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.

Although the estimators presented above have been extensively analyzed for the specific case of additive white Gaussian noise (AWGN) in the statistical signal processing literature [14, 12], conducting such an analysis requires a significantly more extensive mathematical framework. Furthermore, it is worth noting that any bias and variance in these estimators can be readily approximated via Monte Carlo simulations under various parameter settings and noise distributions.

Answer of exercise 3

$$\mathcal{L}(A) = \sum_n \big(x[n] - A\big)^2$$

$$\frac{\partial}{\partial A}\mathcal{L}(A) = 2\sum_n \big(x[n] - A\big)(-1) = 0$$

$$LA = \sum_{n=0}^{L-1} x[n]$$

$$A = \bar{x}[n]$$

$$MSE = \frac{1}{L}\sum_{n=0}^{L-1} \big(x[n] - \bar{x}[n]\big)^2 = \mathrm{Var}\big[x[n]\big]$$

Table 13.2.: 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	$\mathbf{X} = \begin{bmatrix} 1 & \cdots & 1 & 0 & \cdots \\ 1 & \cdots & \cos\left(\frac{2\pi}{N}k\right) & \sin\left(\frac{2\pi}{N}k\right) & \cdots \\ 1 & \cdots & \cos\left(2\frac{2\pi}{N}k\right) & \sin\left(2\frac{2\pi}{N}k\right) & \cdots \\ \vdots & & \vdots & \vdots & \ddots \\ 1 & \cdots & \sin\left((L-1)\frac{2\pi}{N}k\right) & \sin\left((L-1)\frac{2\pi}{N}k\right) & \cdots \end{bmatrix}$

Answer of exercise 4 The solution is based on

$$\mathbf{X} = \begin{bmatrix} \mathbf{1}_L & \mathbf{n} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ \vdots & \vdots \\ 1 & L-1 \end{bmatrix} \tag{13.70}$$

The closed-form analysis is:

$$\begin{aligned}
\mathbf{X}^T\mathbf{X} &= \begin{bmatrix} \mathbf{1}^T\mathbf{1} & \mathbf{1}^T\mathbf{n} \\ \mathbf{n}^T\mathbf{1} & \mathbf{n}^T\mathbf{n} \end{bmatrix} \\
\mathbf{1}^T\mathbf{1} &= L \\
\mathbf{1}^T\mathbf{n} &= 0 + 1 + \cdots + L - 1 = \frac{L}{2}(L-1) \\
&= \mathbf{n}^T\mathbf{1} \\
\mathbf{n}^T\mathbf{n} &= 0 + 1^2 + 2^2 + (L-1)^2 \\
&= \frac{1}{6}(L-1)L(2L-1) \\
\left(\mathbf{X}^T\mathbf{X}\right)^{-1} &= \begin{bmatrix} \frac{6}{L+1} - \frac{2}{L} & -\frac{6}{L^2+L} \\ -\frac{6}{L^2+L} & \frac{12}{L^3-L} \end{bmatrix} \\
\mathbf{X}^T\mathbf{y} &= \begin{bmatrix} \mathbf{1}^T\mathbf{y} \\ \mathbf{n}^T\mathbf{y} \end{bmatrix} \\
&= \begin{bmatrix} \sum_n y[n] \\ \sum_n ny[y] \end{bmatrix} \\
\begin{bmatrix} \hat{A} \\ \hat{B} \end{bmatrix} &= \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}
\end{aligned}$$

14. 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

14.1. Auto-Correlation Function

Goal: Evaluate correlation between a signal and its time-shifted replicas to derive meaningful insights.

14.1.1. Linear Prediction & 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

$$x[n] = h_1 x[n-1] + \epsilon[n]. \tag{14.1}$$

and

$$\hat{x}[n] = h_1 x[n-1] \tag{14.2}$$

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

$$\mathcal{L}(h_1) = \sum_n \big(x[n] - h_1 x[n-1])\big)^2 \tag{14.3}$$

is used. Setting the derivative of $\mathcal{L}(h_1)$ with respect to h_1 to zero leads to

$$\frac{d\mathcal{L}(a)}{da} = 2\sum_n (x[n] - h_1 x[n-1])(-x[n-1]) = 0 \tag{14.4}$$

Finally, the LS solution for h_1 is

$$h_1 = \frac{\sum_n x[n]x[n-1]}{\sum_n x^2[n-1]}. \tag{14.5}$$

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

$$\hat{\mathbf{y}} = \begin{bmatrix} x[1] \\ x[2] \\ \vdots \\ x[L-2] \\ x[L-1] \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x[0] \\ x[1] \\ \vdots \\ x[L-3] \\ x[L-2] \end{bmatrix}, \tag{14.6}$$

such that

$$\hat{\mathbf{y}} = h_1 \mathbf{x}. \tag{14.7}$$

The correspoding MMSE loss is

$$\mathcal{L}(h_1) = \|\mathbf{y} - h_1 \mathbf{x}\|^2 \tag{14.8}$$

and its minimum is given by normal equation, (Eq. (3.5)),

$$h_1 = \left(\mathbf{x}^T \mathbf{x}\right)^{-1} \mathbf{x}^T \mathbf{y}. \tag{14.9}$$

The substitution results in the solution identical to Eq. (14.5).

14.1.2. Auto-correlation function (ACF)

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,

$$\hat{x}[n] = h_k x[n-k]. \tag{14.10}$$

The corresponding solution is very similar to the solution for $k = 1$,

$$h_k = \frac{\sum_n x[n]x[n-k]}{\sum_n x^2[n-k]}. \tag{14.11}$$

The nominator of Eq. (14.11) is termed (raw, or unscaled) auto-correlation function (ACF) at lag k,

$$R_{\mathbf{xx}}[k] = \sum_n x[n]x[n-k], \quad k = 0, \ldots, L-1 \tag{14.12}$$

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

$$E_{\mathbf{x}} = \sum_{n=0}^{L-1} x^2[n] = R_{\mathbf{xx}}[0] \tag{14.13}$$

Biased auto-correlation Another useful definition is averaged sum of $y[n]y[n-k]$,

$$\begin{aligned} R_{\mathbf{xx},biased}[k] &= \frac{1}{L}\sum_n x[n]x[n-k], \quad k = 0, \ldots, L-1 \\ &= \frac{1}{L} R_{\mathbf{xx}}[k] \end{aligned} \tag{14.14}$$

is termed biased ACF. The corresponding average power is given by

$$P_{\mathbf{x}} = \frac{1}{L}\sum_n x^2[n] = R_{\mathbf{xx},biased}[0] \tag{14.15}$$

Normalized auto-correlation Another version is normalized ACF of the form

$$R_{\mathbf{xx},norm}[k] = \frac{R_{\mathbf{xx}}[k]}{R_{\mathbf{xx}}[0]} \lesssim h_k. \tag{14.16}$$

Note, the difference between denominator above and the one in Eq. (14.11) is

$$\sum_n x^2[n] - \sum_n x^2[n-k] = \sum_{n<k} x^2[n] = x^2[0] + \cdots x^2[k-1] \tag{14.17}$$

Nevertheless, h_k expression in Eq. (14.11) and the latter expression are approximately equal for a sufficiently high L. Moreover, the denominator is k-independent.

Unbiased auto-correlation Note, that the ACF includes summation only of $n-k$ terms. Another useful normalization is

$$\begin{aligned} R_{\mathbf{xx},unbiased}[k] &= \frac{1}{L-k}\sum_n x[n]x[n-k] \\ &= \frac{1}{L-k}R_{\mathbf{xx}}[k] \end{aligned} \tag{14.18}$$

This time it is assumed that

$$\begin{aligned} \frac{1}{L}\sum_n x^2[n] &\approx \frac{1}{L-k}\sum_n x^2[n-k] \\ \frac{x^2[0]+x^2[1]+\cdots x^2[L-1]}{L} &\approx \frac{x^2[k]+\cdots x^2[L-1]}{L-k} \end{aligned} \tag{14.19}$$

and the resulting expression

$$\begin{aligned} \frac{L}{L-k}\frac{R_{\mathbf{xx}}[k]}{R_{\mathbf{xx}}[0]} &= \frac{L}{L-k}R_{\mathbf{xx},norm}[k] \\ &= \frac{R_{\mathbf{xx},unbiased}[k]}{R_{\mathbf{xx},unbiased}[0]} \approx h_k \end{aligned} \tag{14.20}$$

is assumed to be closer approximation to h_k than the normalized auto-correlation, for a relatively small values of k.

14.1.3. ACF Properties

The signal energy (14.13) is the highest value of ACF,

$$R_{\mathbf{xx}}[0] > R_{\mathbf{xx}}[k]. \tag{14.21}$$

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),

$$R_{\mathbf{xx}}[k] = R_{\mathbf{xx}}[-k] \tag{14.22}$$

Correlation Coefficient Interpretation

In matrix form (by Eq. (3.22)), the loss is given by

$$\mathcal{L}_{min}(h_k) = \mathbf{e}^T\mathbf{e} = \mathbf{y}^T\mathbf{y} - h_k\mathbf{y}^T\mathbf{x} \tag{14.23}$$

The resulting loss is given by (following Eq. (3.22) and the discussion above for $L \to \infty$)

$$\begin{aligned}
\mathcal{L}_{min}(h_k) &= \sum_{n=0}^{L-1} x^2[n] - h_k \sum_{n=0}^{L-1} x[n]x[n-k] \\
&= R_{\mathbf{xx}}[0] - h_k R_{\mathbf{xx}}[k] \\
&\lesssim R_{\mathbf{xx}}[0] \left(1 - \left(\frac{R_{\mathbf{xx}}[k]}{R_{\mathbf{xx}}[0]}\right)^2\right) \\
&= R_{\mathbf{xx}}[0] \left(1 - R^2_{\mathbf{xx},norm}[k]\right) \\
&\approx R_{\mathbf{xx}}[0] \left(1 - \rho^2_{\mathbf{xx}}[k]\right)
\end{aligned} \tag{14.24}$$

The value of $\rho_{\mathbf{xx}}[k]$ is termed *correlation coefficient* between $x[n]$ and $x[n-k]$,

$$\rho_{\mathbf{xx}}[k] \approx \frac{L}{L-k} R_{\mathbf{xx},norm}[k] \approx R_{\mathbf{xx},norm}[k]. \tag{14.25}$$

It is a measure of a linear dependence. It is bounded by

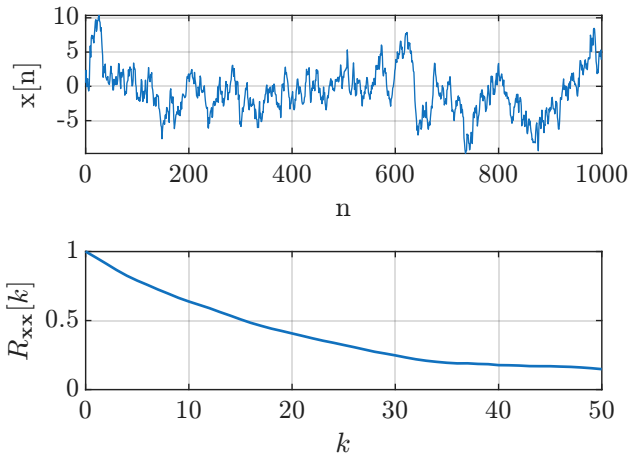
$$\left|\rho_{\mathbf{xx}}[k]\right| \leq 1 \tag{14.26}$$

- 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 $\hat{x}[n] = 0$.

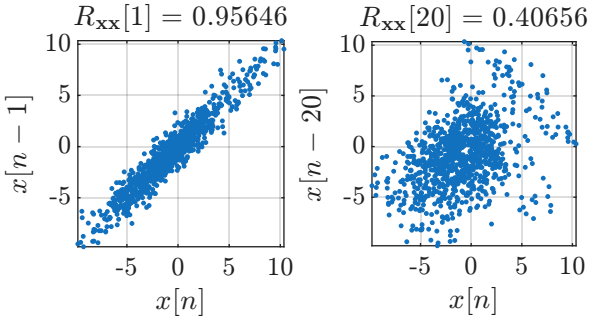
This can be summarized as (see also (14.13) for interpretation)

$$0 \leq \mathcal{L}_{min}(h_k) \leq R_{\mathbf{xx}}[0] \tag{14.27}$$

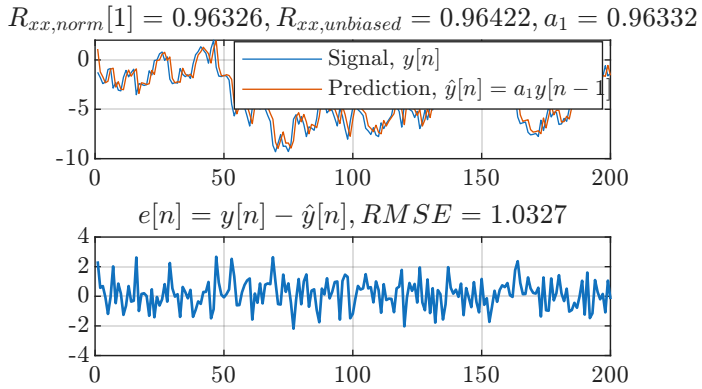
The illustration of the correlation coefficient principles is presented in Fig. 14.1.



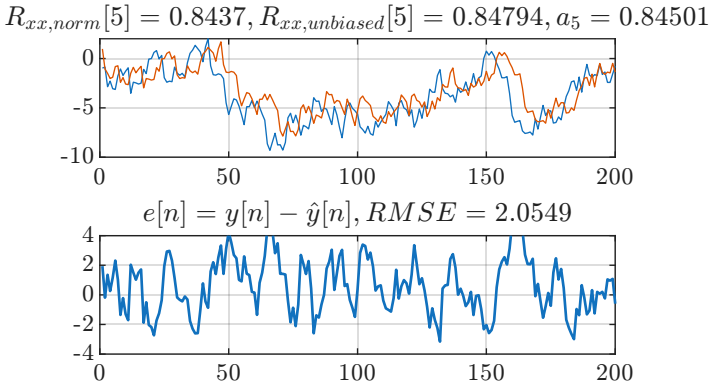
(a) ACF



(b) $x[n]$ vs. $x[n-k]$



(c) Prediction for $k = 1$



(d) Prediction for $k = 5$

Figure 14.1.: Illustration of the linear dependence between $x[n]$ and $x[n-k]$.

MSE and RMSE The corresponding MSE and RMSE metrics are given by

$$MSE(h_k) = \frac{1}{L}\mathcal{L}_{min}(h_k) \tag{14.28a}$$

$$RMSE(h_k) = \sqrt{\frac{1}{L}\mathcal{L}_{min}(h_k)} \tag{14.28b}$$

Correlation time The correlation time, k_c is the lag where $\rho_{\mathbf{xx}}[k_c]$ falls below a threshold,

$$\rho_{\mathbf{xx}}[k_c] = 0.5 \text{ or } 0.1 \text{ or } \exp(-1) \tag{14.29}$$

The predictability is assumed negligible for $k > k_c$,

$$\rho_{\mathbf{xx}}[k > k_c] \approx 0 \tag{14.30}$$

The decision threshold depends on the field of application. For example, one of the most applicable ACF for natural processes is

$$R_{\mathbf{xx},norm}[k] = \exp\left(-\frac{k}{k_c}\right) \tag{14.31}$$

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.

14.1.4. Confidence Interval

Another way to quantify the "importance" of the coefficients is to use confidence bound.

General idea The ACF estimate at a given lag is essentially a sample statistic derived from sums of products of random variables (the data values at different time points). Under typical assumptions of stationarity and weak dependence, these sample averages of random variables invoke the Central Limit Theorem (CLT). The CLT states that when you sum (or average) a sufficiently large number of independent or weakly dependent random variables with finite variance, the resulting distribution approaches a normal distribution, regardless of the variables' original distribution.

In other words, the estimation of the ACF involves averaging products like $x[n]x[n-k]$ across many time indices n. Provided the underlying process is stationary and not too strongly dependent, these averaged terms behave like sums of numerous random contributions. As the sample size L grows large, the distribution of the ACF estimate at each lag approaches normality by virtue of the CLT. This is why the ACF at a given lag can be approximated as a normally distributed random variable in large-sample scenarios.

Derivation Since the estimated ACF at a given lag k can be approximated as a normally distributed random variable with zero mean and variance approximately $\frac{1}{L}$ for large L, the 95% confidence bound is given by

$$R_{\mathbf{xx},norm}[k] \pm \frac{1.96}{\sqrt{L}}. \tag{14.32}$$

The value 1.96 comes from the properties of the standard normal distribution. In a standard normal distribution (mean 0, variance 1), approximately 95% of the area under the curve lies within ± 1.96 standard deviations from the mean.

Interpretation If the estimated ACF value at a particular lag falls outside this range, it is statistically significant at the 95% confidence level (i.e., unlikely to be a result simply due to random chance). If it remains within the interval, it suggests that the observed correlation could be attributed to randomness in the data rather than a meaningful linear relationship.

14.1.5. 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 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. (14.30).

14.2. Power Spectral Density (PSD)

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 137 and 141) of stationary random signal is the Fourier transform of the ACF,

$$S_{\mathbf{xx}}[k] = \text{DFT}\left\{R_{\mathbf{x}}[n]\right\} = \left|\text{DFT}\left\{x[n]\right\}\right|^2 \; k = 0, \ldots, N-1 \tag{14.33}$$

Important properties are

- $S_{\mathbf{xx}}[k] \in \mathscr{R}$, due to the symmetry (Eq. (14.22)) 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.

Interpretation 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.

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). Conversely, if the ACF shows periodicity, the PSD will have distinct peaks at the corresponding harmonic frequencies.

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.

14.3. 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

$$\begin{aligned} y[n] &= h_1 y[n-1] + h_2 y[n-2] + \cdots + h_p y[n-p] + \epsilon[n] \\ &= \sum_{m=1}^{p} h_m y[n-m] + \epsilon[n] \end{aligned} \tag{14.34}$$

and the prediction is of the form

$$\hat{y}[n] = \sum_{m=1}^{p} h_m y[n-m] \tag{14.35}$$

where $h_1, \ldots, h_p$ are the AR model coefficient 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

$$\hat{\mathbf{y}} = \begin{bmatrix} \hat{y}[1] \\ \hat{y}[2] \\ \vdots \\ \hat{y}[L-2] \\ \hat{y}[L-1] \end{bmatrix}, \; \mathbf{X} = \begin{bmatrix} y[0] & 0 & \cdots & 0 \\ y[1] & y[0] & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ y[L-3] & y[L-4] & \cdots & y[L-p-2] \\ y[L-2] & y[L-3] & \cdots & y[L-p-1] \end{bmatrix}, \; \mathbf{h} = \begin{bmatrix} h_1 \\ h_2 \\ \vdots \\ h_p \end{bmatrix} \tag{14.36}$$

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

$$\mathcal{L}(h_i) = \sum_{n} \big(y[n] - \hat{y}[n]\big)^2 = \|\mathbf{y} - \mathbf{X}\mathbf{h}\|^2. \tag{14.37}$$

The LS solution is straightforward,

$$\mathbf{h} = \Big(\mathbf{X}^T \mathbf{X}\Big)^{-1} \mathbf{X}^T \mathbf{y}. \tag{14.38}$$

Note, the practical approach for evaluation of $\mathbf{h}$ is presented in the following section.

Example 14.1: Learn linear prediction of $y[8]$ for $p = 3$ and signal $y[0], y[1], \ldots, y[7]$.

Solution:

$$\mathbf{X} = \begin{bmatrix} y[0] & 0 & 0 \\ y[1] & y[0] & 0 \\ y[2] & y[1] & y[0] \\ y[3] & y[2] & y[1] \\ y[4] & y[3] & y[2] \\ y[5] & y[4] & y[3] \\ y[6] & y[5] & y[4] \end{bmatrix}, \; \mathbf{h} = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix} \; \mathbf{y} = \begin{bmatrix} y[1] \\ y[2] \\ y[3] \\ y[4] \\ y[5] \\ y[6] \\ y[7] \end{bmatrix} \tag{14.39}$$

Finding vector $\mathbf{h}$ values is (almost) trivial by a minimum of the loss function of the form

$$\mathcal{L} = \|\mathbf{y} - \mathbf{X}\mathbf{h}\|^2. \tag{14.40}$$

Once found, $\hat{y}[8] = h_1 y[7] + h_2 y[6] + h_3 y[5]$.

14.3.1. 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,

$$\begin{aligned}\mathbf{R} &= \mathbf{X}^T\mathbf{X} \\ &= \begin{bmatrix} R_{\mathbf{yy}}[0] & R_{\mathbf{yy}}[1] & \cdots & R_{\mathbf{yy}}[p-1] \\ R_{\mathbf{yy}}[1] & R_{\mathbf{yy}}[0] & \cdots & R_{\mathbf{yy}}[p-2] \\ \vdots & \vdots & \ddots & \vdots \\ R_{\mathbf{yy}}[p-1] & R_{\mathbf{yy}}[p-2] & \cdots & R_{\mathbf{yy}}[0] \end{bmatrix},\end{aligned} \tag{14.41}$$

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

$$R_{\mathbf{yy}}[0] = \sum_{n=0}^{L-1} y^2[n] \approx \sum_{n=0}^{L-\mathbf{2}} y^2[n] \tag{14.42}$$

The vector $\mathbf{X}^T\mathbf{y}$ is also comprised of the corresponding $R_{\mathbf{yy}}[k]$ values,

$$\mathbf{r} = \mathbf{X}^T\mathbf{y} = \begin{bmatrix} R_{\mathbf{yy}}[1] \\ R_{\mathbf{yy}}[2] \\ \vdots \\ R_{\mathbf{yy}}[p] \end{bmatrix} \tag{14.43}$$

Using these formulations, the resulting coefficients are given by a solution of a set of linear equations,

$$\mathbf{Rh} = \mathbf{r} \tag{14.44}$$

and the result is similar to Eq. (14.38),

$$\mathbf{h} = \mathbf{R}^{-1}\mathbf{r} \tag{14.45}$$

The use of a $R_{\mathbf{xx},biased}[k]$ (biased ACF, Eq. (14.14)) guaranties the numerically stable solution [18, Theorem 6.1].

Squared error The resulting loss (squared error) is given by (following Eqs. (3.22) and (14.24))

$$\mathcal{L}_{min} = R_{\mathbf{xx}}[0] - \sum_{k=1}^{p} h_k R_{\mathbf{xx}}[k] \tag{14.46}$$

Theoretically, higher value of p results in lower loss in the presence of the sufficiently long correlation time (Eq. (14.29)). Note, the accuracy drops for high values of p to due to reduced accuracy of $R_{\mathbf{xx}}[p]$.

The commonly adopted notation of the signal model in most of the books and software packages is

$$\sum_{m=0}^{p} a_m y[n-m] = \epsilon[n] \tag{14.47}$$

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,

$$\hat{y}[n] = \mu + h_1 y[n-1] + h_2 y[n-2] + \cdots + h_p y[n-p] + \epsilon[n] \tag{14.48}$$

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 $O\left(L^3\right)$.
- 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 (14.38)).
- 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`.

14.3.2. 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,

$$\begin{aligned} \hat{y}[n] &= \frac{1}{p}\left(y[n-1] + \cdots + y[n-p]\right) \\ &= \frac{1}{p}\sum_{m=1}^{p} y[n-m] \end{aligned} \tag{14.49}$$

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.

14.3.3. Nearest Neighbor (Naïve)

A simple baseline is to use the immediate past sample as the prediction with $h_1 = 1$,

$$\hat{y}[n] = y[n-1] \tag{14.50}$$

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.

[1] For example, `solve_toeplitz`

14.4. 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,

$$\hat{y}_k[n] = \phi_{k,1} y[n-1] + \phi_{k,2} y[n-2] + ... + \phi_{k,k} y[n-k] + \epsilon[n] \tag{14.51}$$

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

$$\hat{y}_1[n] = \phi_{1,1} y[n-1] + \epsilon[n] \tag{14.52}$$

  and the coefficient $\beta[1] = \hat{\phi}_{1,1}$ is given by the model solution, h_1 (Eq. (14.5)).
- For the second order PACF, it would be the $\beta[2] = \phi_{2,2}$ coefficient of AR(2) model,

$$\hat{y}_2[n] = \phi_{2,1} y[n-1] + \phi_{2,2} y[n-2] + \epsilon[n] \tag{14.53}$$

- 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.

14.4.1. 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. 14.2.

[2] Stackexchange

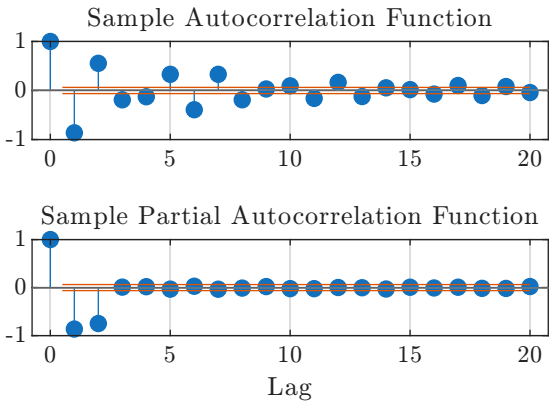


Figure 14.2.: ACF and PACF of AR(2) signal. Note short PACF and long ACF plots.

14.5. 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.

Another model is the moving average model (MA), where the output is a linear combination of the noise values at the different times [14, Example 4.3, pp. 90]

$$y[n] = \epsilon[n] + b_1\epsilon[n-1] + \cdots + c_q\epsilon[n-q] \tag{14.54}$$

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.

14.5.1. 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),

$$\begin{aligned} y[n] &= h_1 y[n-1] + \epsilon[n] \\ &= h_1(h_1 y[n-2] + \epsilon[n-1]) + \epsilon[n] \\ &= h_1^2 y[n-2] + h_1\epsilon[n-1] + \epsilon[n] \\ &= h_1^3 y[n-3] + h_1^2\epsilon[n-2] + h_1\epsilon[n-1] + \epsilon[n] \end{aligned} \tag{14.55}$$

[3] e.g. A simple Estimator of an MA(1) Models.

Continuing this process indefinitely and assuming stability, $h_1 < 1$, $\lim_{k\to\infty} h_1^k \to 0$, we have

$$\begin{aligned} y[n] &= \epsilon[n] + h_1\epsilon[n-1] + h_1^2\epsilon[n-2] + h_1^3\epsilon[n-3] + \cdots \\ &= \sum_{i=0}^{\infty} h_1^i \epsilon[n-i] \end{aligned} \tag{14.56}$$

that it is MA(∞) model, where the coefficients of the MA representation are the infinite geometric sequence powers of h_1.

Presenting MA(1) as AR(∞) An MA(1) process is given by

$$x[n] = \epsilon[n] + c_1\,\epsilon[n-1]. \tag{14.57}$$

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

$$\begin{aligned} \epsilon[n] &= x[n] - c_1\left(x[n-1] - c_1\,\epsilon[n-2]\right) \\ &= x[n] - c_1\,x[n-1] + c_1^2\,\epsilon[n-2] \\ &= x[n] - c_1\,x[n-1] + c_1^2\left(x[n-2] - c_1\,\epsilon[n-3]\right) \\ &= x[n] - c_1\,x[n-1] + c_1^2\,x[n-2] - c_1^3\,\epsilon[n-3] \\ &\vdots \\ &= \sum_{i=0}^{\infty}(-c_1)^i\,x[n-i]. \end{aligned} \tag{14.58}$$

Rearranging, the AR(∞) representation becomes

$$x[n] = \epsilon[n] + \sum_{i=1}^{\infty}(-c_1)^i\,x[n-i]. \tag{14.59}$$

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.

14.5.2. 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. 14.4.1), 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. 14.3. After lag 4, the ACF values remain within the confidence bounds, essentially zero, suggesting the data arise from an MA(4) process.

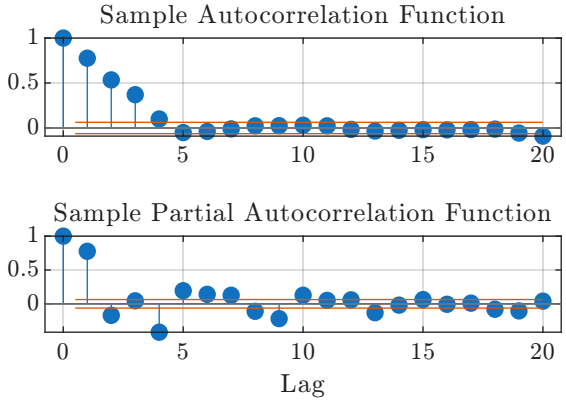


Figure 14.3.: ACF and PACF of a synthetic MA(4) signal. Note short ACF and long PACF plots.

14.6. 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

$$\begin{aligned}\hat{y}[n] &= h_1 y[n-1] + \cdots + h_p y[n-p] \\ &\quad + c_1 \epsilon[n-q] + \cdots + c_q \epsilon[n-q] + \epsilon[n] \\ &= \sum_{i=1}^{p} h_i y[n-i] + \sum_{k=0}^{q} c_k \epsilon[n-k]\end{aligned}, \tag{14.60}$$

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,

$$\hat{y}[n] - \sum_{i=1}^{p} h_i y[n-i] = \sum_{k=0}^{q} c_k \epsilon[n-k] \tag{14.61}$$

As with AR models (Eq. (14.48)), a constant bias μ can be included,

$$\hat{y}[n] = \sum_{i=1}^{p} h_i y[n-i] + \sum_{k=0}^{q} c_k \epsilon[n-k] + \mu \tag{14.62}$$

One of the adopted notation of the ARMA model is

$$\sum_{m=0}^{p} a_m y[n-m] = \sum_{k=0}^{q} c_k \epsilon[n-k] \tag{14.63}$$

with $a_0 = 1$ and $a_m = -h_m$.

15. ARX

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.

Goal: Extension for AR model to ARX model.

The $ARX(p, q)$ model is given by

$$\begin{aligned} y[n] &= a_1 y[n-1] + \cdots + h_p y[n-p] \\ &\quad + b_1 x[n-1] + \cdots + b_q x[n-k] + \epsilon[n], \end{aligned} \tag{15.1}$$

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.

15.1. Cross-Correlation Function

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,

$$\hat{y}[n] = b_k x[n-k], \tag{15.2}$$

The resulting MSE-based loss function is of the form

$$\mathcal{L}(b) = \frac{1}{2} \sum_n \big(y[n] - b_k x[n-k]\big)\big)^2 \tag{15.3}$$

with the solution by

$$\frac{d\mathcal{L}(b)}{db} = \sum_n (y[n] - b_k x[n-k])(-x[n-k]) = 0 \tag{15.4}$$

The corresponding solution is

$$b_k = \frac{\sum_n y[n]x[n-k]}{\sum_n x^2[n-k]}. \tag{15.5}$$

Cross-Correlation Function The resulting coefficients are related to the cross-correlation function,

$$R_{\mathbf{xy}}[k] = \sum_n x[n]y[n-k], k = -L+1, \ldots, L-1 \tag{15.6}$$

Similar to the ACF, cross-correlation can also be defined in biased, unbiased, or normalized forms:

$$R_{\mathbf{xy},biased}[k] = \frac{1}{L} R_{\mathbf{xy}}[k] \tag{15.7}$$

$$R_{\mathbf{xy},unbiased}[k] = \frac{1}{L-|k|} R_{\mathbf{xy}}[k] \tag{15.8}$$

$$R_{\mathbf{xy},norm}[k] = \frac{R_{\mathbf{xy}}[k]}{\sqrt{R_{\mathbf{x}}[0]R_{\mathbf{y}}[0]}} \tag{15.9}$$

Note, these modification are available only if $x[n]$ and $y[n]$ are of the same length. Otherwise, only Eq. (15.6) is used.

The normalized cross-correlation function has correlation coefficient interpretation,

$$R_{\mathbf{xy},norm}[k] \approx \rho_{\mathbf{xy}}[k] \tag{15.10}$$

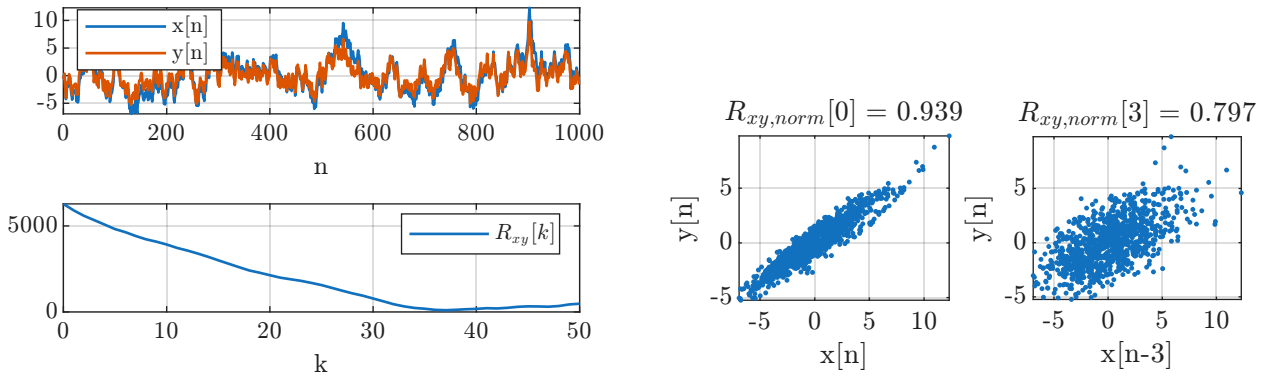
Properties:

$$R_{\mathbf{xy}}[k] = R_{\mathbf{yx}}[-k] \tag{15.11}$$

$$R_{\mathbf{xy}}[-k] = R_{\mathbf{yx}}[k] \tag{15.12}$$

$$\left|R_{\mathbf{xy}}[k]\right| \leqslant \sqrt{R_{\mathbf{x}}[0]R_{\mathbf{y}}[0]} \tag{15.13}$$

$$\left|R_{\mathbf{xy}}[k]\right| \leqslant \frac{1}{2}\left[R_{\mathbf{x}}[0] + R_{\mathbf{y}}[0]\right] \tag{15.14}$$



(a) Cross-correlation

(b) $y[n]$ vs. $x[n-k]$

Figure 15.1.: Illustration of the linear dependence between $y[n]$ and $x[n-k]$.

Example 15.1: The solution in Eq. (13.17) is

$$\hat{A} = \frac{R_{\mathbf{xy}}[0]}{R_{\mathbf{x}}[0]} \tag{15.15}$$

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 $y[n]$.

15.1.1. 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. 14.1.5.

15.2. 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 [14, Example 4.3, pp. 90]

$$\begin{aligned} y[n] &= b_1 x[n-1] + \cdots + b_q x[n-q] + \epsilon[n] \\ &= \sum_{k=1}^{q} b_k x[n-k] + \epsilon[n] \end{aligned} \tag{15.16}$$

In matrix form, the past values are arranged into a matrix $\mathbf{X}$,

$$\underbrace{\begin{bmatrix} \hat{y}[1] \\ \hat{y}[2] \\ \vdots \\ \hat{y}[L-1] \end{bmatrix}}_{\hat{\mathbf{y}}} = \underbrace{\begin{bmatrix} \begin{bmatrix} x[0] & 0 & \cdots & 0 \\ x[1] & x[0] & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ x[L-2] & x[L-3] & \vdots & x[L-m-2] \end{bmatrix} \end{bmatrix}}_{\mathbf{X}} \underbrace{\begin{bmatrix} b_1 \\ \vdots \\ b_q \end{bmatrix}}_{\mathbf{b}} \tag{15.17}$$

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 $x[n]$.

15.3. 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 15.2: ARX(3,3) model with signals

$$\begin{aligned} x[n] &= x[0], x[1], \ldots, x[7] \\ y[n] &= y[0], y[1], \ldots, y[7] \end{aligned}$$

The required difference equation is

$$\begin{aligned}\hat{y}[n] = h_1 y[n-1] + h_2 y[n-2] + h_3 y[n-2] \\ + b_1 x[n-1] + b_2 x[n-2] + b_3 x[n-3]\end{aligned} \tag{15.18}$$

Find prediction of $\hat{y}[8]$.

Solution: The coefficients are given by

$$\arg\min_{\mathbf{w}} \|\mathbf{y} - \mathbf{X}\mathbf{w}\| \tag{15.19}$$

where

$$\mathbf{X} = \begin{bmatrix} x[0] & 0 & 0 & y[0] & 0 & 0 \\ x[1] & x[0] & 0 & y[1] & y[0] & 0 \\ x[2] & x[1] & x[0] & y[2] & y[1] & y[0] \\ x[3] & x[2] & x[1] & y[3] & y[2] & y[1] \\ x[4] & x[3] & x[2] & y[4] & y[3] & y[2] \\ x[5] & x[4] & x[3] & y[5] & y[4] & y[3] \\ x[6] & x[5] & x[4] & y[6] & y[5] & y[4] \end{bmatrix}, \tag{15.20}$$

$$\mathbf{w} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix}, \mathbf{y} = \begin{bmatrix} y[1] \\ y[2] \\ y[3] \\ y[4] \\ y[5] \\ y[6] \\ y[7] \end{bmatrix}$$

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.

The synthetic prediction example is show in Fig. 15.2. In the provided figure, a synthetic prediction example is shown, where the input is a noise-corrupted binary signal processed through an unknown filter. The ARX model, using estimated parameters, approximates this filtering process and predicts future outputs.

15.4. 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,

$$\begin{cases} x[n] \\ y[n] = x[n] + \epsilon[n] \end{cases} \tag{15.21}$$

In another one (Wiener filter), noise and noisy versions are available,

$$\begin{cases} \varepsilon[n] \\ y[n] = x[n] + \epsilon[n] \end{cases} \tag{15.22}$$

In either case, the learned AR(p) model can be used.

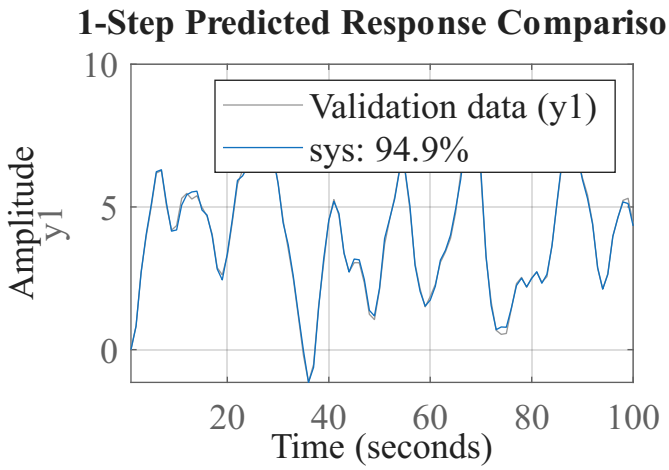


Figure 15.2.: Example of ARX model-based prediction. The input is noise-corrupted binary signal that passed through a filter.

Noise filtering (denoising)

We have training data $\{x[n], y[n]\}_{n=0}^{L-1}$ and we are interested to learn coefficients h_k, such that

$$\hat{x}[n] = \sum_{k=0}^{p} h_k y[n-k] \tag{15.23}$$

The key idea is to construct the AR matrix $\mathbf{X}$ (as in (14.36)) 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

$$\begin{aligned} \mathbf{X} &= \begin{bmatrix} y[p] & y[p-1] & \cdots & y[0] \\ y[p+1] & y[p] & \dots & y[1] \\ \vdots & \cdots & \ddots & \vdots \\ y[L-1] & y[L-2] & \cdots & y[L-1-p] \end{bmatrix}, \\ \mathbf{y} &= \begin{bmatrix} x[p] \\ x[p+1] \\ \vdots \\ x[L-1] \end{bmatrix} \end{aligned} \tag{15.24}$$

and the resulting coefficients can be found by the normal equation.

Using notation in Sec. 14.3.1, 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.

15.5. 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),

$$\begin{aligned} y[n] &= h_1 y[n-1] + \cdots + h_{n_a} y[n-n_a] \\ &+ b_1 x[n-1] + \cdots + b_{n_b} x[n-n_b] \\ &+ c_1 \epsilon[n-1] + \cdots + c_{n_c} \epsilon[n-n_c] + \epsilon[n] \end{aligned} \tag{15.25}$$

15.6. 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.

De-trending

The basic model with linear trend is

$$y[n] = A + Bn + \epsilon[n], \tag{15.26}$$

where B is the slope of the trend. Let's define

$$\begin{aligned} y[n] - y[n-1] &= A + Bn\epsilon[n] - A - B(n-1) - \epsilon[n-1] \\ &= B + \epsilon[n] - \epsilon[n-1] \end{aligned} \tag{15.27}$$

This is known as first-order differencing that effectively removes the constant slope.

The quadratic (or parabolic) trend is given by

$$x(t) = at^2 + bt + c \tag{15.28}$$

Applying the differencing twice,

$$\begin{aligned} y'[n] &= y[n] - y[n-1] \\ y''[n] &= y'[n] - y'[n-1] \\ &= y[n] - y[n-1] - (y[n-1] - y[n-2]) \\ &= y[n] - 2y[n-1] + y[n-1] \end{aligned}$$

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.

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.

15.7. 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:

$$\hat{y}[n] = f(y[n-1], \ldots, y[n-p]) + \epsilon[n] \tag{15.29}$$

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

$$\hat{y}[n] \;=\; f(\mathbf{x}, \mathbf{y}) + \epsilon[n] \tag{15.30}$$

The network learns an arbitrary non-linear mapping from the selected lags of $y[n]$ and $x[n]$ to the current output.

15.8. Vector AR (VAR)

The goal is to use AR multivariate prediction of an N-dimensional signal $\mathbf{y}[n]$ by its L historic values,

$$\left\{ \begin{bmatrix} y_0[0] \\ y_1[0] \\ \vdots \\ y_{N-1}[0] \end{bmatrix}, \begin{bmatrix} y_0[1] \\ y_1[1] \\ \vdots \\ y_{N-1}[1] \end{bmatrix} \cdots \begin{bmatrix} y_0[L-1] \\ y_1[L-1] \\ \vdots \\ y_{N-1}[L-1] \end{bmatrix} \right\} \tag{15.31}$$

VAR(1) Model

For example, the VAR(1) model of 2-dimensional signal ($N = 2$) is

$$\begin{bmatrix} \hat{y}_0[n] \\ \hat{y}_1[n] \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{bmatrix} \begin{bmatrix} y_0[n-1] \\ y_1[n-1] \end{bmatrix} + \begin{bmatrix} \epsilon_0[n] \\ \epsilon_1[n] \end{bmatrix} \tag{15.32}$$

In compressed matrix notation,

$$\hat{\mathbf{y}}[n] = \mathbf{A}\mathbf{y}[n-1] + \boldsymbol{\epsilon}[n] \tag{15.33}$$

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,

$$\begin{aligned}\hat{\mathbf{y}}[n] &= \mathbf{A}\mathbf{y}[n-1] + \boldsymbol{\epsilon}[n] \\ &= \mathbf{A}(\mathbf{A}\mathbf{y}[n-2] + \boldsymbol{\epsilon}[n-1]) + \boldsymbol{\epsilon}[n] \\ &= \mathbf{A}^2\mathbf{y}[n-2] + \mathbf{A}\boldsymbol{\epsilon}[n-1] + \boldsymbol{\epsilon}[n] \\ &= \mathbf{A}^3\mathbf{y}[n-3] + \mathbf{A}^2\boldsymbol{\epsilon}[n-2] + \mathbf{A}\boldsymbol{\epsilon}[n-1] + \boldsymbol{\epsilon}[n]\end{aligned} \tag{15.34}$$

The stability criterion is $\left|\lambda_{max}(\mathbf{A})\right| < 1$.

Biased form In biased form,

$$\hat{\mathbf{y}}[n] = \boldsymbol{\mu} + \mathbf{A}\mathbf{y}[n-1] + \boldsymbol{\epsilon}[n] \tag{15.35}$$

biases $\boldsymbol{\mu} = \begin{bmatrix}\mu_0 \\ \mu_1\end{bmatrix}$.

VAP(p) Model

The further extension to p values history is straightforward,

$$\hat{\mathbf{y}}[n] = \mathbf{A}_1\mathbf{y}[n-1] + \mathbf{A}_2\mathbf{y}[n-2] + \ldots + \mathbf{A}_p\mathbf{y}[n-p]\boldsymbol{\epsilon}[n] \tag{15.36}$$

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.

Part III.

Learning in Signals

16. Feature Extraction from Signals

Goal: Feature extraction (FE) from signals: converting raw signal windows into numerical feature vectors suitable for ML models.

Contents

16.1. Windowing

Goal: Divide a continuous signal into fixed-length segments (windows) suitable for feature extraction.

Non-overlapping windows (Fig. 16.1a) are the standard choice. Overlapping windows (Fig. 16.1b) 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

$$r = \frac{L - S}{L}, \tag{16.1}$$

where $r = 0$ corresponds to non-overlapping windows and $r = 0.5$ to 50% overlap.

> Overlapping windows may introduce data leakage: adjacent windows share samples, so train and test sets may contain nearly identical data if split carelessly.

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.

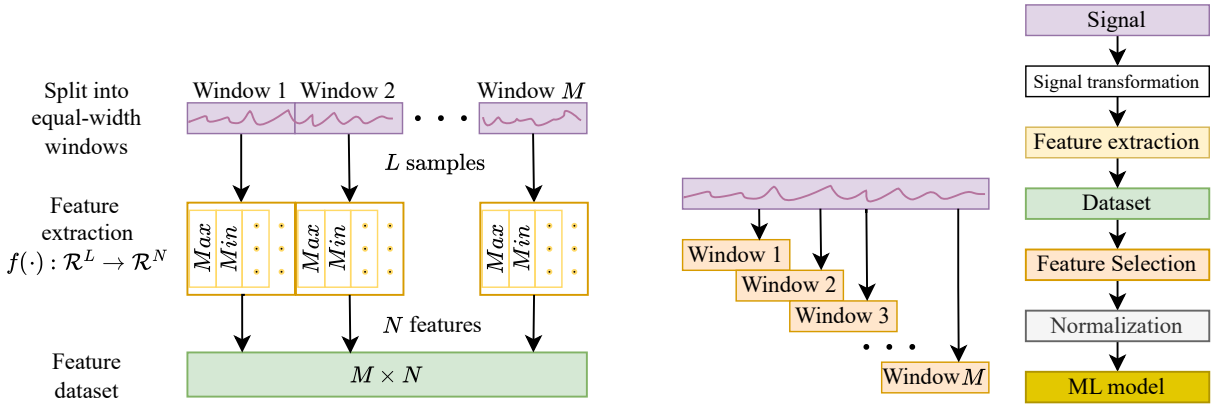
- Hand-picked by visual analysis.
- Hyper-parameter (the worst case).

16.2. Workflow

Goal: Understand the end-to-end pipeline from raw signals to a feature dataset ready for ML.

Feature extraction: A mapping $f : \mathbb{R}^L \to \mathbb{R}^N$ that converts a signal window of L samples into a feature vector of N numerical values.

FE from signals is presented in Fig. 16.1a.



(a) Non-overlapping windows (b) Overlapping windows (c) Typical workflow

Figure 16.1.: Feature extraction from signals. Dataset is further applied for ML tasks, such as regression or classification.

Typical dimensions:

- L may vary between 100s to 10,000s samples.
- N may vary between few to 1,000s features.

Feature extraction and feature selection are distinct steps. FE creates new features from raw data; feature selection (FS) then removes redundant or irrelevant features from the extracted set.

FE from images FE from images uses the similar principles. The main difference is in the applied image feature functions.

16.3. Signal transformation

Goal: Functional mapping of a signal to improve prediction (or classification). The result can be:

- Univariate to another univariate signal (basic case).
- Univariate to multivariate set of signals for further multivariate processing (advanced case).

Notes:

- The transformations have to be invertible.
- Some of the transformations are restricted only for positive signals, $y[n] > 0 \, \forall n$.

Some of the common transformations:

- Logarithmic transformation

$$\tilde{y}[n] = \begin{cases} \log\big(y[n] + 1\big) & y[n] \geq 0 \\ \log\big(y[n]\big) & y[n] > 0 \end{cases} \tag{16.2}$$

  We use $(+1)$ to avoid value of 0.
- Square root transformation,

$$\tilde{y}[n] = \sqrt{y[n]} \tag{16.3}$$

- Exponential transformation,

$$\tilde{y}[n] = y^m[n] \tag{16.4}$$

- Modified Box-Cox transformations [2],

$$w_t = \begin{cases} \log(y_t) & \text{if } \lambda = 0; \\ \text{sign}(y_t)(|y_t|^\lambda - 1)/\lambda & \text{otherwise.} \end{cases} \tag{16.5}$$

  with inverse

$$y_t = \begin{cases} \exp(w_t) & \text{if } \lambda = 0; \\ \text{sign}(\lambda w_t + 1)|\lambda w_t + 1|^{1/\lambda} & \text{otherwise.} \end{cases} \tag{16.6}$$

Example 16.1: A sensor signal $y[n]$ with values in [0, 10,000] is highly right-skewed (Fig. 16.2). 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.

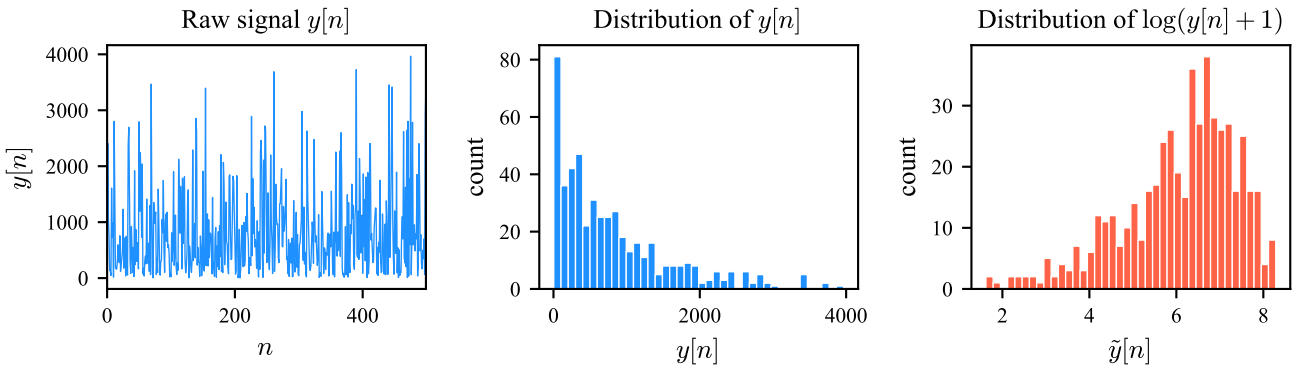


Figure 16.2.: Log transformation of a right-skewed signal.

16.4. Signal Features

Goal: Numerical functions that convert raw signal windows into scalar values that capture the characteristics of interest.

Statistical (time-domain) features

Descriptive statistics computed directly on the signal window $y[1], \ldots, y[L]$:

- Mean:
$$\bar{y} = \frac{1}{L} \sum_{n=1}^{L} y[n] \tag{16.7}$$
- Variance:
$$\sigma^2 = \frac{1}{L} \sum_{n=1}^{L} \Big(y[n] - \bar{y} \Big)^2 \tag{16.8}$$
- Root mean square (RMS):
$$y_{\text{rms}} = \sqrt{\frac{1}{L} \sum_{n=1}^{L} y^2[n]} \tag{16.9}$$
- Zero-crossing rate (ZCR):
$$\text{ZCR} = \frac{1}{L-1} \sum_{n=2}^{L} \mathbf{1}\Big[y[n] \cdot y[n-1] < 0 \Big] \tag{16.10}$$
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 Fourier transform $Y[k]$ of the signal window:

- Spectral centroid — the "center of mass" of the spectrum:
$$\text{SC} = \frac{\sum_{k=1}^{K} f_k |Y[k]|^2}{\sum_{k=1}^{K} |Y[k]|^2} \tag{16.11}$$
where f_k is the frequency of the k-th bin.
- Spectral bandwidth, spectral rolloff, spectral flatness.

Auto-correlation based features

Time-domain features derived from the auto-correlation function, capturing periodicity and temporal structure.

Mixed-domain features

Features based on STFT and wavelet transforms, capturing both time and frequency information simultaneously.

Examples: tsfel feature list

Field-tailored features

Historically-developed features. For example, cepstrum in speech processing, that is logarithm of Fourier transform.

> Features from different domains (statistical, spectral, etc.) may have very different scales. Feature normalization is recommended before combining them into a single feature vector.

16.4.1. Output

At the end of the FE process, each signal window of L samples is mapped to a single N-dimensional feature vector. Stacking all M windows yields an $M \times N$ feature matrix (dataset) ready for ML (Fig. 16.1a).

Complexity Some features (e.g. spectral or wavelet-based) have non-negligible computational cost, especially for large L. Speed-up can be achieved by:

- Parallel computation: since windows are independent, they can be distributed across multiple CPU/GPU cores.
- Selecting a compact feature subset (see feature selection) to avoid computing unnecessary features.

16.5. Dedicated Libraries

Goal: Overview of ready-to-use toolboxes for time-series feature extraction and selection.

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.

`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.

`mlxtend` Supplementary ML tools including sequential feature selection (forward/backward) and model evaluation utilities.

16.6. Train-Test Split in Signals

Goal: Correctly partition signal data into training and test sets without introducing data leakage.

Train and test cannot be adjacent windows.

> 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.

16.6.1. 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 *group splitting*[1] (or *leave-one-group-out* CV). It ensures the model is evaluated on truly unseen sources, not on different windows from a source it has already learned.

Example 16.2: 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.

16.6.2. 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.

[1] In Python: `sklearn.model_selection.GroupKFold`.

Time-series cross-validation Standard k-fold CV shuffles the data, violating temporal order. Instead, time-series CV (Fig. 16.3) 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).

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.

Example in Python

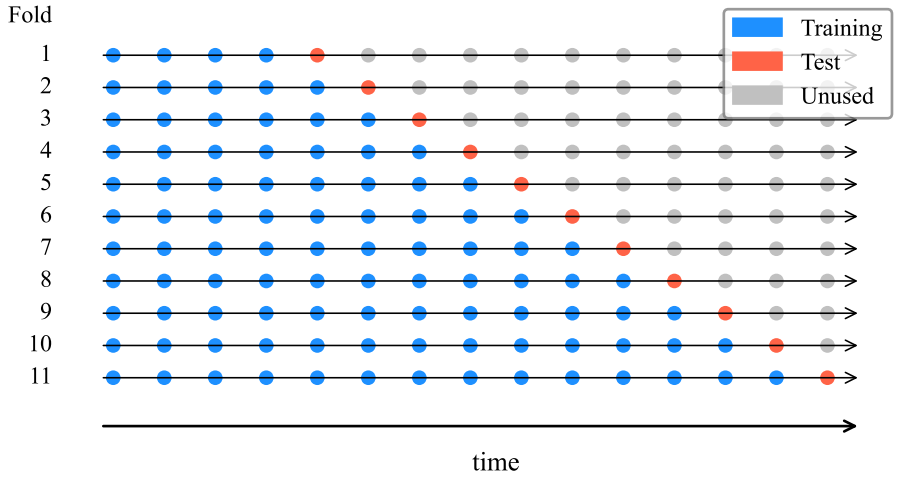


Figure 16.3.: Time-series cross-validation with expanding window. Each row is one fold: training samples (blue) always precede the test sample (orange); future samples (gray) are unused (inspired by source).

16.7. Summary

- Feature extraction maps signal windows of L samples to N-dimensional feature vectors via $f : \mathbb{R}^L \to \mathbb{R}^N$.
- Non-overlapping windows are preferred to avoid redundancy and data leakage.
- Signal transformations (log, Box-Cox, etc.) can improve downstream model performance by reshaping distributions.
- Features span statistical, spectral, auto-correlation, and mixed domains; dedicated libraries (tsfresh, tsfel, catch22) provide ready-to-use implementations.
- Train-test splitting must respect temporal or source structure — random splits are invalid for signal data.

17. Signal classification

Goal: Classify signal.

- Direct approach: distance metrics and classifier.
  Distance metrics examples: MSE, DTW
- Feature approach: feature extraction and classifier.
- Shapelets: discriminative sub-sequences of time-series data and classifier.

17.1. 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

Two signal are defined [11],

$$\begin{aligned} X &= (x_1, \ldots, x_m, \ldots, x_M) \\ Y &= (y_1, \ldots, x_n, \ldots, x_N) \end{aligned} \tag{17.1}$$

The dissimilarity function is the input of DTW algorithm,

$$d_{mn} = f(x_m, y_n) \geq 0 \tag{17.2}$$

For example, it can (1D) euclidean distance, $d_{mn} = |x_m - y_n|$. The selected path has the following restrictions. Start and end points are d_{11} and d_{MN}. Possible moves are monotonous,

- Vertical: $(m, n) \rightarrow (m + 1, n)$
- Horizontal: $(m, n) \rightarrow (m, n + 1)$
- Diagonal: $(m, n) \rightarrow (m + 1, n + 1)$

No skips or turns back are possible. The minimum distance path $\sum d_{mn}$ is calculated. This path can serve as metric of "distance" between two stretch signals.

Example 17.1: Two signals of different lengths: X is a chirp of $M = 1000$ samples and Y is a sinusoid of $N = 399$ samples (Fig. 17.1). The cost matrix $d_{mn} = |x_m - y_n|$ and the optimal warping path are shown in Fig. 17.2. After alignment along the optimal path, the two signals match closely (Fig. 17.3), demonstrating how DTW stretches the time axis to find the best correspondence.

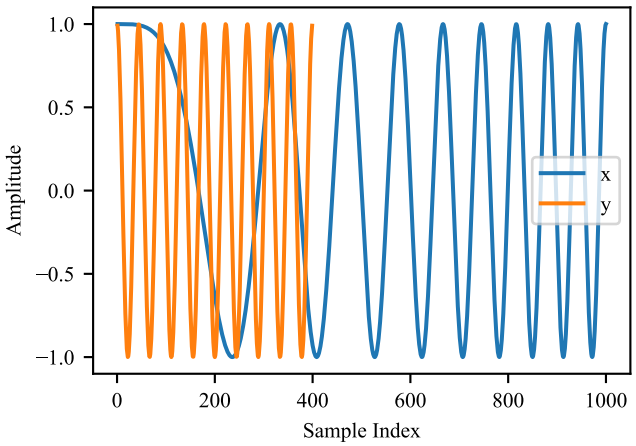


Figure 17.1.: Signal examples.

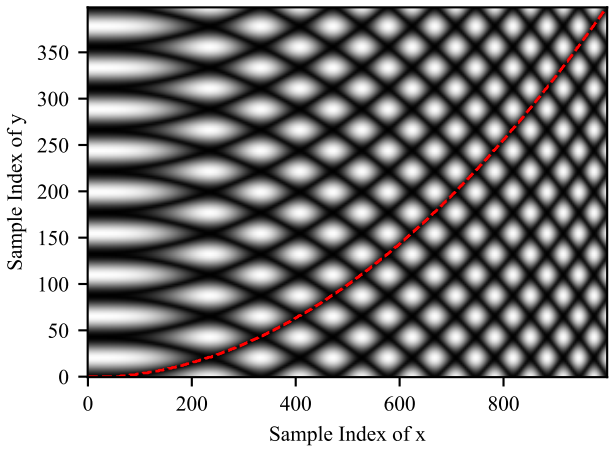


Figure 17.2.: d_{mn} with the optimal path.

Multivariate DTW

Same as univariate, except distance definition. For example, for (complex signals) of dimension K the euclidean distance is

$$d_{mn} = \sqrt{\sum_{k=1}^{K} \left(x_{k,m} - y_{k,n}\right) \left(x_{k,m} - y_{k,n}\right)^*} \tag{17.3}$$

17.2. 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" [20].

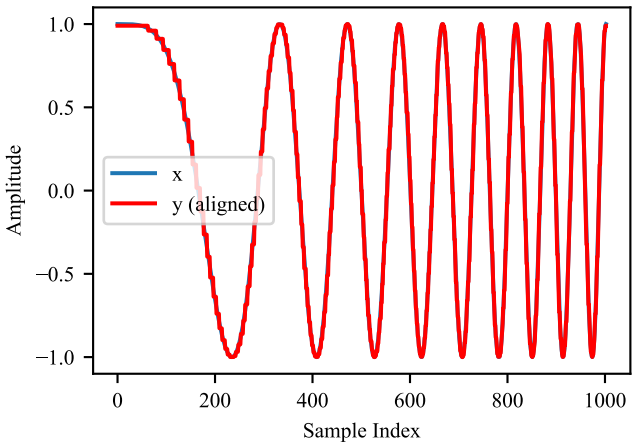


Figure 17.3.: Aligned signals.

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:

$$d(S, X) = \min_{i=1,\ldots,M-l+1} \sqrt{\frac{1}{l} \sum_{j=1}^{l} (s_j - x_{i+j-1})^2} \tag{17.4}$$

This sliding-window RMSE measures how well the shapelet matches anywhere in the signal (Fig. 17.4).

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).

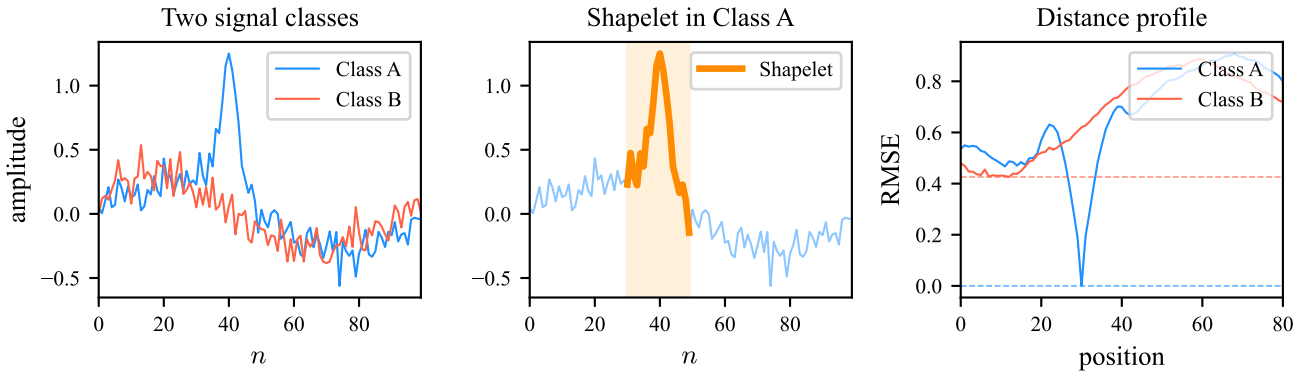


Figure 17.4.: Shapelet-based classification. Left: two signal classes (Class A contains a characteristic bump). Center: the shapelet (orange) extracted from Class A. Right: distance profile — Class A has a near-zero minimum distance, while Class B remains high, enabling discrimination.

17.3. 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.

Variants

Random Interval Spectral Ensemble (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).

17.4. 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 17.2: 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. 17.5, 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.

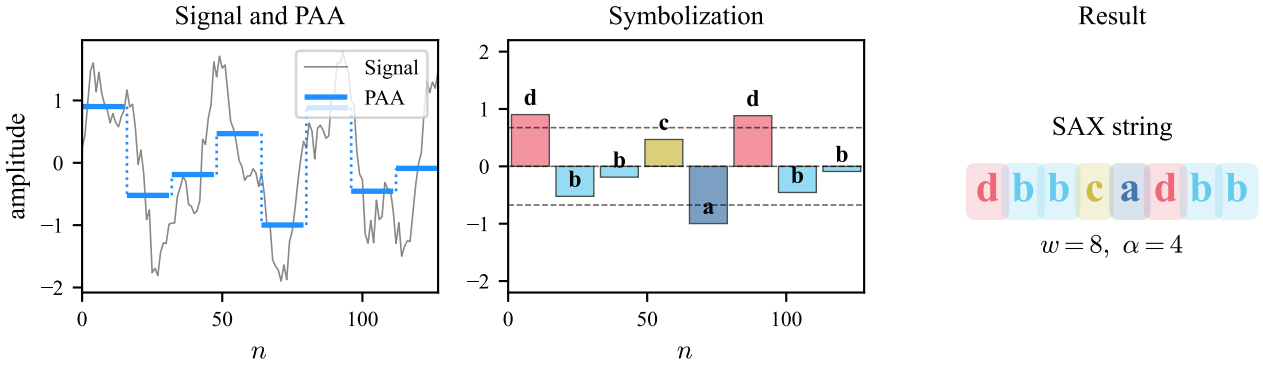


Figure 17.5.: SAX visualization: original signal with PAA overlay (left), symbolization with breakpoints (center), and resulting SAX string (right).

SAX-VSM

SAX combined with the Vector Space Model (SAX-VSM) [19] 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).

17.5. ROCKET

Goal: Classify time series by transforming them with a large number of random convolutional kernels, then training a linear classifier on the resulting features.

RandOm Convolutional KErnel Transform (ROCKET) [7].

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.

MINIROCKET

MINIROCKET [8] 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`).

17.6. HIVE-COTE

Goal: Achieve state-of-the-art accuracy by combining classifiers that operate on different signal representations.

The Hierarchical Vote Collective of Transformation-based Ensembles (HIVE-COTE) [16] 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.

17.7. 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

18. Exponential Smoothing

18.1. Preface

Basic models:

- Naïve, Eq. (14.50).
- Moving average, Eq. (14.49).
- Signal average,

$$\hat{y}[n] = \frac{1}{n}\sum_{i=0}^{n-1} y[n] \tag{18.1}$$

18.2. Exponential Smoothing

Goal: Use decaying sequence of weights for historic values.

The sequence is of the form

$$\sum_{k=0}^{\infty}(1-\alpha)^k = \frac{1}{\alpha}, \quad 0 < \alpha < 1 \tag{18.2}$$

For example, for $\alpha = \frac{1}{2}$,

$$1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots = 2 \tag{18.3}$$

Multiply both sides of the sum by α and we can write that

$$\sum_{k=0}^{\infty}\alpha(1-\alpha)^k = \alpha + \alpha(1-\alpha) + \alpha(1-\alpha)^2 + \cdots + \alpha(1-\alpha)^k + \cdots = 1 \tag{18.4}$$

The resulting prediction (forecasting) model is

$$\begin{aligned}\hat{y}[n+1] = {} & y[n]\alpha \\ & + y[n-1]\alpha(1-\alpha) \\ & + y[n-2]\alpha(1-\alpha)^2 \\ & + \cdots \\ & + y[n-k]\alpha(1-\alpha)^k + \cdots\end{aligned} \tag{18.5}$$

The difference equation form of this model is

$$\hat{y}[n+1] = \alpha y[n] + (1-\alpha)y[n-1] \tag{18.6}$$

that starts with some $y[0]$ and $y[1]$.

The value of $\alpha \in [0, 1]$ may be considered as *memory decay rate*. For higher α the method "forgets" faster. For $\alpha = 1$, just one last sample is used, $\hat{y}[n+1] = y[n]$. For $\alpha \lesssim 1$ the sequence decay is very fast. For $\alpha \gtrsim 0$ the significant number of previous samples is involved.

This method is called *simple exponential smoothing* (SES).

Learning parameter

The learning of α is numerical minimization of some loss function,

$$\arg\min_{\alpha} \sum_{i=2}^{L} \mathcal{L}(y[i], \hat{y}[i]) \tag{18.7}$$

The most common loss is MSE,

$$\mathcal{L}(y[i], \hat{y}[i]) = (y[i] - \hat{y}[i])^2 \tag{18.8}$$

18.3. Double Exponential Smoothing

Trend

Some signals (time-series) have trend (some local slope). The basic model with trend/slope b is given by

$$y[n] = a + bn + \epsilon[n] \tag{18.9}$$

Let's define

$$y[n] - y[n-1] = a + bn - (a - b(n-1)) = b \tag{18.10}$$

This difference is called de-trending.

18.3.1. Method

The use of trend in smoothing results. The prediction is a combination of level and trend,

$$\hat{y}[n+1] = \ell_n + b_n \tag{18.11}$$

Level The level is given by

$$\begin{aligned} \ell_n &= \alpha \cdot \text{new information} + (1-\alpha) \cdot (\text{old level} + \text{trend}) \\ &= \alpha y[n] + (1-\alpha)(\ell_{n-1} + b_{n-1}) \end{aligned} \tag{18.12}$$

The relation is similar to SES with trend "correction".

Trend The trend can be used in smoothing by formula above. The "new" trend is given by difference $\ell_n - \ell_{n-1}$. The smoothing trend formula is given by

$$\begin{aligned} b_n &= \beta \cdot \text{new trend} + (1-\beta) \cdot \text{old trend} \\ &= \beta(\ell_n - \ell_{n-1}) + (1-\beta) b_{n-1} \end{aligned} \tag{18.13}$$

Both of these functions are updated consequently with initial conditions of [17, Sec. 6.4.3.3]

$$\begin{aligned} \ell_0 &= y[0] \\ b_0 &= \begin{cases} y[1] - y[0] \\ \frac{1}{3}\left[(y[1]-y[0]) + (y[2]-y[1]) + (y[3]-y[2])\right] = \frac{y[3]-y[0]}{3} \\ \dfrac{y[n-1]-y[0]}{n} \end{cases} \end{aligned} \tag{18.14}$$

This model is similar to ARIMA(0,1,1) model.

18.4. Triple Exponential Smoothing

18.4.1. Seasonality

Seasonal component is a repetitive component (cyclical repeating pattern) with some lower frequency, i.e. with period M of tens of samples or more. To identify the value of L, the closest-to-zero peak of ACF can be used.

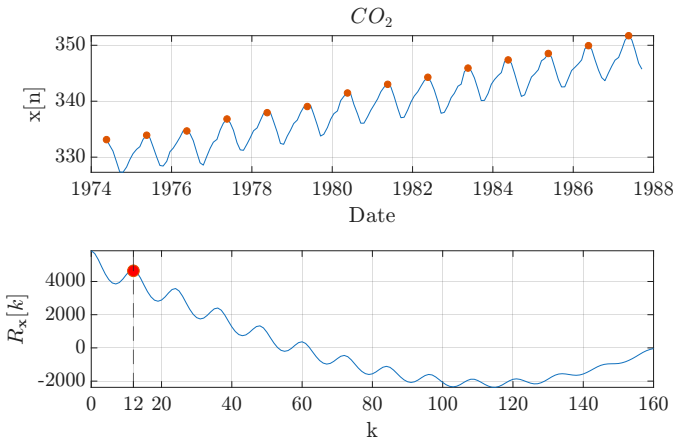


Figure 18.1.: Example of CO_2 concentrations.

18.4.2. Method

The exponential smoothing is applied on seasonality component as well. The trend component is without change. New equations set is

$$\ell_n = \alpha y[n] + (1-\alpha)(\ell_{n-1} + b_{n-1}) \tag{18.15}$$

$$b_n = \beta(\ell_n - \ell_{n-1}) + (1 - \beta)b_{n-1} \tag{18.16}$$

Moreover, it helps to make prediction for more steps in the future.

19. 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

$$e[n] = y[n] - \hat{y}[n]. \tag{19.1}$$

19.1. Scale-Dependent Metrics

Scale-dependent metrics retain the original units of the data, making them appropriate when the absolute error magnitude is meaningful.

MSE and RMSE

$$\mathrm{MSE} = \frac{1}{L}\sum_{n} e^2[n], \qquad \mathrm{RMSE} = \sqrt{\mathrm{MSE}}. \tag{19.2}$$

Heavily penalizes large errors; sensitive to outliers. MSE is also commonly used as a training loss (differentiable, convex).

MAE and MedianAE

$$\mathrm{MAE} = \frac{1}{L}\sum_{n} |e[n]|, \qquad \mathrm{MedAE} = \mathrm{median}_n\ |e[n]|. \tag{19.3}$$

Robust to outliers (especially MedAE). MAE is also used as a training loss (convex, but nondifferentiable at 0).

19.2. Scale-Free Metrics

When comparing forecasts across series with different units or scales, scale-free metrics normalize the error so that results are comparable.

MAPE and sMAPE

$$\mathrm{MAPE} = \frac{100}{L}\sum_{n} \frac{|e[n]|}{|y[n]| + \varepsilon}, \qquad \mathrm{sMAPE} = \frac{100}{L}\sum_{n} \frac{2|e[n]|}{|y[n]| + |\hat{y}[n]| + \varepsilon}. \tag{19.4}$$

A small ε is added to avoid division by zero. sMAPE is bounded in $[0, 200]\%$ but remains biased when target values are close to zero.

MASE (Mean Absolute Scaled Error)

MASE normalizes the MAE by the error of an in-sample naive forecast ($\hat{y}[n] = y[n-1]$):

$$\mathrm{MASE} = \frac{\frac{1}{L}\sum_{n} |e[n]|}{\frac{1}{L-1}\sum_{n=1}^{L-1} |y[n] - y[n-1]|}. \tag{19.5}$$

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.

NRMSE

NRMSE normalizes the RMSE by a measure of the target's spread:

$$\mathrm{NRMSE}_{\sigma} = \frac{\mathrm{RMSE}}{\sigma_y}, \qquad \mathrm{NRMSE}_{\mathrm{range}} = \frac{\mathrm{RMSE}}{\max y - \min y}. \tag{19.6}$$

Example 19.1: Two predictors are compared on the same target signal (Fig. 19.1). Predictor A has small, 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 multiple metrics provides a more complete picture of prediction quality.

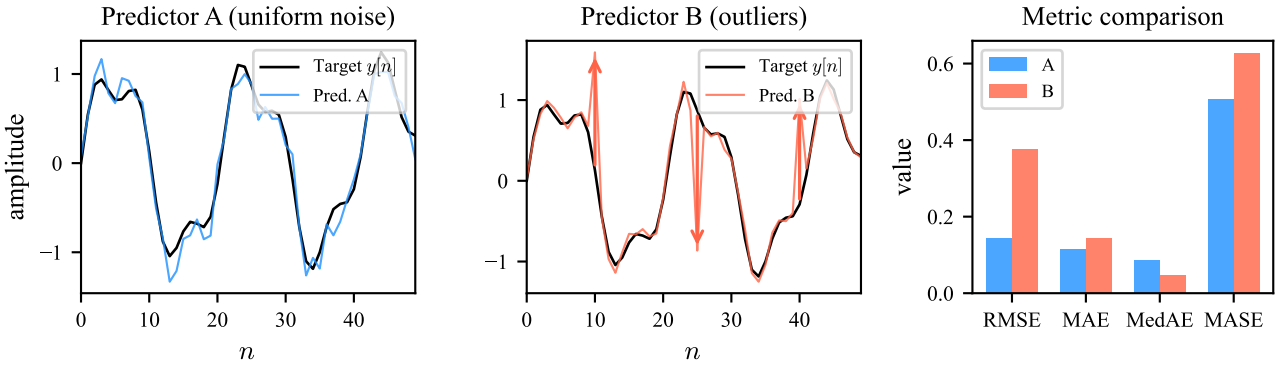


Figure 19.1.: Effect of outliers on metric values. Predictor A has uniform noise; Predictor B has sparse large errors. RMSE is sensitive to outliers, MAE is moderate, and MedAE is robust.

19.3. Information Criteria

Information criteria balance model fit against complexity to prevent overfitting. Given log-likelihood $\ln \mathcal{L}$, number of parameters k, and sample size L:

$$\mathrm{AIC} = 2k - 2\ln \mathcal{L}, \qquad \mathrm{BIC} = k \ln L - 2 \ln \mathcal{L}. \tag{19.7}$$

Lower values indicate a better trade-off. AIC favors predictive fit; BIC penalizes complexity more strongly, preferring simpler models. The corrected variant

$$\mathrm{AICc} = \mathrm{AIC} + \frac{2k(k+1)}{L-k-1} \tag{19.8}$$

adds a finite-sample correction and should be preferred when L/k is small.

19.4. Summary

Metric	**Scale-free**	**Outlier-robust**	**Also used as loss**
MSE / RMSE	No	No	Yes
MAE / MedAE	No	Yes	Yes (MAE)
MAPE / sMAPE	Yes	No	No
MASE	Yes	Yes	No
NRMSE	Yes	No	No

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

Datasets

L	Model complexity
N	Number of features
M	Number of entries in the dataset
K	Number of classes
$\mathbf{w}$ or w_i	Model parameters (vector form)
$f(\cdot;\mathbf{w})$	Model
$h(\mathbf{x})$ or $h(x)$	True unknown function
x_{ij}	Singe data value
$\mathbf{x}_i$	Singe data vector, i column number in $\mathbf{X}$
$\mathbf{X}$	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
$\mathbf{e}$	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

Signals

ω	Angular frequency (discrete)
θ	Phase angle
A	Amplitude
F	Frequency [Hz]
F_s	Sampling frequency
T	Period [sec]

Bibliography

[1] Tomas Andersson. *Selected topics in frequency estimation*. PhD thesis, KTH Royal Institute of Technology, 2003.

[2] Peter J Bickel and Kjell A Doksum. An analysis of transformations revisited. *Journal of the American Statistical Association*, 76(374):296–311, 1981.

[3] 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.

[4] Dima Bykhovsky. Experimental lognormal modeling of harmonics power of switched-mode power supplies. *Energies*, 15(2), 2022.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] Toni Giorgino. Computing and visualizing dynamic time warping alignments in r: the dtw package. *Journal of statistical Software*, 31:1–24, 2009.

[12] Monson H Hayes. *Statistical Digital Signal Processing and Modeling*. John Wiley & Sons, 1996.

[13] 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.

[14] Steven M. Kay. *Fundamentals of Statistical Signal Processing, Volume I: Estimation Theory*. Prentice Hall, 1993.

[15] 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.

[16] 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.

[17] NIST/SEMATECH. e-handbook of statistical methods. http://www.itl.nist.gov/div898/handbook/, Apr 2012. [Accessed 04-07-2024].

[18] Boaz Porat. *Digital processing of random signals: theory and methods*. Courier Dover Publications, 2008.

[19] 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.

[20] 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.