Machine Learning

Lecture Notes

L3–M1 — 2025–2026

Yaë Ulrich Gaba

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E." — Tom Mitchell

March 25, 2026

Contents

Preface

Machine learning is a discipline at the intersection of mathematics, computer science, and statistics. Its fundamental objective is to design algorithms capable of **learning from data**: instead of explicitly programming a decision rule, we let the algorithm discover this rule from examples.

Course Positioning

This course is at the L3/Master 1 level and assumes prerequisites in:

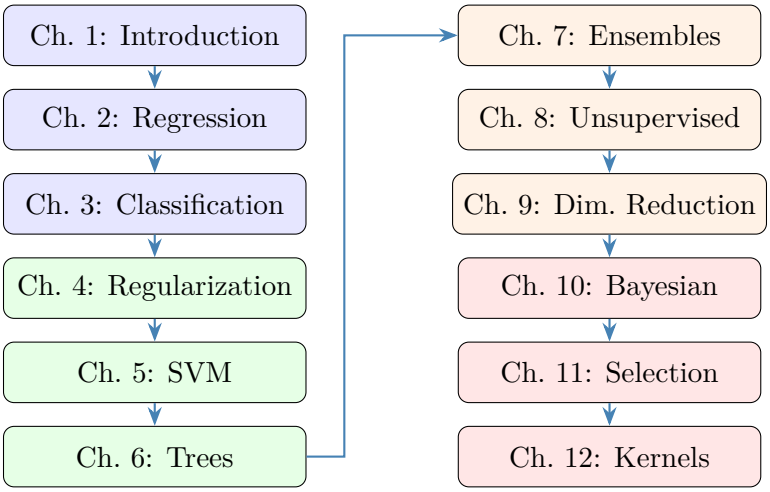
- **Linear algebra**: vector spaces, matrices, eigenvalues, singular value decomposition.
- **Probability and statistics**: random variables, expectation, variance, normal distribution, estimation, tests.
- **Analysis**: partial derivatives, gradient, optimization of multivariate functions.
- **Python programming**: NumPy, Matplotlib; knowledge of scikit-learn is a plus but not required.

Pedagogical Philosophy

Each chapter follows a three-step progression:

1. **Theory**: rigorous mathematical derivation of algorithms, with statement and proof of key results.
2. **Practice**: implementation in Python with scikit-learn and, when instructive, from scratch.
3. **Analysis**: discussion of strengths, weaknesses, assumptions, and use cases of each method.

Course Outline



Chapters 1–6 cover the foundations (linear and nonlinear supervised learning). Chapters 7–9 address ensemble methods and unsupervised learning. Chapters 10–12 treat advanced topics (Bayesian approach, model selection, kernel methods).

Happy reading and happy learning!

Chapter 1

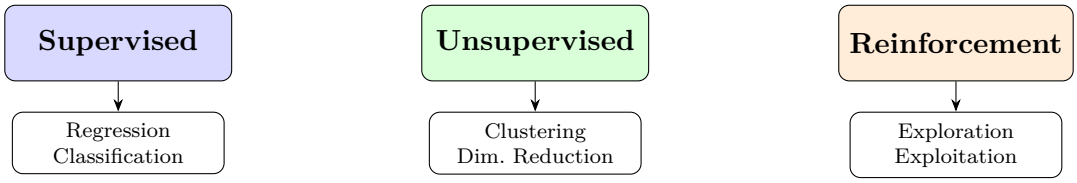
Introduction to Machine Learning

1.1 What is Machine Learning?

Definition 1.1 (Machine Learning)**. Machine learning** is the study of algorithms that automatically improve their performance on a given task through experience (data), without being explicitly programmed for that task [3].

More formally, following Tom Mitchell: a program learns from experience E with respect to a class of tasks T and a performance measure P, if its performance on T, as measured by P, improves with experience E.

1.2 Types of Learning



1.2.1 Supervised Learning

We have a dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ where $\boldsymbol{x}_i \in \mathbb{R}^d$ is a feature vector and y_i is the label. The goal is to learn a function $f : \mathbb{R}^d \to \mathcal{Y}$ such that $f(\boldsymbol{x}) \approx y$ for new data.

- **Regression**: $\mathcal{Y} = \mathbb{R}$ (apartment price, temperature).
- **Classification**: $\mathcal{Y} = \{1, \ldots, K\}$ (spam/not-spam, handwritten digits).

1.2.2 Unsupervised Learning

We have only $\mathcal{D} = \{\boldsymbol{x}_i\}_{i=1}^n$ without labels. The goal is to discover hidden structure: clusters, low-dimensional manifolds, distributions.

1.2.3 Reinforcement Learning

An agent interacts with an environment, receives rewards, and learns a policy maximizing cumulative reward. This paradigm is not covered in this course.

1.3 Mathematical Formalization

1.3.1 Hypothesis Space

Definition 1.2 (Hypothesis space)**.** The **hypothesis space** $\mathcal{H}$ is the set of functions $h : \mathbb{R}^d \to \mathcal{Y}$ among which the learning algorithm searches for the best one.

Example 1.3 (Linear regression)**.** $\mathcal{H} = \{h_{\boldsymbol{w}} : \boldsymbol{x} \mapsto \boldsymbol{w}^\top \boldsymbol{x} + b \mid \boldsymbol{w} \in \mathbb{R}^d, b \in \mathbb{R}\}$.

1.3.2 Loss Function and Risk

Definition 1.4 (Loss function)**.** A **loss function** $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ measures the error between the prediction $\hat{y} = h(\boldsymbol{x})$ and the true value y. Common examples:

$$\text{Squared error:} \quad \ell(y, \hat{y}) = (y - \hat{y})^2 \tag{1.1}$$

$$\text{0-1 error:} \quad \ell(y, \hat{y}) = \mathbb{1}[y \neq \hat{y}] \tag{1.2}$$

$$\text{Log-loss:} \quad \ell(y, \hat{y}) = -y \log \hat{y} - (1 - y) \log(1 - \hat{y}) \tag{1.3}$$

Definition 1.5 (Risk)**.** The **risk** (or generalization error) of a hypothesis h is:

$$R(h) = \mathbb{E}_{(\boldsymbol{x},y)\sim P}[\ell(y, h(\boldsymbol{x}))] \tag{1.4}$$

where P is the (unknown) joint distribution of the data.

Definition 1.6 (Empirical risk)**.** The **empirical risk** is the approximation of the risk on the training data:

$$\hat{R}_n(h) = \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, h(\boldsymbol{x}_i)) \tag{1.5}$$

Empirical Risk Minimization (ERM)

The learning algorithm seeks:

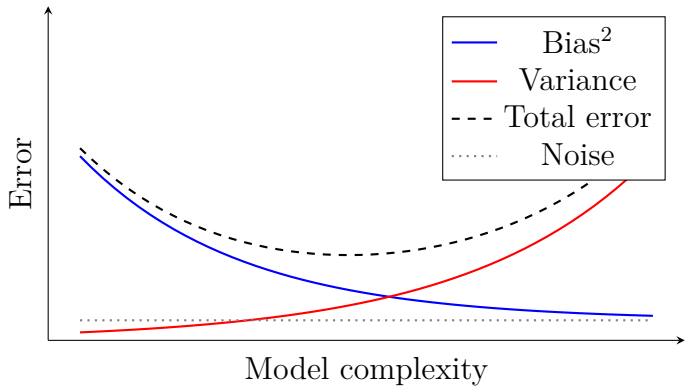
$$h^* = \arg\min_{h \in \mathcal{H}} \hat{R}_n(h) = \arg\min_{h \in \mathcal{H}} \frac{1}{n} \sum_{i=1}^{n} \ell(y_i, h(\boldsymbol{x}_i)) \tag{1.6}$$

1.4 The Bias-Variance Tradeoff

Theorem 1.7 (Bias-variance decomposition)**.** *For the squared loss, the risk decomposes as:*

$$\mathbb{E}[(y - \hat{f}(\boldsymbol{x}))^2] = \underbrace{\text{Bias}^2[\hat{f}(\boldsymbol{x})]}_{bias^2} + \underbrace{\text{Var}[\hat{f}(\boldsymbol{x})]}_{variance} + \underbrace{\sigma_\varepsilon^2}_{irreducible\ noise} \tag{1.7}$$

where $\text{Bias}[\hat{f}(\boldsymbol{x})] = \mathbb{E}[\hat{f}(\boldsymbol{x})] - f(\boldsymbol{x})$ *and* $\sigma_\varepsilon^2 = \text{Var}[\varepsilon]$.



Interpretation

- A model that is too simple (high bias) fails to capture the data structure: **underfitting**.
- A model that is too complex (high variance) fits the noise: **overfitting**.
- The goal is to find the optimal tradeoff.

1.5 Model Evaluation

1.5.1 Data Splitting

Train / Validation / Test

- **Training** (60–80 %): to fit the parameters.
- **Validation** (10–20 %): to choose hyperparameters.
- **Test** (10–20 %): to estimate final performance. **Never** used for model selection.

1.5.2 Cross-Validation

Definition 1.8 (k-fold cross-validation)**.** We partition the data into k subsets (folds). For each fold i:

1. Train on the remaining $k-1$ folds.
2. Evaluate on fold i.

The cross-validation error is the mean of the k errors:

$$\mathrm{CV}(k) = \frac{1}{k}\sum_{i=1}^{k} \hat{R}^{(i)} \tag{1.8}$$

1.5.3 Metrics

Common metrics

Regression:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2, \quad \text{RMSE} = \sqrt{\text{MSE}}, \quad R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2} \tag{1.9}$$

Classification:

$$\text{Precision} = \frac{TP}{TP + FP}, \quad \text{Recall} = \frac{TP}{TP + FN}, \quad F_1 = 2 \cdot \frac{\text{Prec.} \times \text{Recall}}{\text{Prec.} + \text{Recall}} \tag{1.10}$$

1.6 Implementation: First Model with scikit-learn

Iris classification with scikit-learn

```python
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report

# Load data
X, y = load_iris(return_X_y=True)

# Train/test split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# k-NN model
model = KNeighborsClassifier(n_neighbors=5)
model.fit(X_train, y_train)

# Evaluation
print(f"Test accuracy: {model.score(X_test, y_test):.4f}")
print(classification_report(y_test, model.predict(X_test)))

# 5-fold cross-validation
scores = cross_val_score(model, X, y, cv=5, scoring="accuracy")
print(f"CV accuracy: {scores.mean():.4f} +/- {scores.std():.4f}")
```

Output

```
Test accuracy: 1.0000
              precision    recall  f1-score   support
           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00        10
```

```
           2       1.00      1.00      1.00        10
    accuracy                           1.00        30
CV accuracy: 0.9667 +/- 0.0211
```

1.7 Exercises

Exercise 1.1 (Empirical risk)**.** Let $\mathcal{D} = \{(1, 2), (2, 3.5), (3, 6)\}$ and $h_w(x) = wx$.

1. Compute $\hat{R}_3(h_w)$ with the squared loss for $w = 1.5$ and $w = 2$.
2. Find w^* minimizing $\hat{R}_3(h_w)$ analytically.
3. Plot $\hat{R}_3(h_w)$ as a function of w.

Exercise 1.2 (Bias-variance decomposition)**.**

1. Show that $\mathbb{E}[(y-\hat{f})^2] = \text{Bias}^2 + \text{Var} + \sigma^2$, detailing each step.
2. Generate $n = 50$ points with $y = \sin(x) + \varepsilon$, $\varepsilon \sim \mathcal{N}(0, 0.3)$. Fit polynomials of degrees 1, 3, 5, 10, 15 and illustrate the bias-variance tradeoff.

Exercise 1.3 (Cross-validation)**.**

1. Implement k-fold cross-validation from scratch (without `sklearn.model_selection`).
2. Compare your implementation's results with `cross_val_score` for k-NN on the Iris dataset.
3. Study the effect of k (number of neighbours) on CV accuracy. Plot the accuracy vs k curve for $k \in \{1, 3, 5, 7, \ldots, 25\}$.

Chapter 2

Linear and Polynomial Regression

Chapter Overview

Regression is the cornerstone of supervised learning: given input features $\boldsymbol{x}$, predict a continuous target y. This chapter builds the theory from a single variable to arbitrary polynomial complexity, derives every estimator in closed form, and introduces gradient descent as a scalable alternative.

2.1 Simple Linear Regression

Definition 2.1 (Simple Linear Model)**.** Given a dataset $\{(x_i, y_i)\}_{i=1}^n$ with $x_i, y_i \in \mathbb{R}$, the simple linear regression model assumes

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i, \qquad \varepsilon_i \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma^2).$$

The parameters β_0 (intercept) and β_1 (slope) are unknown and must be estimated from data.

2.1.1 Ordinary Least Squares Derivation

We seek $\hat{\beta}_0, \hat{\beta}_1$ that minimise the residual sum of squares:

$$\text{RSS}(\beta_0, \beta_1) = \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i)^2.$$

Theorem 2.2 (OLS Estimators — Simple Case)**.** *Setting the partial derivatives to zero yields*

$$\hat{\beta}_1 = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2} = \frac{\widehat{\text{Cov}}(X, Y)}{\widehat{\text{Var}}(X)}, \qquad \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}.$$

Proof. Taking partial derivatives of RSS:

$$\frac{\partial \text{RSS}}{\partial \beta_0} = -2 \sum_{i=1}^{n} (y_i - \beta_0 - \beta_1 x_i) = 0 \implies n\beta_0 = \sum y_i - \beta_1 \sum x_i, \tag{2.1}$$

$$\frac{\partial \text{RSS}}{\partial \beta_1} = -2 \sum_{i=1}^{n} x_i (y_i - \beta_0 - \beta_1 x_i) = 0. \tag{2.2}$$

Substituting $\beta_0 = \bar{y} - \beta_1 \bar{x}$ from (2.1) into (2.2) and simplifying gives the stated formula for $\hat{\beta}_1$. □

2.1.2 Properties of the OLS Estimators

Proposition 2.3 (Unbiasedness)**.** Under the model assumptions, $\mathbb{E}[\hat{\beta}_0] = \beta_0$ and $\mathbb{E}[\hat{\beta}_1] = \beta_1$.

Proposition 2.4 (Variance of OLS Estimators)**.**

$$\mathrm{Var}(\hat{\beta}_1) = \frac{\sigma^2}{\sum_{i=1}^n (x_i - \bar{x})^2}, \qquad \mathrm{Var}(\hat{\beta}_0) = \sigma^2 \left(\frac{1}{n} + \frac{\bar{x}^2}{\sum_{i=1}^n (x_i - \bar{x})^2} \right).$$

An unbiased estimator of σ^2 is $\hat{\sigma}^2 = \frac{\mathrm{RSS}}{n-2}$.

Definition 2.5 (Coefficient of Determination)**.** The coefficient of determination R^2 measures the proportion of variance explained:

$$R^2 = 1 - \frac{\mathrm{RSS}}{\mathrm{TSS}} = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}, \qquad R^2 \in [0, 1].$$

Remark 2.6. In simple regression, $R^2 = r_{xy}^2$ where r_{xy} is the Pearson correlation coefficient. Adding more features can only increase R^2 (or leave it unchanged), which motivates the *adjusted* R^2:

$$R^2_{\mathrm{adj}} = 1 - \frac{n-1}{n-p-1}(1 - R^2).$$

2.2 Multiple Linear Regression

Definition 2.7 (Multiple Linear Model)**.** With p features the model becomes

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \qquad \boldsymbol{X} \in \mathbb{R}^{n \times (p+1)},\ \boldsymbol{\beta} \in \mathbb{R}^{p+1},\ \boldsymbol{\varepsilon} \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I}_n),$$

where the first column of $\boldsymbol{X}$ is the constant $\boldsymbol{1}_n$ (intercept term).

2.2.1 Normal Equations

Theorem 2.8 (OLS in Matrix Form)**.** *If $\boldsymbol{X}^\top \boldsymbol{X}$ is invertible, the unique minimiser of $\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2$ is*

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

Proof. Define $f(\boldsymbol{\beta}) = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^\top (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})$. Expanding:

$$f(\boldsymbol{\beta}) = \boldsymbol{y}^\top \boldsymbol{y} - 2\boldsymbol{\beta}^\top \boldsymbol{X}^\top \boldsymbol{y} + \boldsymbol{\beta}^\top \boldsymbol{X}^\top \boldsymbol{X} \boldsymbol{\beta}.$$

Taking the gradient and setting it to zero:

$$\nabla_{\boldsymbol{\beta}} f = -2\boldsymbol{X}^\top \boldsymbol{y} + 2\boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{\beta} = \boldsymbol{0} \implies \boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{\beta} = \boldsymbol{X}^\top \boldsymbol{y}.$$

Since $\boldsymbol{X}^\top \boldsymbol{X}$ is positive definite (hence invertible) when $\boldsymbol{X}$ has full column rank, we obtain the stated solution. The Hessian $2\boldsymbol{X}^\top \boldsymbol{X} \succeq 0$ confirms this is a minimum. □

Proposition 2.9 (Gauss–Markov Theorem)**.** Among all linear unbiased estimators of $\boldsymbol{\beta}$, the OLS estimator $\hat{\boldsymbol{\beta}}$ has the smallest variance (in the matrix sense). That is, for any other linear unbiased estimator $\tilde{\boldsymbol{\beta}}$,

$$\mathrm{Var}(\tilde{\boldsymbol{\beta}}) - \mathrm{Var}(\hat{\boldsymbol{\beta}}) \succeq 0 \quad \text{(positive semi-definite)}.$$

Proposition 2.10 (Distribution of OLS Estimator)**.** Under the Gaussian noise assumption:

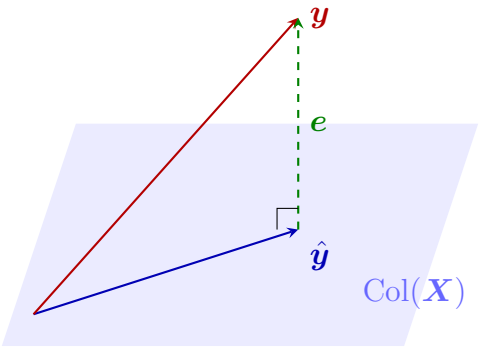
$$\hat{\boldsymbol{\beta}} \sim \mathcal{N}\big(\boldsymbol{\beta},\ \sigma^2(\boldsymbol{X}^\top \boldsymbol{X})^{-1}\big).$$

This allows construction of confidence intervals and hypothesis tests. A t-test for $H_0 : \beta_j = 0$ uses the statistic $t_j = \hat{\beta}_j / \mathrm{se}(\hat{\beta}_j)$ with $n - p - 1$ degrees of freedom.

2.2.2 Geometric Interpretation

Projection Viewpoint

The fitted values $\hat{\boldsymbol{y}} = \boldsymbol{X}\hat{\boldsymbol{\beta}} = \boldsymbol{H}\boldsymbol{y}$ are the orthogonal projection of $\boldsymbol{y}$ onto the column space of $\boldsymbol{X}$, where $\boldsymbol{H} = \boldsymbol{X}(\boldsymbol{X}^\top \boldsymbol{X})^{-1}\boldsymbol{X}^\top$ is the *hat matrix*. The residual vector $\boldsymbol{e} = \boldsymbol{y} - \hat{\boldsymbol{y}}$ is orthogonal to every column of $\boldsymbol{X}$.



2.3 Polynomial Regression

Definition 2.11 (Polynomial Regression of Degree d)**.** Given scalar input x, the degree-d polynomial model is

$$y = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_d x^d + \varepsilon.$$

This is a *linear* model in the extended feature vector $\boldsymbol{\phi}(x) = (1, x, x^2, \ldots, x^d)^\top$.

Overfitting Risk

High-degree polynomials can interpolate training data perfectly while generalising poorly. The bias–variance trade-off governs model selection: low d may underfit (high bias), high d may overfit (high variance). Cross-validation is essential.

Proposition 2.12 (Bias–Variance Decomposition)**.** For a fixed test point $\boldsymbol{x}_0$ and squared loss, the expected prediction error decomposes as

$$\mathbb{E}\big[(y_0 - \hat{f}(\boldsymbol{x}_0))^2\big] = \underbrace{\mathrm{Bias}^2\big(\hat{f}(\boldsymbol{x}_0)\big)}_{\text{systematic error}} + \underbrace{\mathrm{Var}\big(\hat{f}(\boldsymbol{x}_0)\big)}_{\text{estimation variance}} + \sigma^2.$$

Proof. Let $f_0 = f(\boldsymbol{x}_0)$ denote the true regression function. Then

$$\begin{aligned}\mathbb{E}[(y_0 - \hat{f})^2] &= \mathbb{E}[(y_0 - f_0 + f_0 - \mathbb{E}[\hat{f}] + \mathbb{E}[\hat{f}] - \hat{f})^2] \\ &= \mathbb{E}[(y_0 - f_0)^2] + (f_0 - \mathbb{E}[\hat{f}])^2 + \mathbb{E}[(\hat{f} - \mathbb{E}[\hat{f}])^2] \\ &= \sigma^2 + \text{Bias}^2(\hat{f}) + \text{Var}(\hat{f}),\end{aligned}$$

where cross-terms vanish by independence and the fact that $\mathbb{E}[\hat{f} - \mathbb{E}[\hat{f}]] = 0$. □

Example 2.13 (Choosing Polynomial Degree)**.** Consider fitting $y = \sin(2\pi x) + \varepsilon$ on $[0, 1]$:

- $d = 1$ (line): high bias, misses curvature.
- $d = 3$: reasonable compromise, captures the shape.
- $d = 15$: oscillates wildly between data points (Runge phenomenon).

Cross-validation typically selects $d \in \{3, 4, 5\}$ for this example.

2.4 Gradient Descent

Definition 2.14 (Gradient Descent Update)**.** For a differentiable loss $J(\boldsymbol{\beta})$ and learning rate $\eta > 0$:

$$\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} - \eta \nabla J(\boldsymbol{\beta}^{(t)}).$$

Batch Gradient Descent for Linear Regression

1. Initialise $\boldsymbol{\beta}^{(0)}$ (e.g. zeros).
2. At each iteration t, compute
$$\nabla J = \frac{2}{n} \boldsymbol{X}^\top (\boldsymbol{X}\boldsymbol{\beta}^{(t)} - \boldsymbol{y}).$$
3. Update: $\boldsymbol{\beta}^{(t+1)} \leftarrow \boldsymbol{\beta}^{(t)} - \eta \nabla J$.
4. Repeat until $\|\nabla J\| < \epsilon$ or maximum iterations reached.

Theorem 2.15 (Convergence of Gradient Descent)**.** *If J is L-smooth and convex, gradient descent with step size $\eta = 1/L$ satisfies*

$$J(\boldsymbol{\beta}^{(t)}) - J(\boldsymbol{\beta}^*) \leq \frac{L \left\|\boldsymbol{\beta}^{(0)} - \boldsymbol{\beta}^*\right\|^2}{2t}.$$

For linear regression, the smoothness constant is $L = \frac{2}{n}\lambda_{\max}(\boldsymbol{X}^\top \boldsymbol{X})$.

Definition 2.16 (Stochastic Gradient Descent)**.** At each iteration, pick a random index $i_t \in \{1, \ldots, n\}$ and update

$$\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} - \eta_t \nabla J_{i_t}(\boldsymbol{\beta}^{(t)}), \qquad \nabla J_i = 2((\boldsymbol{x}_i^\top \boldsymbol{\beta}^{(t)}) - y_i)\boldsymbol{x}_i.$$

Each step costs $O(p)$ instead of $O(np)$, but introduces variance in the gradient estimate.

Stochastic and Mini-Batch Variants

SGD: Replace the full gradient with a single-sample estimate ∇J_i. Convergence is noisier but each step costs $O(p)$ instead of $O(np)$. Use a decaying learning rate $\eta_t = \eta_0/t$.

Mini-batch: Use a random subset of size B. A good default is $B \in \{32, 64, 128\}$. This balances the speed of SGD with the stability of batch GD.

2.5 Implementation in Python

2.5.1 From Scratch

OLS from Scratch

```python
import numpy as np

def ols_fit(X, y):
    """Ordinary least squares via the normal equations."""
    # Add intercept column
    ones = np.ones((X.shape[0], 1))
    X_aug = np.hstack([ones, X])
    # Normal equations: beta = (X^T X)^{-1} X^T y
    beta = np.linalg.solve(X_aug.T @ X_aug, X_aug.T @ y)
    return beta

def ols_predict(X, beta):
    ones = np.ones((X.shape[0], 1))
    X_aug = np.hstack([ones, X])
    return X_aug @ beta

# Generate synthetic data
np.random.seed(42)
X = 2 * np.random.rand(100, 1)
y = 4 + 3 * X.ravel() + np.random.randn(100)

beta = ols_fit(X, y)
print(f"Intercept: {beta[0]:.3f}, Slope: {beta[1]:.3f}")
```

Output

```
Intercept: 4.215, Slope: 2.770
```

2.5.2 Gradient Descent from Scratch

Gradient Descent Implementation

```
def gradient_descent(X, y, eta=0.1, n_iter=1000):
    ones = np.ones((X.shape[0], 1))
    X_aug = np.hstack([ones, X])
    n, p = X_aug.shape
    beta = np.zeros(p)
    losses = []
    for t in range(n_iter):
        residuals = X_aug @ beta - y
        grad = (2 / n) * X_aug.T @ residuals
        beta -= eta * grad
        losses.append(np.mean(residuals**2))
    return beta, losses

beta_gd, loss_history = gradient_descent(X, y, eta=0.05, n_iter=2000)
print(f"GD Intercept: {beta_gd[0]:.3f}, GD Slope: {beta_gd[1]:.3f}")
print(f"Final MSE: {loss_history[-1]:.4f}")
```

Output

```
GD Intercept: 4.215, GD Slope: 2.770
Final MSE: 0.9225
```

2.5.3 Using Scikit-Learn

Linear and Polynomial Regression with scikit-learn

```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error, r2_score

# Simple linear regression
model = LinearRegression()
model.fit(X, y)
print(f"sklearn Intercept: {model.intercept_:.3f}")
print(f"sklearn Slope:     {model.coef_[0]:.3f}")
print(f"R^2:               {model.score(X, y):.4f}")

# Polynomial regression (degree 3)
poly_model = make_pipeline(
    PolynomialFeatures(degree=3, include_bias=False),
    LinearRegression()
)
poly_model.fit(X, y)
```

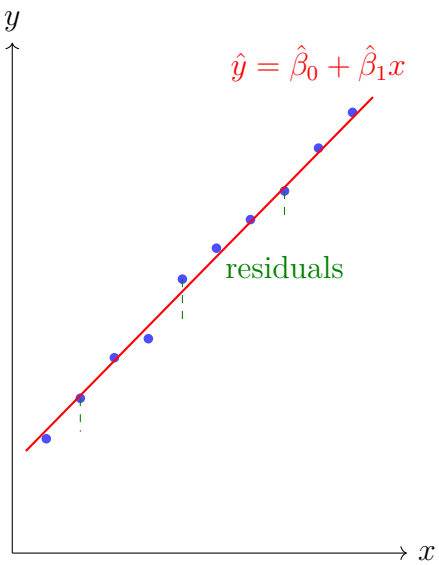
```
y_pred = poly_model.predict(X)
print(f"Poly3 MSE: {mean_squared_error(y, y_pred):.4f}")
print(f"Poly3 R^2: {r2_score(y, y_pred):.4f}")

# Cross-validation to select degree
for d in [1, 2, 3, 5, 10]:
    pipe = make_pipeline(
        PolynomialFeatures(degree=d, include_bias=False),
        LinearRegression()
    )
    cv_mse = -cross_val_score(pipe, X, y,
                              scoring="neg_mean_squared_error", cv=5)
    print(f"Degree {d:2d}: CV MSE = {cv_mse.mean():.4f} "
          f"(+/- {cv_mse.std():.4f})")
```

Output

```
sklearn Intercept: 4.215
sklearn Slope:     2.770
R^2:               0.8901
Poly3 MSE: 0.9102
Poly3 R^2: 0.8932
Degree  1: CV MSE = 1.0243 (+/- 0.2148)
Degree  2: CV MSE = 1.0067 (+/- 0.2314)
Degree  3: CV MSE = 1.0401 (+/- 0.2589)
Degree  5: CV MSE = 1.2315 (+/- 0.4012)
Degree 10: CV MSE = 5.8934 (+/- 8.1247)
```

2.6 Regression Diagram



2.7 Assumptions and Diagnostics

Definition 2.17 (Assumptions of Linear Regression)**.** The classical linear model relies on four key assumptions:

1. **Linearity:** $\mathbb{E}[y|\boldsymbol{x}] = \boldsymbol{x}^\top \boldsymbol{\beta}$.
2. **Independence:** the errors ε_i are mutually independent.
3. **Homoscedasticity:** $\mathrm{Var}(\varepsilon_i) = \sigma^2$ for all i.
4. **Normality:** $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$ (needed for exact inference, not for OLS consistency).

Diagnostic Plots

- **Residuals vs. fitted:** should show no pattern. A funnel shape indicates heteroscedasticity.
- **Q–Q plot:** residuals should lie on the 45° line if normality holds.
- **Scale-location:** $\sqrt{|\text{standardised residuals}|}$ vs. fitted values checks for non-constant variance.
- **Leverage plot:** identifies influential observations via hat values h_{ii} and Cook's distance D_i.

Definition 2.18 (Cook's Distance)**.** Cook's distance for observation i measures its influence on the fitted values:

$$D_i = \frac{(\hat{\boldsymbol{y}} - \hat{\boldsymbol{y}}_{(i)})^\top (\hat{\boldsymbol{y}} - \hat{\boldsymbol{y}}_{(i)})}{(p+1)\hat{\sigma}^2} = \frac{e_i^2}{(p+1)\hat{\sigma}^2} \cdot \frac{h_{ii}}{(1-h_{ii})^2},$$

where $\hat{\boldsymbol{y}}_{(i)}$ denotes predictions when observation i is removed. Points with $D_i > 1$ (or $D_i > 4/n$) deserve closer inspection.

2.8 Exercises

Exercise 2.1 (Derivation)**.** Starting from the RSS for simple linear regression, verify the second-order conditions: show that the Hessian matrix is positive definite, confirming that the OLS solution is indeed a minimum.

Exercise 2.2 (Normal Equations Complexity)**.** Show that solving the normal equations $\boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{\beta} = \boldsymbol{X}^\top \boldsymbol{y}$ via Cholesky decomposition costs $O(np^2 + p^3/3)$. When is gradient descent preferable?

Exercise 2.3 (Polynomial Overfitting)**.** Generate $n = 30$ points from $y = \sin(2\pi x) + \varepsilon$ with $x \sim \mathcal{U}(0,1)$ and $\varepsilon \sim \mathcal{N}(0, 0.2)$.

1. Fit polynomials of degrees $d \in \{1, 3, 5, 10, 20\}$.
2. Plot training and test MSE as a function of d.
3. Identify the best degree using 5-fold cross-validation.

Exercise 2.4 (Gradient Descent Convergence)**.** Implement gradient descent for linear regression. Experiment with learning rates $\eta \in \{0.001, 0.01, 0.1, 1.0\}$ and plot the loss curve $J(\boldsymbol{\beta}^{(t)})$ versus iteration t. Explain what happens when η is too large.

Exercise 2.5 (Hat Matrix Properties)**.** Prove the following properties of the hat matrix $\boldsymbol{H} = \boldsymbol{X}(\boldsymbol{X}^\top \boldsymbol{X})^{-1}\boldsymbol{X}^\top$:

1. $\boldsymbol{H}$ is symmetric and idempotent ($\boldsymbol{H}^2 = \boldsymbol{H}$).
2. $\operatorname{tr}(\boldsymbol{H}) = p + 1$.
3. $(\boldsymbol{I} - \boldsymbol{H})\boldsymbol{X} = \boldsymbol{0}$.
4. The eigenvalues of $\boldsymbol{H}$ are either 0 or 1.

Exercise 2.6 (Multi-Feature Regression)**.** Load the `diabetes` dataset from `sklearn.datasets`. Fit a multiple linear regression, report R^2 on an 80/20 train/test split, and interpret the three largest coefficients. Compute the adjusted R^2 and compare.

Exercise 2.7 (Comparing Normal Equations and GD)**.** On the `diabetes` dataset, compare the coefficients obtained via the normal equations (`np.linalg.solve`) with those from gradient descent (1000 iterations, $\eta = 0.01$). Plot $\left\|\boldsymbol{\beta}^{(t)} - \hat{\boldsymbol{\beta}}_{\text{OLS}}\right\|$ vs. t. How many iterations are needed to achieve $< 10^{-6}$ error?

Chapter 2 Summary

- **Simple OLS:** $\hat{\beta}_1 = \frac{\widehat{\operatorname{Cov}}(X,Y)}{\widehat{\operatorname{Var}}(X)}, \quad \hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$
- **Multiple OLS:** $\hat{\boldsymbol{\beta}} = (\boldsymbol{X}^\top \boldsymbol{X})^{-1}\boldsymbol{X}^\top \boldsymbol{y}$
- **Gradient Descent:** $\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} - \frac{2\eta}{n}\boldsymbol{X}^\top(\boldsymbol{X}\boldsymbol{\beta}^{(t)} - \boldsymbol{y})$
- **Polynomial:** Linear in $\boldsymbol{\phi}(x) = (1, x, \ldots, x^d)^\top$
- **Bias–Variance:** $\mathbb{E}[(y - \hat{f})^2] = \text{Bias}^2 + \text{Var} + \sigma^2$
- R^2**:** $1 - \text{RSS}/\text{TSS}, \quad R^2_{\text{adj}} = 1 - \frac{n-1}{n-p-1}(1 - R^2)$

CHAPTER 2. LINEAR AND POLYNOMIAL REGRESSION

Chapter 3

Classification — Logistic Regression, k-NN, Naive Bayes

Chapter Overview

Classification assigns discrete labels to inputs. This chapter covers three foundational classifiers—logistic regression (a discriminative probabilistic model), k-nearest neighbours (a non-parametric method), and Naive Bayes (a generative model)—together with the evaluation metrics needed to compare them.

3.1 Logistic Regression

3.1.1 The Sigmoid Function and Model Formulation

Definition 3.1 (Logistic / Sigmoid Function)**.** The sigmoid function $\sigma : \mathbb{R} \to (0, 1)$ is defined by

$$\sigma(z) = \frac{1}{1 + e^{-z}}, \qquad \sigma'(z) = \sigma(z)(1 - \sigma(z)).$$

Proposition 3.2 (Symmetry Property)**.** The sigmoid satisfies $\sigma(-z) = 1 - \sigma(z)$ for all $z \in \mathbb{R}$. Consequently, the function is symmetric about the point $\left(0, \frac{1}{2}\right)$.

Definition 3.3 (Binary Logistic Model)**.** For $y \in \{0, 1\}$ and feature vector $\boldsymbol{x} \in \mathbb{R}^p$:

$$P(Y = 1 \mid \boldsymbol{x}; \boldsymbol{\beta}) = \sigma(\boldsymbol{\beta}^\top \boldsymbol{x}) = \frac{1}{1 + \exp(-\boldsymbol{\beta}^\top \boldsymbol{x})},$$

where $\boldsymbol{x}$ includes a constant 1 for the intercept. Equivalently, the log-odds (logit) are linear:

$$\log \frac{P(Y = 1|\boldsymbol{x})}{P(Y = 0|\boldsymbol{x})} = \boldsymbol{\beta}^\top \boldsymbol{x}.$$

Remark 3.4. Unlike linear regression, logistic regression does not model y directly but the *probability* that $y = 1$. The sigmoid ensures outputs lie in $(0, 1)$. A unit increase in x_j multiplies the odds by e^{β_j}.

3.1.2 Maximum Likelihood Estimation

Theorem 3.5 (Log-Likelihood for Logistic Regression)**.** *Given* $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ *with* $y_i \in \{0, 1\}$*, the log-likelihood is*

$$\ell(\boldsymbol{\beta}) = \sum_{i=1}^{n} \big[y_i \log \sigma(\boldsymbol{\beta}^\top \boldsymbol{x}_i) + (1 - y_i) \log(1 - \sigma(\boldsymbol{\beta}^\top \boldsymbol{x}_i)) \big].$$

This function is concave, so any local maximum is global.

Proof. Since $y_i \sim \text{Bernoulli}(\sigma(\boldsymbol{\beta}^\top \boldsymbol{x}_i))$, the likelihood is $\prod_i \sigma_i^{y_i}(1-\sigma_i)^{1-y_i}$. Taking the logarithm gives the stated expression. Concavity follows from the Hessian $\nabla^2 \ell = -\boldsymbol{X}^\top \boldsymbol{W} \boldsymbol{X}$ where $\boldsymbol{W} = \text{diag}(\sigma_i(1-\sigma_i)) \succ 0$, hence $\nabla^2 \ell \preceq 0$. □

Definition 3.6 (Binary Cross-Entropy Loss)**.** Minimising the negative log-likelihood defines the cross-entropy loss:

$$J(\boldsymbol{\beta}) = -\frac{1}{n} \sum_{i=1}^{n} \big[y_i \log \hat{p}_i + (1 - y_i) \log(1 - \hat{p}_i) \big], \qquad \hat{p}_i = \sigma(\boldsymbol{\beta}^\top \boldsymbol{x}_i).$$

3.1.3 Gradient and Update Rule

Proposition 3.7 (Gradient of the Cross-Entropy)**.**

$$\nabla_{\boldsymbol{\beta}} J = \frac{1}{n} \sum_{i=1}^{n} (\hat{p}_i - y_i)\, \boldsymbol{x}_i = \frac{1}{n} \boldsymbol{X}^\top (\hat{\boldsymbol{p}} - \boldsymbol{y}).$$

This has the same form as the linear regression gradient, but with $\hat{\boldsymbol{p}} = \sigma(\boldsymbol{X}\boldsymbol{\beta})$ instead of $\boldsymbol{X}\boldsymbol{\beta}$.

Gradient Descent for Logistic Regression

1. Initialise $\boldsymbol{\beta}^{(0)} = \boldsymbol{0}$.
2. At iteration t: compute $\hat{\boldsymbol{p}} = \sigma(\boldsymbol{X}\boldsymbol{\beta}^{(t)})$.
3. Gradient: $\boldsymbol{g} = \frac{1}{n} \boldsymbol{X}^\top (\hat{\boldsymbol{p}} - \boldsymbol{y})$.
4. Update: $\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} - \eta\, \boldsymbol{g}$.
5. Repeat until convergence ($\|\boldsymbol{g}\| < \epsilon$ or max iterations).

Remark 3.8. Newton's method (IRLS) converges faster by using the Hessian:

$$\boldsymbol{\beta}^{(t+1)} = \boldsymbol{\beta}^{(t)} + (\boldsymbol{X}^\top \boldsymbol{W}^{(t)} \boldsymbol{X})^{-1} \boldsymbol{X}^\top (\boldsymbol{y} - \hat{\boldsymbol{p}}^{(t)}),$$

where $\boldsymbol{W}^{(t)} = \text{diag}(\hat{p}_i^{(t)}(1 - \hat{p}_i^{(t)}))$. This typically converges in 5–10 iterations but costs $O(p^2 n + p^3)$ per step.

3.1.4 Softmax Multiclass Extension

Definition 3.9 (Softmax Regression)**.** For K classes, define K weight vectors $\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_K$. The probability of class k is

$$P(Y = k \mid \boldsymbol{x}) = \frac{\exp(\boldsymbol{\beta}_k^\top \boldsymbol{x})}{\sum_{j=1}^{K} \exp(\boldsymbol{\beta}_j^\top \boldsymbol{x})}.$$

The loss is the multiclass cross-entropy:

$$J = -\frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{K} \mathbb{1}\{y_i = k\} \log P(Y = k \mid \boldsymbol{x}_i).$$

Proposition 3.10 (Softmax Gradient)**.** The gradient with respect to weight vector $\boldsymbol{\beta}_k$ is

$$\nabla_{\boldsymbol{\beta}_k} J = \frac{1}{n} \sum_{i=1}^{n} (P(Y = k|\boldsymbol{x}_i) - \mathbb{1}\{y_i = k\})\, \boldsymbol{x}_i.$$

3.2 k-Nearest Neighbours

Definition 3.11 (k-NN Classifier)**.** Given a query point $\boldsymbol{x}_0$, let $\mathcal{N}_k(\boldsymbol{x}_0)$ denote its k nearest neighbours in the training set (under a chosen distance, typically Euclidean). The predicted class is

$$\hat{y} = \arg\max_{c} \sum_{i \in \mathcal{N}_k(\boldsymbol{x}_0)} \mathbb{1}\{y_i = c\}.$$

Definition 3.12 (Common Distance Metrics)**.**

- **Euclidean:** $d(\boldsymbol{x}, \boldsymbol{x}') = \|\boldsymbol{x} - \boldsymbol{x}'\|_2 = \sqrt{\sum_j (x_j - x'_j)^2}$.
- **Manhattan:** $d(\boldsymbol{x}, \boldsymbol{x}') = \|\boldsymbol{x} - \boldsymbol{x}'\|_1 = \sum_j |x_j - x'_j|$.
- **Minkowski:** $d(\boldsymbol{x}, \boldsymbol{x}') = \|\boldsymbol{x} - \boldsymbol{x}'\|_p = \left(\sum_j |x_j - x'_j|^p\right)^{1/p}$.

Proposition 3.13 (Asymptotic Optimality)**.** As $n \to \infty$ and $k/n \to 0$ with $k \to \infty$, the k-NN classifier converges to the Bayes-optimal classifier. For $k = 1$, the asymptotic error rate is at most twice the Bayes error.

Curse of Dimensionality

In high dimensions, distances concentrate: for $\boldsymbol{x} \sim \mathcal{U}([0,1]^p)$, the ratio of the distance to the nearest neighbour to the farthest neighbour tends to 1 as $p \to \infty$. This makes k-NN unreliable without dimensionality reduction. Feature scaling is critical.

Choosing k

- Small k (e.g. $k = 1$): low bias, high variance (noisy boundaries).
- Large k: high bias, low variance (smooth boundaries).

- Use odd k for binary classification to avoid ties.
- Select k via cross-validation, typically $k \in \{3, 5, 7, \ldots\}$.
- k-NN has no training phase (lazy learner), but prediction costs $O(np)$ per query (or $O(p \log n)$ with KD-trees).

3.3 Naive Bayes

Definition 3.14 (Naive Bayes Classifier)**.** Applying Bayes' theorem with the conditional independence assumption $P(\boldsymbol{x} \mid y) = \prod_{j=1}^{p} P(x_j \mid y)$:

$$\hat{y} = \arg\max_c \; P(Y = c) \prod_{j=1}^{p} P(x_j \mid Y = c).$$

Remark 3.15. The "naive" independence assumption is rarely true, yet Naive Bayes often performs surprisingly well. This is because classification only requires the correct *ranking* of posterior probabilities, not their exact values. Even if $P(Y = c|\boldsymbol{x})$ is poorly estimated, the arg max may still be correct.

3.3.1 Gaussian Naive Bayes

Theorem 3.16 (Gaussian NB Decision Rule)**.** *Assume $X_j \mid Y = c \sim \mathcal{N}(\mu_{jc}, \sigma_{jc}^2)$. The log-posterior (up to a constant) is*

$$\log P(Y = c \mid \boldsymbol{x}) \propto \log \pi_c - \sum_{j=1}^{p} \left[\frac{(x_j - \mu_{jc})^2}{2\sigma_{jc}^2} + \log \sigma_{jc} \right],$$

where $\pi_c = P(Y = c)$ is the class prior. Maximum likelihood estimates are $\hat{\mu}_{jc} = \frac{1}{n_c} \sum_{i:y_i=c} x_{ij}$ and $\hat{\sigma}_{jc}^2 = \frac{1}{n_c} \sum_{i:y_i=c} (x_{ij} - \hat{\mu}_{jc})^2$.

Proof. By Bayes' theorem, $P(Y = c|\boldsymbol{x}) \propto P(Y = c) \prod_j P(x_j|Y = c)$. Taking logarithms and substituting the Gaussian density $\frac{1}{\sqrt{2\pi}\sigma_{jc}} \exp\!\left(-\frac{(x_j-\mu_{jc})^2}{2\sigma_{jc}^2}\right)$ for each factor, then dropping terms constant across classes, yields the stated discriminant function. □

Proposition 3.17 (Linear vs. Quadratic Boundaries)**.** When all σ_{jc} are equal across classes, the Gaussian NB decision boundary is linear (equivalent to LDA under independence). Unequal variances yield quadratic boundaries, analogous to QDA.

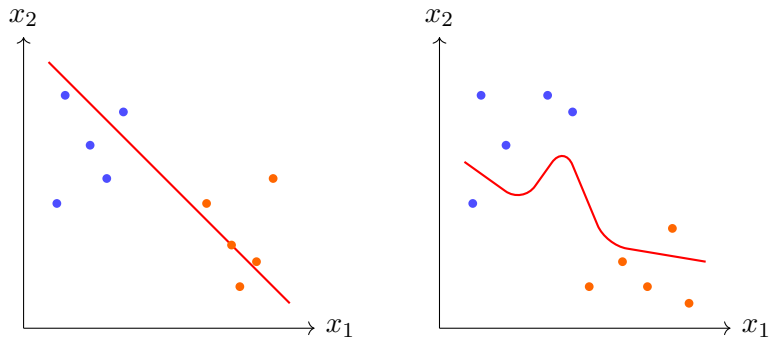
3.3.2 Other Naive Bayes Variants

Definition 3.18 (Multinomial and Bernoulli NB)**.**

- **Multinomial NB:** for count data (e.g. word frequencies in text). Assumes $P(\boldsymbol{x}|Y = c) \propto \prod_j \theta_{jc}^{x_j}$.
- **Bernoulli NB:** for binary features. Assumes $P(x_j|Y = c) = \theta_{jc}^{x_j}(1 - \theta_{jc})^{1-x_j}$.

Laplace smoothing ($\alpha = 1$) prevents zero probabilities: $\hat{\theta}_{jc} = \frac{n_{jc}+\alpha}{n_c+\alpha V}$ where V is the vocabulary size.

3.4 Decision Boundaries



Logistic Regression k-NN ($k = 3$)

3.5 Evaluation Metrics

Definition 3.19 (Confusion Matrix)**.** For a binary classifier with predictions $\hat{y}_i$ and true labels y_i:

	$\hat{y} = 1$	$\hat{y} = 0$
$y = 1$	TP	FN
$y = 0$	FP	TN

Definition 3.20 (Precision, Recall, F1-Score)**.**

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \qquad \text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \qquad F_1 = \frac{2 \cdot \text{Prec} \cdot \text{Rec}}{\text{Prec} + \text{Rec}}.$$

Proposition 3.21 (F-beta Score)**.** The general F_β score weights recall β times more than precision:

$$F_\beta = (1 + \beta^2) \frac{\text{Prec} \cdot \text{Rec}}{\beta^2 \cdot \text{Prec} + \text{Rec}}.$$

F_2 emphasises recall (useful when missing positives is costly); $F_{0.5}$ emphasises precision.

Definition 3.22 (ROC Curve and AUC)**.** The *Receiver Operating Characteristic* (ROC) curve plots the True Positive Rate (Recall) against the False Positive Rate $\text{FPR} = \frac{\text{FP}}{\text{FP}+\text{TN}}$ as the classification threshold varies from 0 to 1. The *Area Under the Curve* (AUC) summarises discriminative power:

$$\text{AUC} = P(\hat{p}_{Y=1} > \hat{p}_{Y=0}),$$

i.e., the probability that a randomly chosen positive is scored higher than a randomly chosen negative.

Proposition 3.23 (AUC Interpretation)**.** AUC $= 0.5$ corresponds to a random classifier, AUC $= 1.0$ to a perfect classifier. AUC is invariant to the classification threshold and to class prevalence. For imbalanced datasets, the *precision–recall AUC* (PR-AUC) is often more informative.

Accuracy Is Not Enough

On imbalanced datasets (e.g. 95% negatives), a classifier that always predicts negative achieves 95% accuracy. Precision, recall, F1, and AUC give a more faithful picture of performance in such settings.

3.6 Implementation in Python

3.6.1 Logistic Regression from Scratch

Logistic Regression — Gradient Descent

```
import numpy as np

def sigmoid(z):
    return 1 / (1 + np.exp(-np.clip(z, -500, 500)))

def logistic_fit(X, y, eta=0.1, n_iter=1000):
    n, p = X.shape
    X_aug = np.column_stack([np.ones(n), X])
    beta = np.zeros(X_aug.shape[1])
    for _ in range(n_iter):
        p_hat = sigmoid(X_aug @ beta)
        grad = (1 / n) * X_aug.T @ (p_hat - y)
        beta -= eta * grad
    return beta

def logistic_predict(X, beta, threshold=0.5):
    X_aug = np.column_stack([np.ones(X.shape[0]), X])
    return (sigmoid(X_aug @ beta) >= threshold).astype(int)
```

3.6.2 Scikit-Learn Classifiers

Logistic Regression, k-NN, Naive Bayes with scikit-learn

```
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import (accuracy_score, classification_report,
                             roc_auc_score, confusion_matrix)

X, y = load_iris(return_X_y=True)
# Binary task: versicolor vs. virginica
mask = y != 0
X, y = X[mask], (y[mask] - 1)  # labels 0 and 1
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42)
scaler = StandardScaler().fit(X_train)
X_tr = scaler.transform(X_train)
X_te = scaler.transform(X_test)

models = {
    "Logistic": LogisticRegression(max_iter=500),
    "k-NN (k=5)": KNeighborsClassifier(n_neighbors=5),
    "Gaussian NB": GaussianNB(),
}
for name, clf in models.items():
    clf.fit(X_tr, y_train)
    y_pred = clf.predict(X_te)
    acc = accuracy_score(y_test, y_pred)
    print(f"{name:15s}  Accuracy: {acc:.3f}")
```

Output

```
Logistic         Accuracy: 0.967
k-NN (k=5)       Accuracy: 0.933
Gaussian NB      Accuracy: 0.967
```

ROC Curve and Confusion Matrix

```
from sklearn.metrics import RocCurveDisplay, ConfusionMatrixDisplay
import matplotlib.pyplot as plt

fig, axes = plt.subplots(1, 2, figsize=(10, 4))

# ROC curve for logistic regression
lr = models["Logistic"]
RocCurveDisplay.from_estimator(lr, X_te, y_test, ax=axes[0])
axes[0].set_title("ROC Curve -- Logistic Regression")

# Confusion matrix
ConfusionMatrixDisplay.from_estimator(
    lr, X_te, y_test, ax=axes[1], cmap="Blues")
axes[1].set_title("Confusion Matrix")

plt.tight_layout()
plt.savefig("roc_cm.pdf")
```

Cross-Validation for k-NN

```
from sklearn.model_selection import cross_val_score
import numpy as np
```

```python
k_values = [1, 3, 5, 7, 9, 15, 25]
cv_scores = []
for k in k_values:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, X_tr, y_train, cv=5,
                             scoring="accuracy")
    cv_scores.append(scores.mean())
    print(f"k={k:2d}: CV accuracy = {scores.mean():.3f} "
          f"(+/- {scores.std():.3f})")

best_k = k_values[np.argmax(cv_scores)]
print(f"\nBest k = {best_k}")
```

Output

```
k= 1: CV accuracy = 0.914 (+/- 0.049)
k= 3: CV accuracy = 0.929 (+/- 0.048)
k= 5: CV accuracy = 0.943 (+/- 0.040)
k= 7: CV accuracy = 0.943 (+/- 0.040)
k= 9: CV accuracy = 0.929 (+/- 0.048)
k=15: CV accuracy = 0.914 (+/- 0.049)
k=25: CV accuracy = 0.900 (+/- 0.056)

Best k = 5
```

3.7 Exercises

Exercise 3.1 (Sigmoid Properties)**.** Prove that $\sigma(-z) = 1 - \sigma(z)$ and that $\sigma'(z) = \sigma(z)(1 - \sigma(z))$. Derive the Hessian of the binary cross-entropy and show it is positive semi-definite.

Exercise 3.2 (Decision Boundary Geometry)**.** For a binary logistic regression with two features (x_1, x_2), show that the decision boundary $\{x : P(Y = 1|\boldsymbol{x}) = 0.5\}$ is a line in $\mathbb{R}^2$. What determines its slope and intercept?

Exercise 3.3 (k-NN Experiments)**.** Using the `make_moons` dataset from scikit-learn:

1. Fit k-NN for $k \in \{1, 3, 5, 15, 50\}$ and plot decision boundaries.
2. Report 5-fold cross-validation accuracy for each k.
3. Discuss the bias–variance trade-off as k varies.

Exercise 3.4 (Naive Bayes Independence Assumption)**.** Consider two features X_1, X_2 with $\text{Cov}(X_1, X_2|Y = c) \neq 0$. Explain why Gaussian NB may still perform well in practice despite the violated independence assumption. Under what conditions does it fail?

Exercise 3.5 (Multiclass Softmax)**.** Derive the gradient $\nabla_{\boldsymbol{\beta}_k} J$ of the multiclass cross-entropy for softmax regression. Implement softmax regression from scratch on the Iris dataset (3 classes) and compare with `LogisticRegression(multi_class='multinomial')`.

Exercise 3.6 (ROC Analysis)**.** On a dataset of your choice, train logistic regression and Gaussian NB. Plot both ROC curves on the same axes and compare AUC scores. When does Naive Bayes outperform logistic regression?

Exercise 3.7 (Regularized Logistic Regression)**.** scikit-learn's `LogisticRegression` applies L2 regularization by default (parameter `C` = inverse strength). Using the breast cancer dataset, plot test accuracy as a function of $C \in \{10^{-3}, 10^{-2}, \ldots, 10^{3}\}$. Which C gives the best performance? Why is regularization important in logistic regression?

Chapter 3 Summary

- **Logistic:** $P(Y=1|\boldsymbol{x}) = \sigma(\boldsymbol{\beta}^\top \boldsymbol{x}), \quad \nabla J = \frac{1}{n}\boldsymbol{X}^\top(\hat{\boldsymbol{p}} - \boldsymbol{y})$
- **Softmax:** $P(Y=k|\boldsymbol{x}) = \frac{\exp(\boldsymbol{\beta}_k^\top \boldsymbol{x})}{\sum_j \exp(\boldsymbol{\beta}_j^\top \boldsymbol{x})}$
- **k-NN:** $\hat{y} = \arg\max_c \sum_{i \in \mathcal{N}_k} \mathbb{1}\{y_i = c\}$
- **Gaussian NB:** $\hat{y} = \arg\max_c \log \pi_c - \sum_j \frac{(x_j - \mu_{jc})^2}{2\sigma_{jc}^2}$
- **F1:** $\frac{2 \cdot \text{Prec} \cdot \text{Rec}}{\text{Prec} + \text{Rec}}$, **AUC:** area under ROC curve

CHAPTER 3. CLASSIFICATION – LOGISTIC REGRESSION, K-NN, NAIVE BAYES

Chapter 4

Regularization — Ridge, Lasso, Elastic Net

Chapter Overview

When the number of features is large relative to the sample size, or when features are correlated, OLS overfits and its estimates become unstable. Regularization adds a penalty to the loss function, shrinking coefficients and improving generalisation. This chapter derives Ridge, Lasso, and Elastic Net, and provides geometric, Bayesian, and computational perspectives.

4.1 The Overfitting Problem

Definition 4.1 (Overfitting)**.** A model *overfits* when it achieves low training error but high test error. Formally, if $\hat{f}$ is learned from a training set $\mathcal{D}$, overfitting occurs when

$$\mathbb{E}_{\mathcal{D}}\big[\mathrm{Err}_{\text{test}}(\hat{f})\big] \gg \mathrm{Err}_{\text{train}}(\hat{f}).$$

Proposition 4.2 (Variance of OLS Estimator)**.** Under the linear model $\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}$ with $\mathrm{Var}(\boldsymbol{\varepsilon}) = \sigma^2\boldsymbol{I}$:

$$\mathrm{Var}(\hat{\boldsymbol{\beta}}_{\text{OLS}}) = \sigma^2(\boldsymbol{X}^\top\boldsymbol{X})^{-1}.$$

When $\boldsymbol{X}^\top\boldsymbol{X}$ has small eigenvalues (near-collinearity), the diagonal entries of $(\boldsymbol{X}^\top\boldsymbol{X})^{-1}$ become large, inflating variance.

Multicollinearity

If two features are highly correlated, the OLS solution is not unique in the limit. Even moderate collinearity inflates standard errors, making individual coefficients unreliable. The *Variance Inflation Factor* $\mathrm{VIF}_j = \frac{1}{1-R_j^2}$ (where R_j^2 is the R^2 from regressing x_j on all other features) quantifies this; $\mathrm{VIF} > 10$ is a common concern threshold. Regularization stabilises the solution by adding a positive quantity to the diagonal of $\boldsymbol{X}^\top\boldsymbol{X}$.

Proposition 4.3 (MSE Decomposition for Estimators)**.** For any estimator $\tilde{\boldsymbol{\beta}}$ of $\boldsymbol{\beta}$:

$$\mathrm{MSE}(\tilde{\boldsymbol{\beta}}) = \mathbb{E}[\left\|\tilde{\boldsymbol{\beta}} - \boldsymbol{\beta}\right\|^2] = \mathrm{tr}(\mathrm{Var}(\tilde{\boldsymbol{\beta}})) + \left\|\mathrm{Bias}(\tilde{\boldsymbol{\beta}})\right\|^2.$$

A biased estimator can have lower MSE than OLS if it sufficiently reduces variance.

4.2 Ridge Regression (L2 Penalty)

Definition 4.4 (Ridge Regression)**.** The Ridge estimator minimises the penalised objective

$$J_{\text{Ridge}}(\boldsymbol{\beta}) = \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2 + \lambda \|\boldsymbol{\beta}\|_2^2 = \sum_{i=1}^{n}(y_i - \boldsymbol{x}_i^\top \boldsymbol{\beta})^2 + \lambda \sum_{j=1}^{p} \beta_j^2,$$

where $\lambda \geq 0$ is the regularization strength. Note: the intercept β_0 is typically *not* penalised; features should be centred and standardised.

Theorem 4.5 (Ridge Closed-Form Solution)**.** *The Ridge estimator has the unique closed-form solution*

$$\hat{\boldsymbol{\beta}}_{Ridge} = (\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I}_p)^{-1} \boldsymbol{X}^\top \boldsymbol{y}.$$

The matrix $\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I}_p$ is always invertible for $\lambda > 0$.

Proof. Taking the gradient of J_{Ridge}:

$$\nabla_{\boldsymbol{\beta}} J = -2\boldsymbol{X}^\top \boldsymbol{y} + 2\boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{\beta} + 2\lambda\boldsymbol{\beta} = \boldsymbol{0} \implies (\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I})\boldsymbol{\beta} = \boldsymbol{X}^\top \boldsymbol{y}.$$

For $\lambda > 0$, $\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I}$ is strictly positive definite (its eigenvalues are $d_j^2 + \lambda > 0$), hence invertible. □

Proposition 4.6 (SVD Interpretation of Ridge)**.** Let $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top$ be the SVD. Then

$$\hat{\boldsymbol{y}}_{\text{Ridge}} = \boldsymbol{X}\hat{\boldsymbol{\beta}}_{\text{Ridge}} = \sum_{j=1}^{p} \boldsymbol{u}_j \frac{d_j^2}{d_j^2 + \lambda} \boldsymbol{u}_j^\top \boldsymbol{y}.$$

Ridge shrinks the contribution of principal components with small singular values d_j. The *effective degrees of freedom* of Ridge regression are $\text{df}(\lambda) = \sum_j \frac{d_j^2}{d_j^2 + \lambda} \leq p$.

Theorem 4.7 (Bias–Variance of Ridge)**.** *The Ridge estimator has bias* $\text{Bias}(\hat{\boldsymbol{\beta}}_{Ridge}) = -\lambda(\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I})^{-1}\boldsymbol{\beta}$ *and variance*

$$\text{Var}(\hat{\boldsymbol{\beta}}_{Ridge}) = \sigma^2 (\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I})^{-1} \boldsymbol{X}^\top \boldsymbol{X} (\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I})^{-1}.$$

Increasing λ increases bias but decreases variance. There exists an optimal λ^* minimising the total MSE.

Proposition 4.8 (Ridge Shrinkage Factors)**.** For orthonormal design ($\boldsymbol{X}^\top \boldsymbol{X} = \boldsymbol{I}$), Ridge applies uniform shrinkage:

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

All coefficients are shrunk toward zero by the same factor $1/(1 + \lambda)$, but none are set exactly to zero.

4.2.1 Bayesian Interpretation

Proposition 4.9 (Ridge as MAP with Gaussian Prior)**.** If we place a Gaussian prior $\boldsymbol{\beta} \sim \mathcal{N}(\mathbf{0}, \tau^2 \boldsymbol{I})$ and assume $\boldsymbol{y}|\boldsymbol{X}, \boldsymbol{\beta} \sim \mathcal{N}(\boldsymbol{X}\boldsymbol{\beta}, \sigma^2 \boldsymbol{I})$, the MAP estimate equals the Ridge estimator with $\lambda = \sigma^2/\tau^2$.

Proof. The posterior is $P(\boldsymbol{\beta}|\boldsymbol{y}) \propto P(\boldsymbol{y}|\boldsymbol{\beta})P(\boldsymbol{\beta})$. Taking $-\log$:

$$-\log P(\boldsymbol{\beta}|\boldsymbol{y}) = \frac{1}{2\sigma^2} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2 + \frac{1}{2\tau^2} \|\boldsymbol{\beta}\|^2 + C.$$

Minimising is equivalent to minimising $\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2 + \frac{\sigma^2}{\tau^2} \|\boldsymbol{\beta}\|^2$, which is Ridge with $\lambda = \sigma^2/\tau^2$. □

Bayesian Perspective

A small prior variance τ^2 encodes strong belief that $\boldsymbol{\beta}$ is near zero, corresponding to large λ (heavy regularization). A large τ^2 (vague prior) gives small λ, approaching OLS. The full Bayesian posterior $P(\boldsymbol{\beta}|\boldsymbol{y})$ is Gaussian with mean $\hat{\boldsymbol{\beta}}_{\text{Ridge}}$ and covariance $\sigma^2(\boldsymbol{X}^\top \boldsymbol{X} + \lambda \boldsymbol{I})^{-1}$.

4.3 Lasso Regression (L1 Penalty)

Definition 4.10 (Lasso Regression)**.** The Lasso (Least Absolute Shrinkage and Selection Operator) minimises

$$J_{\text{Lasso}}(\boldsymbol{\beta}) = \frac{1}{2n} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2 + \lambda \|\boldsymbol{\beta}\|_1 = \frac{1}{2n} \sum_{i=1}^{n} (y_i - \boldsymbol{x}_i^\top \boldsymbol{\beta})^2 + \lambda \sum_{j=1}^{p} |\beta_j|.$$

Theorem 4.11 (Sparsity of Lasso Solutions)**.** *For sufficiently large λ, the Lasso sets some coefficients exactly to zero. Specifically, for the orthonormal design $\boldsymbol{X}^\top \boldsymbol{X} = \boldsymbol{I}$:*

$$\hat{\beta}_j^{Lasso} = \text{sign}(\hat{\beta}_j^{OLS}) \big(|\hat{\beta}_j^{OLS}| - \lambda\big)_+,$$

which is the soft-thresholding *operator $S_\lambda(z) = \text{sign}(z)(|z| - \lambda)_+$.*

Proof. For orthonormal $\boldsymbol{X}$, the Lasso objective separates into p independent problems:

$$\min_{\beta_j} \frac{1}{2}(\hat{\beta}_j^{\text{OLS}} - \beta_j)^2 + \lambda |\beta_j|.$$

The subdifferential condition $0 \in -(\hat{\beta}_j^{\text{OLS}} - \beta_j) + \lambda \, \partial|\beta_j|$ yields three cases:

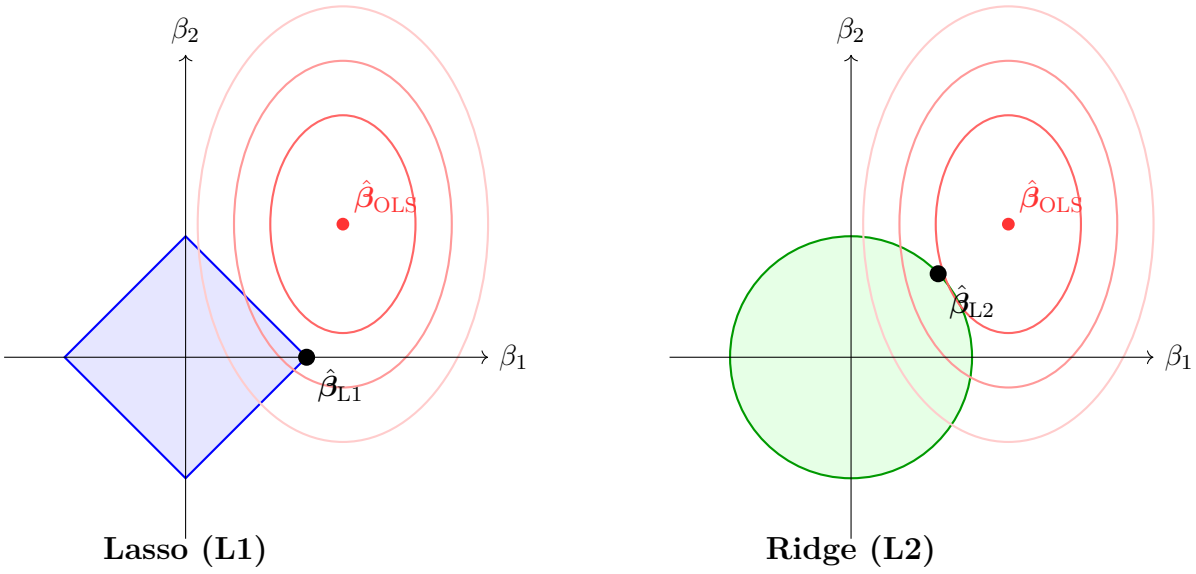
- If $\hat{\beta}_j^{\text{OLS}} > \lambda$: $\hat{\beta}_j = \hat{\beta}_j^{\text{OLS}} - \lambda > 0$.
- If $\hat{\beta}_j^{\text{OLS}} < -\lambda$: $\hat{\beta}_j = \hat{\beta}_j^{\text{OLS}} + \lambda < 0$.
- If $|\hat{\beta}_j^{\text{OLS}}| \leq \lambda$: $\hat{\beta}_j = 0$.

This produces the soft-thresholding formula. □

Proposition 4.12 (Bayesian Interpretation of Lasso)**.** The Lasso corresponds to MAP estimation with a Laplace prior $P(\beta_j) = \frac{1}{2b} \exp(-|\beta_j|/b)$ with $b = \sigma^2/(n\lambda)$. The sharp peak at zero in the Laplace density explains why Lasso produces sparse solutions.

Remark 4.13. Unlike Ridge, the Lasso has no closed-form solution for general $\boldsymbol{X}$. It is typically solved via coordinate descent or the LARS (Least Angle Regression) algorithm, which efficiently computes the entire regularization path.

4.3.1 Geometric Interpretation



Why L1 Produces Sparsity

The L1 constraint region (diamond) has corners on the coordinate axes. Since the RSS contours are ellipsoidal, their first point of contact with the diamond typically occurs at a corner, where one or more $\beta_j = 0$. The L2 ball (circle) has no corners, so the contact point generically has all $\beta_j \neq 0$. In p dimensions, the L1 ball has $2p$ corners and 2^p faces, making sparse intersections increasingly likely as p grows.

4.4 Elastic Net

Definition 4.14 (Elastic Net)**.** The Elastic Net combines L1 and L2 penalties:

$$J_{\text{EN}}(\boldsymbol{\beta}) = \frac{1}{2n} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|^2 + \lambda\left[\alpha \|\boldsymbol{\beta}\|_1 + \tfrac{1-\alpha}{2} \|\boldsymbol{\beta}\|_2^2\right],$$

where $\alpha \in [0, 1]$ controls the mix ($\alpha = 1$: pure Lasso, $\alpha = 0$: pure Ridge).

Theorem 4.15 (Elastic Net Grouping Effect)**.** *If $x_j \approx x_k$ (correlated features), the Elastic Net satisfies*

$$|\hat{\beta}_j - \hat{\beta}_k| \leq \frac{\|\boldsymbol{y}\|}{\lambda(1-\alpha)} \sqrt{2(1 - r_{jk})},$$

where r_{jk} is the sample correlation between x_j and x_k. As $r_{jk} \to 1$, the coefficients are forced to be similar.

When to Use Elastic Net

- Groups of correlated features: Lasso tends to select one arbitrarily; Elastic Net selects the group.
- $p \gg n$: Lasso selects at most n features; Elastic Net has no such limitation.
- In practice, set $\alpha \in \{0.1, 0.5, 0.9, 0.95, 1.0\}$ and select via CV.

4.5 Regularization Paths

Definition 4.16 (Regularization Path)**.** The *regularization path* is the set of solutions $\{\hat{\boldsymbol{\beta}}(\lambda) : \lambda \geq 0\}$ as λ varies from 0 (OLS) to ∞ (all-zero coefficients). For Lasso, the path is piecewise linear. For Ridge, coefficients shrink smoothly toward zero.

Proposition 4.17 (Maximum λ for Lasso)**.** The smallest λ for which $\hat{\boldsymbol{\beta}}_{\text{Lasso}} = \mathbf{0}$ is $\lambda_{\max} = \frac{1}{n} \left\| \boldsymbol{X}^\top \boldsymbol{y} \right\|_\infty$. This provides a natural starting point for the regularization path.

4.5.1 Cross-Validation for λ

K-Fold CV for Regularization Strength

1. Choose a grid $\lambda_1 > \lambda_2 > \cdots > \lambda_m > 0$ (typically log-spaced).
2. For each λ_k and each fold $j \in \{1, \ldots, K\}$:
   - Train on $\mathcal{D} \setminus \mathcal{D}_j$, evaluate on $\mathcal{D}_j$.
3. $\hat{\lambda} = \arg\min_{\lambda_k} \frac{1}{K} \sum_{j=1}^{K} \text{Err}_j(\lambda_k)$.
4. (Optional) *One-SE rule*: choose the largest λ within one standard error of the minimum CV error for a sparser model.

Remark 4.18. The one-standard-error rule, proposed by Breiman et al., selects a simpler model that is statistically indistinguishable from the best. It often yields better generalisation because it avoids selecting an overly complex model due to noise in the CV estimate.

4.6 Implementation in Python

Ridge, Lasso, Elastic Net with scikit-learn

```
import numpy as np
from sklearn.datasets import make_regression
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import (Ridge, Lasso, ElasticNet,
                                  RidgeCV, LassoCV, ElasticNetCV)
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error

# Generate high-dimensional data with few informative features
X, y = make_regression(n_samples=200, n_features=50,
                       n_informative=10, noise=15, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42)

scaler = StandardScaler().fit(X_train)
X_tr = scaler.transform(X_train)
X_te = scaler.transform(X_test)
```

```python
# Ridge with CV
ridge_cv = RidgeCV(alphas=np.logspace(-3, 3, 50), cv=5)
ridge_cv.fit(X_tr, y_train)
print(f"Ridge  alpha={ridge_cv.alpha_:.4f}  "
      f"Test MSE={mean_squared_error(y_test,
      ↪  ridge_cv.predict(X_te)):.2f}")

# Lasso with CV
lasso_cv = LassoCV(alphas=np.logspace(-3, 1, 50), cv=5, max_iter=10000)
lasso_cv.fit(X_tr, y_train)
n_nonzero = np.sum(lasso_cv.coef_ != 0)
print(f"Lasso  alpha={lasso_cv.alpha_:.4f}  "
      f"Non-zero: {n_nonzero}/50  "
      f"Test MSE={mean_squared_error(y_test,
      ↪  lasso_cv.predict(X_te)):.2f}")

# Elastic Net with CV
en_cv = ElasticNetCV(l1_ratio=[0.1, 0.5, 0.9, 0.95],
                     alphas=np.logspace(-3, 1, 50),
                     cv=5, max_iter=10000)
en_cv.fit(X_tr, y_train)
print(f"ElasticNet alpha={en_cv.alpha_:.4f}  l1_ratio={en_cv.l1_ratio_}
↪  "
      f"Test MSE={mean_squared_error(y_test, en_cv.predict(X_te)):.2f}")
```

Output

```
Ridge  alpha=1.2328  Test MSE=254.31
Lasso  alpha=0.2310  Non-zero: 12/50  Test MSE=218.47
ElasticNet alpha=0.1274  l1_ratio=0.9  Test MSE=215.83
```

Plotting Regularization Paths

```python
import matplotlib.pyplot as plt
from sklearn.linear_model import lasso_path

alphas, coefs, _ = lasso_path(X_tr, y_train,
                              alphas=np.logspace(-3, 1, 100))

fig, ax = plt.subplots(figsize=(8, 5))
for i in range(coefs.shape[0]):
    ax.plot(np.log10(alphas), coefs[i], linewidth=0.8)
ax.set_xlabel(r"$\log_{10}(\lambda)$")
ax.set_ylabel(r"$\hat{\beta}_j$")
ax.set_title("Lasso Regularization Path")
ax.axvline(np.log10(lasso_cv.alpha_), color="k",
           linestyle="--", label=r"$\lambda_{\mathrm{CV}}$")
```

```
ax.legend()
plt.tight_layout()
plt.savefig("lasso_path.pdf")
```

Ridge from Scratch

```
def ridge_fit(X, y, lam=1.0):
    """Ridge regression via the normal equations."""
    n, p = X.shape
    I = np.eye(p)
    beta = np.linalg.solve(X.T @ X + lam * I, X.T @ y)
    return beta

# Compare with sklearn
beta_scratch = ridge_fit(X_tr, y_train, lam=ridge_cv.alpha_)
beta_sklearn = Ridge(alpha=ridge_cv.alpha_).fit(X_tr, y_train).coef_
print(f"Max difference: {np.max(np.abs(beta_scratch -
↪  beta_sklearn)):.2e}")
```

Output

```
Max difference: 7.11e-15
```

4.7 Comparison Summary

Ridge vs. Lasso vs. Elastic Net

	Ridge	Lasso	Elastic Net
Penalty	$\lambda \|\boldsymbol{\beta}\|_2^2$	$\lambda \|\boldsymbol{\beta}\|_1$	$\lambda[\alpha \|\boldsymbol{\beta}\|_1 + \frac{1-\alpha}{2} \|\boldsymbol{\beta}\|_2^2]$
Sparsity	No	Yes	Yes
Closed-form	Yes	No	No
Correlated features	Shared weight	One selected	Group selected
Bayesian prior	Gaussian	Laplace	Mixture
Max features ($p > n$)	All	$\leq n$	All

4.8 Exercises

Exercise 4.1 (Ridge Derivation)**.** Starting from the Ridge objective, derive the closed-form solution. Verify that the Hessian is positive definite for any $\lambda > 0$, even when $\boldsymbol{X}$ does not have full column rank.

Exercise 4.2 (SVD and Ridge Shrinkage)**.** Let $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top$ be the SVD of the centred design matrix. Show that the Ridge fitted values are $\hat{\boldsymbol{y}} = \sum_j \frac{d_j^2}{d_j^2+\lambda} \boldsymbol{u}_j \boldsymbol{u}_j^\top \boldsymbol{y}$. Plot the shrinkage factors $d_j^2/(d_j^2+\lambda)$ for several values of λ.

Exercise 4.3 (Soft Thresholding)**.** Prove the soft-thresholding formula for the Lasso with orthonormal design. Graph the mapping $\hat{\beta}^{\text{OLS}} \mapsto \hat{\beta}^{\text{Lasso}}$ and compare with the Ridge shrinkage $\hat{\beta}^{\text{Ridge}} = \hat{\beta}^{\text{OLS}}/(1+\lambda)$.

Exercise 4.4 (Regularization Path Experiment)**.** Using `make_regression` with $p = 100$ features (only 5 informative):

1. Plot Lasso and Ridge coefficient paths as λ varies.
2. Identify the Lasso λ at which only the informative features remain.
3. Use 10-fold CV to select the optimal λ and report test MSE.

Exercise 4.5 (Elastic Net Grouping Effect)**.** Generate $n = 100$ observations with $p = 3$ features where $X_1 = X_2 + \epsilon$ (strong correlation) and X_3 is independent. Fit Lasso and Elastic Net. Show that Elastic Net assigns similar weights to X_1 and X_2 while Lasso arbitrarily selects one.

Exercise 4.6 (Bayesian Regularization)**.** Derive the MAP estimator under the prior $\beta_j \sim \text{Laplace}(0, b)$ and likelihood $y|\boldsymbol{X}, \boldsymbol{\beta} \sim \mathcal{N}(\boldsymbol{X}\boldsymbol{\beta}, \sigma^2 \boldsymbol{I})$. Express the regularization parameter λ in terms of b and σ^2.

Exercise 4.7 (Effective Degrees of Freedom)**.** Show that the effective degrees of freedom of Ridge regression, $\text{df}(\lambda) = \text{tr}(\boldsymbol{H}_\lambda)$ where $\boldsymbol{H}_\lambda = \boldsymbol{X}(\boldsymbol{X}^\top\boldsymbol{X} + \lambda \boldsymbol{I})^{-1}\boldsymbol{X}^\top$, satisfies $\text{df}(\lambda) = \sum_j d_j^2/(d_j^2+\lambda)$. Verify that $\text{df}(0) = p$ and $\text{df}(\lambda) \to 0$ as $\lambda \to \infty$.

Chapter 5

Support Vector Machines

Chapter Overview

Support Vector Machines find the hyperplane that separates classes with the *largest margin*. This geometric principle leads to a convex optimisation problem with strong theoretical guarantees. This chapter derives the hard-margin and soft-margin formulations, introduces the dual problem and KKT conditions, and presents the kernel trick for nonlinear classification.

5.1 Maximum Margin Classifier

5.1.1 Setup and Geometric Motivation

Definition 5.1 (Linear Classifier)**.** Given training data $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ with $\boldsymbol{x}_i \in \mathbb{R}^p$ and $y_i \in \{-1, +1\}$, a linear classifier predicts

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

where $\boldsymbol{w} \in \mathbb{R}^p$ is the weight vector and $b \in \mathbb{R}$ the bias.

Definition 5.2 (Functional and Geometric Margin)**.** The *functional margin* of sample $(\boldsymbol{x}_i, y_i)$ is $\hat{\gamma}_i = y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b)$. The *geometric margin* is the signed distance to the hyperplane:

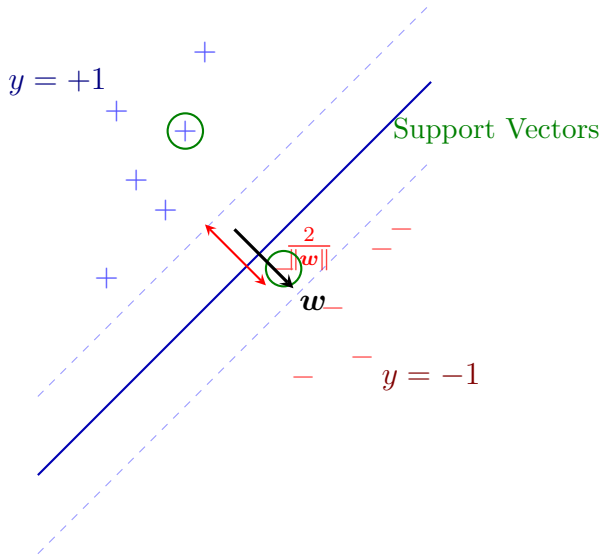
$$\gamma_i = \frac{y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b)}{\|\boldsymbol{w}\|}.$$

The margin of the dataset is $\gamma = \min_i \gamma_i$. A positive functional margin indicates correct classification.

Lemma 5.3 (Distance to a Hyperplane)**.** *For a hyperplane $H = \{\boldsymbol{x} : \boldsymbol{w}^\top \boldsymbol{x} + b = 0\}$, the Euclidean distance from any point $\boldsymbol{x}_0$ to H is*

$$d(\boldsymbol{x}_0, H) = \frac{|\boldsymbol{w}^\top \boldsymbol{x}_0 + b|}{\|\boldsymbol{w}\|}.$$

Proof. Any point on the hyperplane satisfies $\boldsymbol{w}^\top \boldsymbol{x} + b = 0$. The projection of $\boldsymbol{x}_0$ onto H along the normal $\boldsymbol{w}/\|\boldsymbol{w}\|$ gives $\boldsymbol{x}_0 - \frac{\boldsymbol{w}^\top \boldsymbol{x}_0 + b}{\|\boldsymbol{w}\|^2}\boldsymbol{w}$. The distance is $\left\| \frac{\boldsymbol{w}^\top \boldsymbol{x}_0 + b}{\|\boldsymbol{w}\|^2}\boldsymbol{w} \right\| = \frac{|\boldsymbol{w}^\top \boldsymbol{x}_0 + b|}{\|\boldsymbol{w}\|}$. □



5.2 Hard-Margin SVM

Definition 5.4 (Hard-Margin Primal Problem)**.** Assuming the data is linearly separable, the maximum-margin hyperplane solves

$$\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2 \qquad \text{s.t.} \quad y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1, \quad i = 1, \ldots, n.$$

Theorem 5.5 (Margin Width)**.** *The margin of the optimal hyperplane is* $\gamma^* = \frac{2}{\|\boldsymbol{w}^*\|}$*. The constraints* $y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1$ *normalise the functional margin to* 1*, so the geometric margin equals* $1/\|\boldsymbol{w}\|$ *on each side.*

Remark 5.6. Maximising $\frac{2}{\|\boldsymbol{w}\|}$ is equivalent to minimising $\frac{1}{2}\|\boldsymbol{w}\|^2$, which is a convex quadratic objective with linear constraints — a Quadratic Program (QP). The $\frac{1}{2}$ is for algebraic convenience and does not affect the solution.

Proposition 5.7 (Uniqueness)**.** Since the objective is strictly convex and the constraints are linear (hence convex), the hard-margin SVM has a unique solution $(\boldsymbol{w}^*, b^*)$ whenever the feasible set is non-empty (data is linearly separable).

5.3 Soft-Margin SVM

Definition 5.8 (Soft-Margin Primal Problem)**.** When data is not perfectly separable, introduce slack variables $\xi_i \geq 0$:

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n} \xi_i \qquad \text{s.t.} \quad \begin{cases} y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1 - \xi_i, \\ \xi_i \geq 0, \quad i = 1, \ldots, n. \end{cases}$$

The parameter $C > 0$ trades off margin width against training errors.

Proposition 5.9 (Interpretation of ξ_i)**.**

- $\xi_i = 0$: point i is correctly classified and outside the margin.
- $0 < \xi_i < 1$: correctly classified but inside the margin.

- $\xi_i = 1$: on the decision boundary.
- $\xi_i > 1$: misclassified.

The sum $\sum_i \xi_i$ is an upper bound on the number of training errors.

Role of C

$C \to \infty$ recovers the hard-margin SVM (no violations allowed). Small C permits more margin violations, producing a wider margin but more misclassifications. The optimal C is typically selected via cross-validation over a logarithmic grid (e.g. $C \in \{10^{-3}, 10^{-2}, \ldots, 10^{3}\}$).

Proposition 5.10 (Hinge Loss Formulation)**.** The soft-margin SVM is equivalent to the unconstrained problem

$$\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{n} \max\bigl(0,\, 1 - y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b)\bigr),$$

where $\ell_{\text{hinge}}(z) = \max(0, 1 - z)$ is the *hinge loss*.

5.4 Dual Formulation and KKT Conditions

5.4.1 Lagrangian and Dual

Theorem 5.11 (Dual Problem of the Soft-Margin SVM)**.** *The Lagrangian dual of the soft-margin SVM is*

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle \qquad s.t. \quad \begin{cases} 0 \le \alpha_i \le C, & \forall\, i, \\ \sum_{i=1}^{n} \alpha_i y_i = 0. \end{cases}$$

Proof. Form the Lagrangian with multipliers $\alpha_i \ge 0$ (for the margin constraints) and $\mu_i \ge 0$ (for $\xi_i \ge 0$):

$$\mathcal{L} = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_i \xi_i - \sum_i \alpha_i [y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) - 1 + \xi_i] - \sum_i \mu_i \xi_i.$$

Setting partial derivatives to zero:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = \boldsymbol{w} - \sum_i \alpha_i y_i \boldsymbol{x}_i = \boldsymbol{0} \implies \boldsymbol{w} = \sum_i \alpha_i y_i \boldsymbol{x}_i, \tag{5.1}$$

$$\frac{\partial \mathcal{L}}{\partial b} = -\sum_i \alpha_i y_i = 0, \tag{5.2}$$

$$\frac{\partial \mathcal{L}}{\partial \xi_i} = C - \alpha_i - \mu_i = 0 \implies \alpha_i \le C. \tag{5.3}$$

Substituting (5.1)–(5.3) back into $\mathcal{L}$ eliminates $\boldsymbol{w}$, b, and $\boldsymbol{\xi}$:

$$\begin{aligned} \mathcal{L} &= \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle - \sum_{i,j} \alpha_i \alpha_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle + \sum_i \alpha_i \\ &= \sum_i \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle, \end{aligned}$$

yielding the dual objective. □

Remark 5.12. The dual problem is a concave QP with box constraints ($0 \le \alpha_i \le C$) and one linear equality constraint ($\sum \alpha_i y_i = 0$). The SMO (Sequential Minimal Optimisation) algorithm by Platt (1998) solves it efficiently by iteratively optimising pairs of α_i values.

5.4.2 KKT Conditions

Theorem 5.13 (KKT Conditions for SVM)**.** *At optimality, the following conditions hold for all i:*

1. ***Primal feasibility:*** $y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \ge 1 - \xi_i$, $\xi_i \ge 0$.
2. ***Dual feasibility:*** $0 \le \alpha_i \le C$.
3. ***Complementary slackness:***

$$\alpha_i[y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) - 1 + \xi_i] = 0, \qquad (C - \alpha_i)\xi_i = 0.$$

4. ***Stationarity:*** $\boldsymbol{w} = \sum_i \alpha_i y_i \boldsymbol{x}_i$ *and* $\sum_i \alpha_i y_i = 0$.

Proposition 5.14 (Classification of Training Points via KKT)**.** The KKT conditions partition training points into three categories:

- $\alpha_i = 0$: non-support vector, lies outside the margin, $\xi_i = 0$.
- $0 < \alpha_i < C$: support vector on the margin, $\xi_i = 0$, $y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) = 1$.
- $\alpha_i = C$: support vector inside or on wrong side of margin, $\xi_i > 0$.

Proposition 5.15 (Support Vectors)**.** A training point $(\boldsymbol{x}_i, y_i)$ is a *support vector* if and only if $\alpha_i > 0$. The optimal $\boldsymbol{w}$ depends only on support vectors via $\boldsymbol{w} = \sum_{i:\alpha_i>0} \alpha_i y_i \boldsymbol{x}_i$.

Proposition 5.16 (Recovery of b)**.** For any support vector with $0 < \alpha_i < C$ (so $\xi_i = 0$ by the second complementary slackness condition):

$$b = y_i - \boldsymbol{w}^\top \boldsymbol{x}_i = y_i - \sum_{j:\alpha_j>0} \alpha_j y_j \langle \boldsymbol{x}_j, \boldsymbol{x}_i \rangle.$$

In practice, average over all such support vectors for numerical stability.

5.5 Kernel Trick — Introduction

Definition 5.17 (Kernel Function)**.** A function $K : \mathbb{R}^p \times \mathbb{R}^p \to \mathbb{R}$ is a (positive definite) kernel if there exists a feature map $\phi : \mathbb{R}^p \to \mathcal{H}$ into a Hilbert space such that

$$K(\boldsymbol{x}, \boldsymbol{x}') = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \rangle_{\mathcal{H}}.$$

Theorem 5.18 (Kernelised SVM)**.** *In the dual, the data appear only via inner products $\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$. Replacing these by $K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ implicitly maps data to a higher-dimensional (possibly infinite) space without ever computing $\phi(\boldsymbol{x})$. The decision function becomes*

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b.$$

Example 5.19 (Common Kernels).

- **Linear:** $K(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^\top \boldsymbol{x}'$.
- **Polynomial:** $K(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x}^\top \boldsymbol{x}' + c)^d$, $c \geq 0$, $d \in \mathbb{N}$.
- **RBF (Gaussian):** $K(\boldsymbol{x}, \boldsymbol{x}') = \exp\!\left(-\gamma \left\|\boldsymbol{x} - \boldsymbol{x}'\right\|^2\right)$, $\gamma = \frac{1}{2\sigma^2} > 0$.
- **Sigmoid:** $K(\boldsymbol{x}, \boldsymbol{x}') = \tanh(\kappa\, \boldsymbol{x}^\top \boldsymbol{x}' + c)$ (valid only for certain κ, c).

Lemma 5.20 (Mercer's Condition). *A continuous symmetric function K is a valid kernel if and only if the Gram matrix $[K(\boldsymbol{x}_i, \boldsymbol{x}_j)]_{i,j}$ is positive semi-definite for any finite set $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$.*

Proposition 5.21 (RBF Feature Space). The RBF kernel corresponds to an infinite-dimensional feature map. Its Taylor expansion $e^{-\gamma\|\boldsymbol{x}-\boldsymbol{x}'\|^2} = e^{-\gamma\|\boldsymbol{x}\|^2} e^{-\gamma\|\boldsymbol{x}'\|^2} \sum_{k=0}^{\infty} \frac{(2\gamma)^k}{k!} (\boldsymbol{x}^\top \boldsymbol{x}')^k$ shows it is a weighted sum of polynomial kernels of all degrees.

Choosing the Kernel

- Start with a linear kernel; it is fast and interpretable.
- Use RBF when the decision boundary is nonlinear; tune $\gamma = 1/(2\sigma^2)$ and C via grid search.
- Polynomial kernels are useful when feature interactions of a known degree matter.
- Always standardise features before training an SVM.
- scikit-learn default for `SVC`: RBF kernel with $\gamma = 1/(p \cdot \mathrm{Var}(X))$ (`gamma='scale'`).

5.6 Implementation in Python

5.6.1 SVM with scikit-learn

SVM Classification with scikit-learn

```
import numpy as np
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, classification_report

X, y = make_classification(n_samples=300, n_features=2,
                           n_redundant=0, n_clusters_per_class=1,
                           random_state=42)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42)

scaler = StandardScaler().fit(X_train)
X_tr = scaler.transform(X_train)
```

```
X_te = scaler.transform(X_test)

# Grid search over C and kernel
param_grid = {
    "C": [0.1, 1, 10, 100],
    "kernel": ["linear", "rbf"],
    "gamma": ["scale", "auto"],
}
grid = GridSearchCV(SVC(), param_grid, cv=5, scoring="accuracy")
grid.fit(X_tr, y_train)

print(f"Best params: {grid.best_params_}")
print(f"CV accuracy: {grid.best_score_:.3f}")
print(f"Test accuracy: {accuracy_score(y_test, grid.predict(X_te)):.3f}")

# Number of support vectors
best_svm = grid.best_estimator_
print(f"Support vectors per class: {best_svm.n_support_}")
print(f"Total support vectors: {sum(best_svm.n_support_)} /
↪  {len(y_train)}")
```

Output

```
Best params: {'C': 1, 'gamma': 'scale', 'kernel': 'rbf'}
CV accuracy: 0.919
Test accuracy: 0.922
Support vectors per class: [32 35]
Total support vectors: 67 / 210
```

5.6.2 SVM from Scratch (Simplified)

Soft-Margin SVM via Subgradient Descent

```
import numpy as np

def svm_sgd(X, y, C=1.0, eta0=0.01, n_iter=1000):
    """Train a linear SVM using subgradient descent on the
    primal hinge loss: (1/2)||w||^2 + C * sum max(0, 1-y_i(w.x_i+b))."""
    n, p = X.shape
    w = np.zeros(p)
    b = 0.0
    for t in range(1, n_iter + 1):
        eta = eta0 / t  # decaying learning rate
        for i in range(n):
            margin = y[i] * (X[i] @ w + b)
            if margin < 1:
                w -= eta * (w - C * y[i] * X[i])
                b += eta * C * y[i]
```

```
            else:
                w -= eta * w
    return w, b

def svm_predict(X, w, b):
    return np.sign(X @ w + b)

# Train and evaluate
y_train_svm = 2 * y_train - 1  # convert {0,1} -> {-1,+1}
y_test_svm = 2 * y_test - 1

w, b = svm_sgd(X_tr, y_train_svm, C=1.0, eta0=0.01, n_iter=500)
preds = svm_predict(X_te, w, b)
acc = np.mean(preds == y_test_svm)
print(f"From-scratch SVM accuracy: {acc:.3f}")
```

Output

```
From-scratch SVM accuracy: 0.911
```

Visualising Decision Boundary

```
import matplotlib.pyplot as plt

def plot_svm_boundary(clf, X, y, ax, title):
    h = 0.02
    x_min, x_max = X[:, 0].min() - 0.5, X[:, 0].max() + 0.5
    y_min, y_max = X[:, 1].min() - 0.5, X[:, 1].max() + 0.5
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                         np.arange(y_min, y_max, h))
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    ax.contourf(xx, yy, Z, alpha=0.3, cmap="coolwarm")
    ax.scatter(X[:, 0], X[:, 1], c=y, cmap="coolwarm",
               edgecolors="k", s=20)
    ax.set_title(title)

fig, axes = plt.subplots(1, 2, figsize=(10, 4))
for ax, kernel in zip(axes, ["linear", "rbf"]):
    clf = SVC(kernel=kernel, C=1).fit(X_tr, y_train)
    plot_svm_boundary(clf, X_tr, y_train, ax, f"SVM ({kernel})")
plt.tight_layout()
plt.savefig("svm_boundaries.pdf")
```

5.7 Exercises

Exercise 5.1 (Margin Derivation)**.** Show that for a hyperplane $\boldsymbol{w}^\top \boldsymbol{x} + b = 0$, the distance from a point $\boldsymbol{x}_0$ to the hyperplane is $\frac{|\boldsymbol{w}^\top \boldsymbol{x}_0 + b|}{\|\boldsymbol{w}\|}$. Use this to prove that the margin width is $\frac{2}{\|\boldsymbol{w}\|}$ when the closest points satisfy $y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) = 1$.

Exercise 5.2 (Dual Derivation)**.** Starting from the hard-margin Lagrangian $\mathcal{L} = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_i \alpha_i [y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) - 1]$, derive the dual problem step by step. Show that strong duality holds (Slater's condition is satisfied when the data is separable).

Exercise 5.3 (KKT Analysis)**.** Consider a trained SVM with $n = 5$ training points. Suppose $\alpha = (0, 0.8, 0, 1.5, 0)$ with $C = 2$. Determine which points are support vectors, which are on the margin, and which are margin violators. What are the values of ξ_i for each point?

Exercise 5.4 (Kernel Computation)**.** Show that the polynomial kernel $K(\boldsymbol{x}, \boldsymbol{x}') = (1 + \boldsymbol{x}^\top \boldsymbol{x}')^2$ in $\mathbb{R}^2$ corresponds to the feature map $\phi(x_1, x_2) = (1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1 x_2)^\top$. Verify by computing both K and $\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \rangle$ for specific vectors.

Exercise 5.5 (RBF Kernel SVM)**.** Using `make_moons(noise=0.3)`:

1. Train SVMs with linear and RBF kernels and plot decision boundaries.
2. Perform a grid search over $C \in \{0.1, 1, 10, 100\}$ and $\gamma \in \{0.01, 0.1, 1, 10\}$ for the RBF kernel.
3. Report the best parameters and test accuracy.
4. Discuss the effect of γ: what happens when γ is very large? Very small?

Exercise 5.6 (SVM vs. Logistic Regression)**.** Prove that the SVM hinge loss $\max(0, 1 - y\, f(\boldsymbol{x}))$ upper bounds the 0-1 loss $\mathbb{1}\{y \neq \text{sign}(f(\boldsymbol{x}))\}$. Compare the hinge loss with the logistic loss $\log(1 + e^{-y\, f(\boldsymbol{x})})$ analytically and numerically. When might one prefer logistic regression over SVM?

Exercise 5.7 (Number of Support Vectors)**.** On the `breast_cancer` dataset, train a linear SVM for $C \in \{10^{-3}, 10^{-2}, \ldots, 10^{3}\}$. For each C, record the number of support vectors and the test accuracy. Plot both quantities. What is the relationship between C, the number of support vectors, and generalisation?

Chapter 5 Summary

- **Primal (soft):** $\min \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum \xi_i \quad \text{s.t. } y_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b) \geq 1 - \xi_i$
- **Hinge loss:** $\min \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum \max(0, 1 - y_i f(\boldsymbol{x}_i))$
- **Dual:** $\max \sum \alpha_i - \frac{1}{2}\sum_{i,j} \alpha_i \alpha_j y_i y_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \quad \text{s.t. } 0 \leq \alpha_i \leq C$
- **Decision:** $f(\boldsymbol{x}) = \sum_i \alpha_i y_i K(\boldsymbol{x}_i, \boldsymbol{x}) + b$
- **KKT:** $\alpha_i[y_i f(\boldsymbol{x}_i) - 1 + \xi_i] = 0$, $(C - \alpha_i)\xi_i = 0$
- **Common kernels:** Linear, Polynomial $(\boldsymbol{x}^\top \boldsymbol{x}' + c)^d$, RBF $e^{-\gamma\|\boldsymbol{x} - \boldsymbol{x}'\|^2}$

Chapter 6

Decision Trees

Chapter Overview

Decision trees partition the feature space into axis-aligned regions via a sequence of binary splits. They are interpretable, handle mixed data types, and require no feature scaling. This chapter derives the splitting criteria (entropy, Gini, variance reduction), presents the CART algorithm, discusses pruning, and covers regression trees.

6.1 CART Algorithm

Definition 6.1 (Decision Tree)**.** A decision tree is a function $T : \mathbb{R}^p \to \mathcal{Y}$ defined by a binary tree where each internal node tests a condition $x_j \leq t$ for some feature j and threshold t, and each leaf assigns a prediction $\hat{y}$.

CART (Classification and Regression Trees)

1. Start with the full dataset at the root node.
2. For each node, find the best split (j^*, t^*):

$$(j^*, t^*) = \arg\min_{j,t} \Big[\frac{n_L}{n} Q(R_L) + \frac{n_R}{n} Q(R_R) \Big],$$

   where $R_L = \{i : x_{ij} \leq t\}$, $R_R = \{i : x_{ij} > t\}$, and Q is an impurity measure.
3. Create two child nodes with the subsets R_L and R_R.
4. Recurse on each child until a stopping criterion is met (max depth, min samples, or pure node).
5. Assign to each leaf the majority class (classification) or mean target (regression).

Remark 6.2. CART uses *greedy* top-down splitting. Finding the globally optimal tree is NP-hard, so the greedy heuristic is standard. Each split considers all p features and $O(n)$ thresholds, giving $O(np)$ cost per node.

Proposition 6.3 (Computational Complexity)**.** Building a full binary tree has depth

$O(\log n)$ to $O(n)$. At each level, all n samples are scanned across p features. Sorting-based splitting achieves $O(np \log n)$ time per level and $O(np \log^2 n)$ overall for balanced trees. The space complexity is $O(n + |T|)$ where $|T|$ is the number of nodes.

6.2 Splitting Criteria for Classification

6.2.1 Entropy and Information Gain

Definition 6.4 (Shannon Entropy)**.** For a node with class proportions $\hat{p}_k = \frac{n_k}{n}$, $k = 1, \ldots, K$, the entropy is

$$H = -\sum_{k=1}^{K} \hat{p}_k \log_2 \hat{p}_k,$$

with the convention $0 \log 0 = 0$. Entropy is maximised when all classes are equally likely ($H = \log_2 K$) and minimised at $H = 0$ for a pure node.

Theorem 6.5 (Entropy Bounds)**.** *For K classes: $0 \leq H \leq \log_2 K$. Equality on the left holds iff one $\hat{p}_k = 1$; on the right iff $\hat{p}_k = 1/K$ for all k.*

Proof. Non-negativity follows from $\hat{p}_k \in [0, 1]$ so $-\hat{p}_k \log_2 \hat{p}_k \geq 0$. The upper bound follows from Jensen's inequality applied to the concave function $f(x) = -x \log_2 x$:

$$H = \sum_k f(\hat{p}_k) \leq K \cdot f\left(\frac{1}{K}\right) = K \cdot \frac{1}{K} \log_2 K = \log_2 K.$$

Alternatively, $H \leq \log_2 K$ is equivalent to $D_{\mathrm{KL}}(\hat{p} \,\|\, \text{Uniform}) \geq 0$. □

Definition 6.6 (Information Gain)**.** The information gain from splitting a node S into subsets S_L and S_R is

$$\mathrm{IG}(S, j, t) = H(S) - \frac{|S_L|}{|S|} H(S_L) - \frac{|S_R|}{|S|} H(S_R).$$

CART selects the split (j^*, t^*) that maximises IG.

Proposition 6.7 (Non-Negativity of Information Gain)**.** $\mathrm{IG}(S, j, t) \geq 0$ for all splits, with equality iff the class distribution is identical in S_L and S_R. This follows from the concavity of entropy (Jensen's inequality).

Example 6.8 (Computing Information Gain)**.** Consider a node with 20 samples: 12 positive, 8 negative. Entropy: $H = -\frac{12}{20} \log_2 \frac{12}{20} - \frac{8}{20} \log_2 \frac{8}{20} \approx 0.971$.

A split sends 10 positive and 2 negative to the left ($n_L = 12$), and 2 positive and 6 negative to the right ($n_R = 8$):

$$\begin{aligned} H(S_L) &= -\tfrac{10}{12} \log_2 \tfrac{10}{12} - \tfrac{2}{12} \log_2 \tfrac{2}{12} \approx 0.650, \\ H(S_R) &= -\tfrac{2}{8} \log_2 \tfrac{2}{8} - \tfrac{6}{8} \log_2 \tfrac{6}{8} \approx 0.811, \\ \mathrm{IG} &= 0.971 - \tfrac{12}{20}(0.650) - \tfrac{8}{20}(0.811) \approx 0.256. \end{aligned}$$

6.2.2 Gini Impurity

Definition 6.9 (Gini Impurity)**.**

$$G = \sum_{k=1}^{K} \hat{p}_k(1 - \hat{p}_k) = 1 - \sum_{k=1}^{K} \hat{p}_k^2.$$

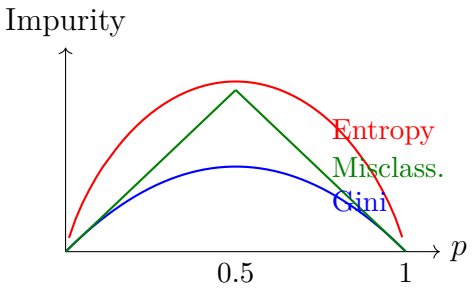
Proposition 6.10 (Gini Properties)**.** For K classes: $0 \leq G \leq 1 - 1/K$. Gini measures the probability that two randomly drawn samples from the node belong to different classes. For binary classification ($K = 2$), $G = 2\hat{p}(1 - \hat{p})$.

Theorem 6.11 (Relationship Between Entropy and Gini)**.** *For binary classification with $\hat{p}_1 = p$, both $H(p) = -p \log_2 p - (1-p) \log_2(1-p)$ and $G(p) = 2p(1-p)$ are concave, symmetric around $p = 0.5$, and zero at $p \in \{0, 1\}$. A second-order Taylor expansion of H around $p = 0.5$ gives $H \approx 1 - \frac{2}{\ln 2}(p - 0.5)^2$, which is close to $\frac{1}{\ln 2} G$ after rescaling.*

Entropy vs. Gini

- In practice, they produce very similar trees (disagreement in $< 2\%$ of cases).
- Gini is slightly faster to compute (no logarithm).
- Entropy tends to produce slightly more balanced splits.
- scikit-learn defaults to Gini (`criterion="gini"`).

6.2.3 Comparison of Impurity Measures



Remark 6.12. The misclassification error $1 - \max_k \hat{p}_k$ is not differentiable and not strictly concave, making it unsuitable for greedy splitting (it cannot distinguish between splits that improve purity at different rates). However, it is a valid metric for evaluating the final tree.

6.3 Regression Trees

Definition 6.13 (Regression Tree Impurity)**.** For regression, the impurity of a node R is the variance of the target values:

$$Q(R) = \frac{1}{|R|} \sum_{i \in R} (y_i - \bar{y}_R)^2, \qquad \bar{y}_R = \frac{1}{|R|} \sum_{i \in R} y_i.$$

The prediction at each leaf is $\hat{y} = \bar{y}_R$, the mean of the training targets in that region.

Proposition 6.14 (Optimal Constant Prediction)**.** For squared loss, the constant c minimising $\sum_{i \in R}(y_i - c)^2$ is $c = \bar{y}_R$. This justifies the use of the mean as the leaf prediction.

Theorem 6.15 (Variance Reduction)**.** *The reduction in impurity for a split is*

$$\Delta Q = Q(R) - \frac{|R_L|}{|R|} Q(R_L) - \frac{|R_R|}{|R|} Q(R_R) = \frac{|R_L| \cdot |R_R|}{|R|^2} (\bar{y}_{R_L} - \bar{y}_{R_R})^2.$$

The split maximising ΔQ separates the node into subsets with maximally different means.

Proof. Expand $Q(R) = \frac{1}{|R|} \sum_i y_i^2 - \bar{y}_R^2$ and apply the same decomposition to $Q(R_L)$ and $Q(R_R)$. Using $\bar{y}_R = \frac{|R_L|}{|R|} \bar{y}_{R_L} + \frac{|R_R|}{|R|} \bar{y}_{R_R}$, algebraic simplification yields the stated result. □

Remark 6.16. The variance reduction formula shows that regression tree splits are equivalent to finding the split that maximises the between-group sum of squares, analogous to a one-way ANOVA.

6.4 Pruning

Definition 6.17 (Cost-Complexity Pruning)**.** Define the cost-complexity criterion for a subtree T of the full tree T_0:

$$C_\alpha(T) = \sum_{m=1}^{|T|} \sum_{i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha \, |T|,$$

where $|T|$ is the number of leaves and $\alpha \geq 0$ controls the trade-off between fit and complexity. For each α, there is a unique smallest subtree T_α minimising C_α.

Minimal Cost-Complexity Pruning

1. Grow the full tree T_0 (no stopping criterion other than pure leaves or minimum node size).
2. For increasing α, compute the sequence of subtrees $T_0 \supseteq T_1 \supseteq \cdots \supseteq \{root\}$ by *weakest-link cutting*: repeatedly remove the internal node whose removal increases the training error the least per removed leaf.
3. Select α^* via K-fold cross-validation.
4. Return T_{α^*}.

Proposition 6.18 (Weakest Link Cutting)**.** For an internal node t with subtree T_t, define the effective α at which pruning t becomes worthwhile:

$$g(t) = \frac{R(t) - R(T_t)}{|T_t| - 1},$$

where $R(t)$ is the impurity of node t treated as a leaf and $R(T_t) = \sum_{m \in \text{leaves}(T_t)} R(m)$. At each step, prune the node with the smallest $g(t)$.

Overfitting Without Pruning

An unpruned tree can achieve zero training error by creating one leaf per training sample. This memorises the data and yields poor generalisation. Controlling tree depth, minimum samples per leaf, or post-hoc pruning are essential. Common hyperparameters: `max_depth`, `min_samples_split`, `min_samples_leaf`, `max_leaf_nodes`, `ccp_alpha`.

6.5 Advantages and Limitations

Strengths and Weaknesses of Decision Trees

Strengths:

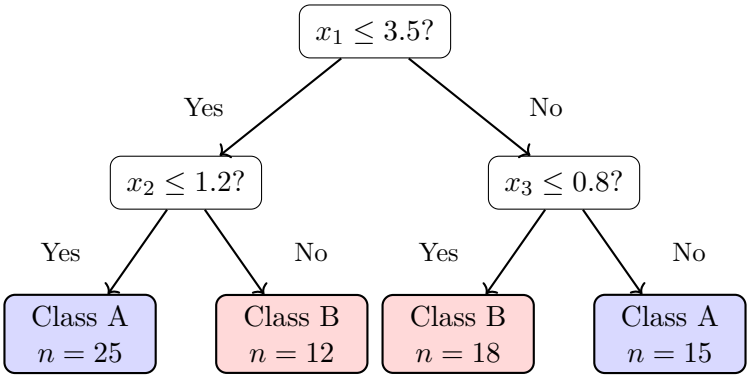
- Highly interpretable ("white box" model).
- Handle numerical and categorical features without preprocessing.
- Invariant to monotone transformations of features (no need for standardisation).
- Capture non-linear relationships and interactions automatically.
- Fast training and prediction.

Weaknesses:

- High variance: small changes in data can produce very different trees.
- Axis-aligned splits cannot efficiently represent diagonal boundaries.
- Prone to overfitting without regularization (pruning).
- Greedy construction does not guarantee global optimality.

These weaknesses motivate ensemble methods (bagging, random forests, boosting).

6.6 Tree Diagram



6.7 Implementation in Python

Decision Tree Classification with scikit-learn

```python
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.tree import DecisionTreeClassifier, export_text, plot_tree
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

X, y = load_iris(return_X_y=True)
feature_names = load_iris().feature_names
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42)

# Train with default settings (no pruning)
tree_full = DecisionTreeClassifier(random_state=42)
tree_full.fit(X_train, y_train)
print(f"Full tree depth: {tree_full.get_depth()}")
print(f"Full tree leaves: {tree_full.get_n_leaves()}")
print(f"Train acc: {tree_full.score(X_train, y_train):.3f}")
print(f"Test acc:  {tree_full.score(X_test, y_test):.3f}")

# Train with pruning constraints
tree_pruned = DecisionTreeClassifier(
    max_depth=3, min_samples_leaf=5, random_state=42)
tree_pruned.fit(X_train, y_train)
print(f"\nPruned tree depth: {tree_pruned.get_depth()}")
print(f"Pruned test acc:   {tree_pruned.score(X_test, y_test):.3f}")

# Print tree rules
print("\nTree rules:")
print(export_text(tree_pruned, feature_names=feature_names))
```

Output

```
Full tree depth: 5
Full tree leaves: 9
Train acc: 1.000
Test acc:  0.956

Pruned tree depth: 3
Pruned test acc:   0.956

Tree rules:
|--- petal length (cm) <= 2.45
|   |--- class: 0
|--- petal length (cm) >  2.45
|   |--- petal width (cm) <= 1.75
```

```
|   |   |--- petal length (cm) <= 4.95
|   |   |   |--- class: 1
|   |   |--- petal length (cm) >  4.95
|   |   |   |--- class: 2
|   |--- petal width (cm) >  1.75
|   |   |--- class: 2
```

Cost-Complexity Pruning Path

```
# Find optimal alpha via cost-complexity pruning
path = tree_full.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas = path.ccp_alphas

# Train a tree for each alpha and evaluate via CV
cv_scores = []
for alpha in ccp_alphas:
    clf = DecisionTreeClassifier(ccp_alpha=alpha, random_state=42)
    scores = cross_val_score(clf, X_train, y_train, cv=5)
    cv_scores.append(scores.mean())

best_idx = np.argmax(cv_scores)
best_alpha = ccp_alphas[best_idx]
print(f"Best ccp_alpha: {best_alpha:.5f}")
print(f"Best CV accuracy: {cv_scores[best_idx]:.3f}")

# Final model
tree_ccp = DecisionTreeClassifier(
    ccp_alpha=best_alpha, random_state=42)
tree_ccp.fit(X_train, y_train)
print(f"Test accuracy: {tree_ccp.score(X_test, y_test):.3f}")
print(f"Number of leaves: {tree_ccp.get_n_leaves()}")
```

Visualising the Tree

```
fig, ax = plt.subplots(figsize=(14, 6))
plot_tree(tree_ccp, feature_names=feature_names,
          class_names=load_iris().target_names,
          filled=True, rounded=True, ax=ax,
          fontsize=9)
ax.set_title("Cost-Complexity Pruned Decision Tree")
plt.tight_layout()
plt.savefig("decision_tree_plot.pdf")
```

Regression Tree

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
```

```python
# Synthetic 1D regression data
np.random.seed(42)
X_reg = np.sort(5 * np.random.rand(200, 1), axis=0)
y_reg = np.sin(X_reg.ravel()) + np.random.randn(200) * 0.2

# Fit trees of different depths
fig, axes = plt.subplots(1, 3, figsize=(14, 4))
for ax, depth in zip(axes, [2, 5, 15]):
    reg = DecisionTreeRegressor(max_depth=depth, random_state=42)
    reg.fit(X_reg, y_reg)
    X_plot = np.linspace(0, 5, 500).reshape(-1, 1)
    ax.scatter(X_reg, y_reg, s=10, alpha=0.5, label="Data")
    ax.plot(X_plot, reg.predict(X_plot), "r-", lw=2,
            label=f"depth={depth}")
    mse = mean_squared_error(y_reg, reg.predict(X_reg))
    ax.set_title(f"Depth {depth}, Train MSE={mse:.3f}")
    ax.legend()
plt.tight_layout()
plt.savefig("regression_trees.pdf")
```

Feature Importance

```python
# Feature importance from the pruned tree
importances = tree_ccp.feature_importances_
for name, imp in zip(feature_names, importances):
    print(f"  {name:25s}: {imp:.4f}")

# Bar plot
fig, ax = plt.subplots(figsize=(6, 3))
ax.barh(feature_names, importances, color="steelblue")
ax.set_xlabel("Feature Importance (Gini)")
ax.set_title("Decision Tree Feature Importances")
plt.tight_layout()
plt.savefig("tree_importances.pdf")
```

Output

```
  sepal length (cm)         : 0.0000
  sepal width (cm)          : 0.0000
  petal length (cm)         : 0.4484
  petal width (cm)          : 0.5516
```

6.8 Exercises

Exercise 6.1 (Entropy Computation)**.** A node contains 30 samples: 10 from class A, 12 from class B, 8 from class C. Compute the entropy and Gini impurity. A split sends the

10 A samples and 2 B samples to the left child, and the remaining to the right. Compute the information gain.

Exercise 6.2 (Gini Derivation)**.** Prove that for binary classification, the split threshold t^* that minimises the weighted Gini impurity satisfies a condition involving the cumulative class frequencies. Show that the optimal split can be found in $O(n \log n)$ time by sorting the feature values once and scanning.

Exercise 6.3 (Regression Tree by Hand)**.** Given the dataset $\{(1, 2.1), (2, 3.8), (3, 4.2), (4, 8.1), (5, 7.9), (6,$ build a regression tree of depth 2 by hand. At each node, enumerate all possible splits, compute the variance reduction, and select the best.

Exercise 6.4 (Pruning Experiment)**.** Using the `breast_cancer` dataset:

1. Train a full (unpruned) decision tree. Report depth, number of leaves, train/test accuracy.
2. Plot CV accuracy versus `ccp_alpha`.
3. Select the optimal α and compare the pruned tree with the full tree.
4. Compare the number of features used by each tree.

Exercise 6.5 (Instability of Trees)**.** Decision trees are known for high variance. Generate 10 bootstrap samples from the Iris dataset. Train a decision tree on each and compare the selected root split. How often does it change? Discuss how ensemble methods (bagging, random forests) address this issue.

Exercise 6.6 (Feature Importance)**.** scikit-learn computes feature importance as the total reduction in impurity brought by each feature. Formally, for feature j:

$$\text{Imp}(j) = \sum_{t \in T:\, j(t)=j} \frac{n_t}{n} \Delta Q(t),$$

where the sum is over all nodes t that split on feature j. Train a tree on the Iris dataset, extract `feature_importances_`, and verify the formula by computing it manually from the tree structure using `tree_` attributes.

Exercise 6.7 (Comparison with Other Models)**.** On the `wine` dataset, compare a decision tree with logistic regression and k-NN:

1. Report 5-fold CV accuracy for each method.
2. For the decision tree, find the optimal `max_depth` via CV.
3. Discuss the interpretability–performance trade-off.

Chapter 6 Summary

- **Entropy:** $H = -\sum_k \hat{p}_k \log_2 \hat{p}_k$
- **Gini:** $G = 1 - \sum_k \hat{p}_k^2$
- **Info Gain:** $\text{IG} = H(S) - \frac{|S_L|}{|S|} H(S_L) - \frac{|S_R|}{|S|} H(S_R)$

- **Regression impurity:** $Q(R) = \frac{1}{|R|}\sum_{i \in R}(y_i - \bar{y}_R)^2$
- **Variance reduction:** $\Delta Q = \frac{n_L n_R}{n^2}(\bar{y}_L - \bar{y}_R)^2$
- **Cost-complexity:** $C_\alpha(T) = \sum_m \mathrm{RSS}_m + \alpha|T|$
- **CART:** Greedy, top-down, binary splits; $O(np \log n)$ per level

Chapter 7

Ensemble Methods

Why Combine Models?

A single decision tree is a high-variance estimator: small perturbations of the training set can lead to radically different trees. By aggregating many such estimators, we can dramatically reduce variance while preserving (or even improving) predictive accuracy. This chapter formalises that intuition and develops the two great families of ensembles: *bagging* and *boosting*.

7.1 Bias–Variance Decomposition for Ensembles

Theorem 7.1 (Variance Reduction by Averaging)**.** *Let $f_1, \dots, f_B$ be B estimators, each with $\mathbb{E}[f_b(\boldsymbol{x})] = \mu(\boldsymbol{x})$, $\mathrm{Var}[f_b(\boldsymbol{x})] = \sigma^2$, and pairwise correlation ρ. Define the ensemble prediction*

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

Then

$$\mathrm{Var}[\bar{f}(\boldsymbol{x})] = \rho\,\sigma^2 + \frac{1-\rho}{B}\,\sigma^2.$$

Proof.

$$\mathrm{Var}\Big[\frac{1}{B}\sum_b f_b\Big] = \frac{1}{B^2}\Big[\sum_b \mathrm{Var}[f_b] + \sum_{b\neq b'} \mathrm{Cov}(f_b, f_{b'})\Big] = \frac{1}{B^2}\big[B\sigma^2 + B(B-1)\rho\sigma^2\big] = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2.$$

□

Remark 7.2. As $B \to \infty$, $\mathrm{Var}[\bar{f}] \to \rho\sigma^2$. The irreducible term $\rho\sigma^2$ vanishes only if the base learners are uncorrelated ($\rho = 0$). Bagging and random forests are precisely designed to reduce ρ.

7.2 Bagging (Bootstrap Aggregating)

Definition 7.3 (Bootstrap Sample)**.** Given a dataset $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$, a *bootstrap sample* $\mathcal{D}^{*b}$ is obtained by drawing n observations uniformly with replacement from $\mathcal{D}$.

Bagging

1. For $b = 1, \dots, B$:
   (a) Draw bootstrap sample $\mathcal{D}^{*b}$ from $\mathcal{D}$.
   (b) Fit base learner $\hat{f}_b$ on $\mathcal{D}^{*b}$.
2. Aggregate:

$$\hat{f}_{\text{bag}}(\boldsymbol{x}) = \begin{cases} \frac{1}{B}\sum_{b=1}^{B} \hat{f}_b(\boldsymbol{x}) & \text{(regression)}, \\ \arg\max_k \sum_{b=1}^{B} \mathbb{1}\{\hat{f}_b(\boldsymbol{x}) = k\} & \text{(classification)}. \end{cases}$$

Proposition 7.4 (Out-of-Bag Probability)**.** The probability that observation i is *not* selected in a particular bootstrap sample is

$$\left(1 - \frac{1}{n}\right)^n \xrightarrow{n\to\infty} e^{-1} \approx 0.368.$$

Hence roughly 36.8% of observations are left out of each bootstrap sample.

7.3 Random Forests

Definition 7.5 (Random Forest)**.** A *random forest* is a bagged ensemble of decision trees where, at each split, only a random subset of m features (out of p) is considered as candidates. Typical defaults are $m = \lfloor \sqrt{p} \rfloor$ (classification) and $m = \lfloor p/3 \rfloor$ (regression).

The feature subsampling further decorrelates the trees, reducing ρ in the variance formula and hence reducing overall variance.

7.3.1 Out-of-Bag Error

Definition 7.6 (OOB Error)**.** For each observation $(\boldsymbol{x}_i, y_i)$, the *OOB prediction* is the aggregate prediction using only those trees b for which $i \notin \mathcal{D}^{*b}$:

$$\hat{f}_{\text{OOB}}(\boldsymbol{x}_i) = \frac{1}{|\{b : i \notin \mathcal{D}^{*b}\}|} \sum_{b:\, i \notin \mathcal{D}^{*b}} \hat{f}_b(\boldsymbol{x}_i).$$

The OOB error is the average loss over all training observations evaluated via their OOB predictions. It provides an unbiased estimate of generalisation error without a separate validation set.

7.4 AdaBoost

Boosting builds the ensemble *sequentially*: each new learner focuses on the examples that previous learners misclassified.

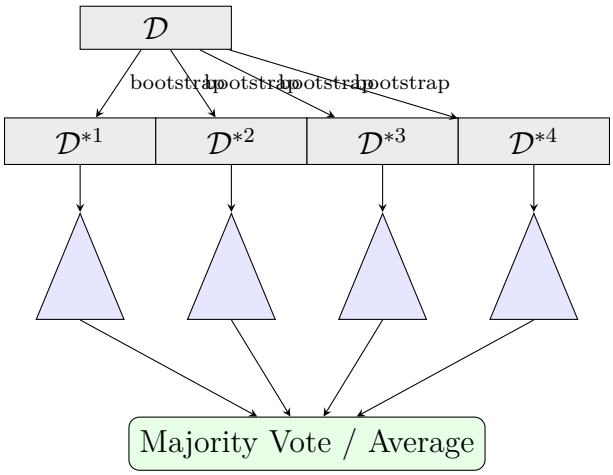


Figure 7.1: Schematic of the bagging / random forest procedure.

AdaBoost (Binary Classification)

Input: Training set $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, $y_i \in \{-1, +1\}$; number of rounds T.

Initialise weights $w_i^{(1)} = 1/n$ for all i.

For $t = 1, \ldots, T$:

1. Fit weak learner h_t to training data weighted by $\boldsymbol{w}^{(t)}$.
2. Compute weighted error: $\epsilon_t = \sum_{i=1}^n w_i^{(t)} \mathbb{1}\{h_t(\boldsymbol{x}_i) \neq y_i\}$.
3. Compute learner weight: $\alpha_t = \frac{1}{2} \ln\!\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$.
4. Update sample weights: $w_i^{(t+1)} = \frac{w_i^{(t)} \exp(-\alpha_t\, y_i\, h_t(\boldsymbol{x}_i))}{Z_t}$, where Z_t is a normalisation constant.

Output: $H(\boldsymbol{x}) = \operatorname{sign}\!\left(\sum_{t=1}^T \alpha_t\, h_t(\boldsymbol{x})\right)$.

Theorem 7.7 (AdaBoost Training Error Bound)**.** *The training error of the final classifier satisfies*

$$\frac{1}{n}\sum_{i=1}^{n} \mathbb{1}\{H(\boldsymbol{x}_i) \neq y_i\} \;\leq\; \prod_{t=1}^{T} Z_t \;=\; \prod_{t=1}^{T} 2\sqrt{\epsilon_t(1-\epsilon_t)}.$$

Proof. For any misclassified example, $y_i H(\boldsymbol{x}_i) < 0$, so $\mathbb{1}\{H(\boldsymbol{x}_i) \neq y_i\} \leq \exp(-y_i \sum_t \alpha_t h_t(\boldsymbol{x}_i))$. Averaging over i and unrolling the weight updates gives

$$\frac{1}{n}\sum_{i} \exp\!\Big(-y_i \sum_{t} \alpha_t h_t(\boldsymbol{x}_i)\Big) = \prod_{t=1}^{T} Z_t.$$

Each normalisation constant equals $Z_t = 2\sqrt{\epsilon_t(1-\epsilon_t)}$, which follows by direct computation of the partition function using the definition of α_t. □

Remark 7.8. The choice $\alpha_t = \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t}$ is exactly the value that minimises Z_t (and hence the training error upper bound) at each round. This is derived by differentiating Z_t with respect to α_t and setting the derivative to zero.

7.5 Gradient Boosting

Definition 7.9 (Gradient Boosting)**.** Gradient boosting builds an additive model $F_T(\boldsymbol{x}) = \sum_{t=1}^{T} \gamma_t h_t(\boldsymbol{x})$ by performing *functional gradient descent* in the space of functions. At round t, the pseudo-residuals are

$$r_i^{(t)} = -\frac{\partial \mathcal{L}(y_i, F(\boldsymbol{x}_i))}{\partial F(\boldsymbol{x}_i)}\bigg|_{F=F_{t-1}},$$

and the next base learner h_t is fitted to $(\boldsymbol{x}_i, r_i^{(t)})_{i=1}^n$.

For squared loss $\mathcal{L}(y, F) = \frac{1}{2}(y - F)^2$, the pseudo-residuals are simply $r_i^{(t)} = y_i - F_{t-1}(\boldsymbol{x}_i)$, i.e. the actual residuals.

7.5.1 XGBoost and LightGBM

XGBoost Objective

At round t, XGBoost minimises a second-order approximation of the regularised objective:

$$\tilde{\mathcal{L}}^{(t)} = \sum_{i=1}^{n} \left[g_i\, h_t(\boldsymbol{x}_i) + \tfrac{1}{2} h_i\, h_t(\boldsymbol{x}_i)^2 \right] + \Omega(h_t),$$

where $g_i = \partial_F \mathcal{L}(y_i, F_{t-1}(\boldsymbol{x}_i))$, $h_i = \partial_F^2 \mathcal{L}(y_i, F_{t-1}(\boldsymbol{x}_i))$, and $\Omega(h) = \gamma\, T_{\text{leaves}} + \frac{\lambda}{2} \sum_j w_j^2$ is a regularisation term penalising tree complexity.
The optimal leaf weight for leaf j is:

$$w_j^* = -\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i + \lambda},$$

and the corresponding gain for a candidate split is:

$$\text{Gain} = \frac{1}{2}\left[\frac{(\sum_{i \in I_L} g_i)^2}{\sum_{i \in I_L} h_i + \lambda} + \frac{(\sum_{i \in I_R} g_i)^2}{\sum_{i \in I_R} h_i + \lambda} - \frac{(\sum_{i \in I_j} g_i)^2}{\sum_{i \in I_j} h_i + \lambda} \right] - \gamma.$$

XGBoost vs LightGBM

- **XGBoost**: level-wise tree growth; exact or approximate split finding via quantile sketch.
- **LightGBM**: leaf-wise growth (best-first); Gradient-based One-Side Sampling (GOSS) and Exclusive Feature Bundling (EFB) for speed on large datasets.
- Both support GPU training, early stopping, and built-in cross-validation.

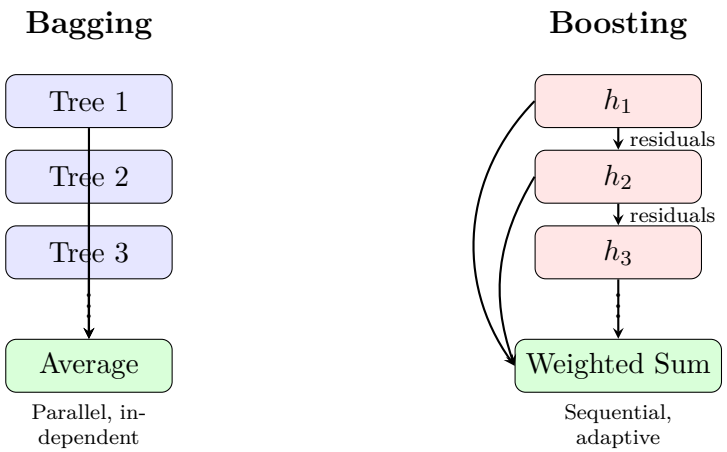


Figure 7.2: Bagging (parallel, reduces variance) vs Boosting (sequential, reduces bias).

Table 7.1: Summary of ensemble methods.

Method	Strategy	Main effect	Base learner	Risk
Bagging	Parallel	↓ Variance	Any (often trees)	Low overfitting
Random Forest	Parallel	↓↓ Variance	Decorrelated trees	Low overfitting
AdaBoost	Sequential	↓ Bias	Weak (stumps)	Sensitive to noise
Gradient Boost	Sequential	↓ Bias	Regression trees	Can overfit
XGBoost	Sequential	↓ Bias	Regularised trees	Robust

7.6 Bagging vs Boosting: A Visual Comparison

7.7 Comparison of Ensemble Methods

7.8 Implementation in Python

Random Forest, AdaBoost, and Gradient Boosting with scikit-learn

```
import numpy as np
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import (
    BaggingClassifier,
    RandomForestClassifier,
    AdaBoostClassifier,
    GradientBoostingClassifier,
)
from sklearn.tree import DecisionTreeClassifier

# Generate synthetic data
X, y = make_classification(
    n_samples=1000, n_features=20,
    n_informative=10, random_state=42
```

```
)

models = {
    "Bagging (Trees)": BaggingClassifier(
        estimator=DecisionTreeClassifier(),
        n_estimators=100, random_state=42
    ),
    "Random Forest": RandomForestClassifier(
        n_estimators=100, max_features="sqrt", random_state=42
    ),
    "AdaBoost": AdaBoostClassifier(
        n_estimators=100, learning_rate=0.5, random_state=42
    ),
    "Gradient Boosting": GradientBoostingClassifier(
        n_estimators=200, learning_rate=0.1,
        max_depth=3, random_state=42
    ),
}

for name, model in models.items():
    scores = cross_val_score(model, X, y, cv=5, scoring="accuracy")
    print(f"{name:25s}  accuracy = {scores.mean():.4f} +/-
    ↪  {scores.std():.4f}")
```

Output

```
Bagging (Trees)           accuracy = 0.9190 +/- 0.0222
Random Forest             accuracy = 0.9320 +/- 0.0198
AdaBoost                  accuracy = 0.9090 +/- 0.0265
Gradient Boosting         accuracy = 0.9410 +/- 0.0177
```

OOB Error and Feature Importance

```
# OOB error estimation (no separate validation set needed)
rf = RandomForestClassifier(
    n_estimators=300, oob_score=True, random_state=42
)
rf.fit(X, y)
print(f"OOB accuracy: {rf.oob_score_:.4f}")

# Feature importances (mean decrease in impurity)
importances = rf.feature_importances_
indices = np.argsort(importances)[::-1]
print("\nTop 5 features:")
for rank, idx in enumerate(indices[:5], 1):
    print(f"  {rank}. Feature {idx}: importance =
    ↪  {importances[idx]:.4f}")
```

XGBoost with Early Stopping

```
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

xgb = XGBClassifier(
    n_estimators=500, learning_rate=0.05,
    max_depth=4, reg_lambda=1.0, subsample=0.8,
    colsample_bytree=0.8, early_stopping_rounds=20,
    eval_metric="logloss", random_state=42
)
xgb.fit(
    X_train, y_train,
    eval_set=[(X_test, y_test)], verbose=False
)
print(f"Best iteration: {xgb.best_iteration}")
print(f"Test accuracy:  {xgb.score(X_test, y_test):.4f}")
```

7.9 Exercises

Exercise 7.1 (Variance of Bagged Estimator)**.** Let $f_1, \dots, f_B$ be i.i.d. with variance σ^2. Show that the variance of their average is σ^2/B. Now suppose they share pairwise correlation $\rho > 0$; prove the formula $\mathrm{Var}[\bar{f}] = \rho\sigma^2 + (1-\rho)\sigma^2/B$ and interpret the limit as $B \to \infty$.

Exercise 7.2 (AdaBoost Weight Update)**.** Show that the choice $\alpha_t = \frac{1}{2}\ln\frac{1-\epsilon_t}{\epsilon_t}$ minimises the normalisation constant $Z_t = \sum_i w_i^{(t)} \exp(-\alpha_t y_i h_t(\boldsymbol{x}_i))$. *Hint:* Split the sum into correctly and incorrectly classified subsets.

Exercise 7.3 (Random Forest vs Bagging)**.** Using the `make_classification` dataset above with $p = 50$ features, compare the 5-fold CV accuracy of `BaggingClassifier` (full trees) and `RandomForestClassifier` for $B \in \{10, 50, 100, 200, 500\}$. Plot both curves. At what B does each method plateau?

Exercise 7.4 (Gradient Boosting from Scratch)**.** Implement gradient boosting for regression (squared loss) using `DecisionTreeRegressor(max_depth=3)` as the base learner. Use a learning rate $\eta = 0.1$ and $T = 100$ rounds. Compare your implementation with `GradientBoostingRegressor` on the Boston-style housing dataset.

Exercise 7.5 (Hyperparameter Tuning for XGBoost)**.** Use `RandomizedSearchCV` to tune the following XGBoost hyperparameters on a dataset of your choice: `max_depth` $\in \{3, \dots, 10\}$, `learning_rate` $\in [0.01, 0.3]$, `subsample` $\in [0.6, 1.0]$, `colsample_bytree` $\in [0.6, 1.0]$, `n_estimators` $\in \{100, \dots, 1000\}$. Report the best parameters and test set performance.

Chapter 8

Unsupervised Learning — Clustering

Learning Without Labels

In unsupervised learning we are given only inputs $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_n\} \subset \mathbb{R}^p$ with no target variable. Clustering seeks to partition the data into groups of "similar" points. The challenge is to formalise similarity and to determine the number of clusters without ground-truth labels.

8.1 k-Means Clustering

8.1.1 Objective and Algorithm

Definition 8.1 (k-Means Objective)**.** Given data $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_n\} \subset \mathbb{R}^p$ and a number of clusters k, the k-means objective is

$$J(\boldsymbol{C}, \boldsymbol{\mu}) = \sum_{j=1}^{k} \sum_{i \in C_j} \|\boldsymbol{x}_i - \boldsymbol{\mu}_j\|^2,$$

where $C_1, \dots, C_k$ is a partition of $\{1, \dots, n\}$ and $\boldsymbol{\mu}_j$ is the centroid of cluster C_j.

Lloyd's Algorithm (k-Means)

1. **Initialise** centroids $\boldsymbol{\mu}_1^{(0)}, \dots, \boldsymbol{\mu}_k^{(0)}$ (e.g. random selection from data).
2. **Repeat** until convergence:
   (a) **Assignment:** $C_j^{(t)} = \left\{ i : j = \arg\min_\ell \left\| \boldsymbol{x}_i - \boldsymbol{\mu}_\ell^{(t-1)} \right\|^2 \right\}$.
   (b) **Update:** $\boldsymbol{\mu}_j^{(t)} = \frac{1}{|C_j^{(t)}|} \sum_{i \in C_j^{(t)}} \boldsymbol{x}_i$.

Theorem 8.2 (Convergence of k-Means)**.** *Lloyd's algorithm monotonically decreases the objective J:*

$$J^{(t+1)} \leq J^{(t)},$$

with equality if and only if the algorithm has converged. Since there are finitely many partitions, the algorithm terminates in a finite number of steps.

Proof. The assignment step minimises J over partitions for fixed centroids (each point is assigned to its nearest centroid). The update step minimises J over centroids for a fixed partition (the mean minimises the sum of squared distances). Hence J is non-increasing at each half-step. Since $J \geq 0$ and the number of distinct partitions is finite, the algorithm must converge. □

Local Minima

The k-means objective is non-convex; Lloyd's algorithm converges to a *local* minimum that depends on the initialisation. In practice, the algorithm is run multiple times with different random seeds, and the best solution (lowest J) is retained.

8.1.2 k-Means++ Initialisation

k-Means++

1. Choose $\boldsymbol{\mu}_1$ uniformly at random from $\{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$.
2. For $j = 2, \ldots, k$: choose $\boldsymbol{\mu}_j = \boldsymbol{x}_i$ with probability proportional to $\min_{\ell<j} \|\boldsymbol{x}_i - \boldsymbol{\mu}_\ell\|^2$.
3. Run standard k-means with these initial centroids.

Theorem 8.3 (k-Means++ Guarantee)**.** *The k-means++ initialisation produces an initial clustering whose expected objective satisfies*

$$\mathbb{E}[J_{init}] \leq 8(\ln k + 2)\, J^*,$$

where J^* is the globally optimal k-means objective.

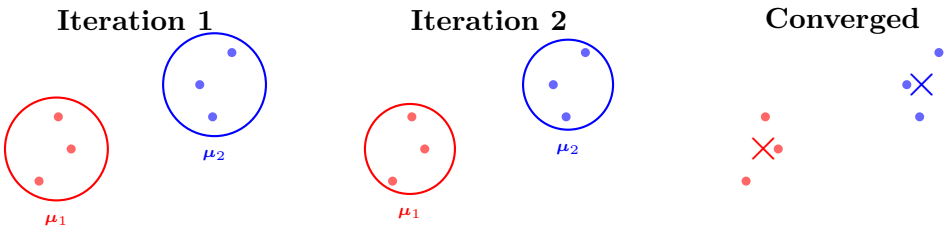


Figure 8.1: Illustration of k-means iterations with $k = 2$.

8.2 Gaussian Mixture Models and EM

Definition 8.4 (Gaussian Mixture Model)**.** A *Gaussian Mixture Model* (GMM) assumes the data is generated from a mixture of k Gaussian distributions:

$$p(\boldsymbol{x} \mid \boldsymbol{\theta}) = \sum_{j=1}^{k} \pi_j \, \mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j),$$

where $\pi_j \geq 0$, $\sum_j \pi_j = 1$ are mixing coefficients, and $\boldsymbol{\theta} = \{(\pi_j, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)\}_{j=1}^{k}$.

8.2.1 The EM Algorithm

The log-likelihood of the observed data is

$$\ell(\boldsymbol{\theta}) = \sum_{i=1}^{n} \ln \left[\sum_{j=1}^{k} \pi_j \, \mathcal{N}(\boldsymbol{x}_i \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) \right],$$

which is intractable to maximise directly due to the logarithm of a sum.

Definition 8.5 (Responsibilities)**.** The *responsibility* of component j for observation $\boldsymbol{x}_i$ is

$$\gamma_{ij} = \frac{\pi_j \, \mathcal{N}(\boldsymbol{x}_i \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}{\sum_{\ell=1}^{k} \pi_\ell \, \mathcal{N}(\boldsymbol{x}_i \mid \boldsymbol{\mu}_\ell, \boldsymbol{\Sigma}_\ell)}.$$

EM Algorithm for GMM

Initialise parameters $\boldsymbol{\theta}^{(0)}$. Repeat until convergence:

E-step: Compute responsibilities using current parameters:

$$\gamma_{ij}^{(t)} = \frac{\pi_j^{(t)} \, \mathcal{N}(\boldsymbol{x}_i \mid \boldsymbol{\mu}_j^{(t)}, \boldsymbol{\Sigma}_j^{(t)})}{\sum_{\ell=1}^{k} \pi_\ell^{(t)} \, \mathcal{N}(\boldsymbol{x}_i \mid \boldsymbol{\mu}_\ell^{(t)}, \boldsymbol{\Sigma}_\ell^{(t)})}.$$

M-step: Update parameters using responsibilities. Let $N_j = \sum_{i=1}^{n} \gamma_{ij}^{(t)}$:

$$\boldsymbol{\mu}_j^{(t+1)} = \frac{1}{N_j} \sum_{i=1}^{n} \gamma_{ij}^{(t)} \, \boldsymbol{x}_i, \tag{8.1}$$

$$\boldsymbol{\Sigma}_j^{(t+1)} = \frac{1}{N_j} \sum_{i=1}^{n} \gamma_{ij}^{(t)} \, (\boldsymbol{x}_i - \boldsymbol{\mu}_j^{(t+1)})(\boldsymbol{x}_i - \boldsymbol{\mu}_j^{(t+1)})^\top, \tag{8.2}$$

$$\pi_j^{(t+1)} = \frac{N_j}{n}. \tag{8.3}$$

Theorem 8.6 (EM Monotonicity)**.** *The EM algorithm monotonically increases the log-likelihood:*

$$\ell(\boldsymbol{\theta}^{(t+1)}) \geq \ell(\boldsymbol{\theta}^{(t)}).$$

Proof. Define the complete-data log-likelihood $\ell_c(\boldsymbol{\theta}) = \sum_i \sum_j z_{ij} [\ln \pi_j + \ln \mathcal{N}(\boldsymbol{x}_i \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)]$ where z_{ij} are latent indicators. By Jensen's inequality applied to the ELBO (Evidence Lower Bound):

$$\ell(\boldsymbol{\theta}) \geq \sum_i \sum_j q(z_{ij}) \big[\ln \pi_j + \ln \mathcal{N}(\boldsymbol{x}_i \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j) - \ln q(z_{ij}) \big] =: \mathcal{F}(q, \boldsymbol{\theta}),$$

for any distribution q. The E-step sets $q(z_{ij}) = \gamma_{ij}$, making the bound tight at $\boldsymbol{\theta}^{(t)}$. The M-step maximises $\mathcal{F}$ over $\boldsymbol{\theta}$, so $\mathcal{F}(q^{(t)}, \boldsymbol{\theta}^{(t+1)}) \geq \mathcal{F}(q^{(t)}, \boldsymbol{\theta}^{(t)}) = \ell(\boldsymbol{\theta}^{(t)})$. Since the ELBO is a lower bound, $\ell(\boldsymbol{\theta}^{(t+1)}) \geq \mathcal{F}(q^{(t)}, \boldsymbol{\theta}^{(t+1)}) \geq \ell(\boldsymbol{\theta}^{(t)})$. □

8.3 DBSCAN

Definition 8.7 (DBSCAN)**.** Given parameters $\epsilon > 0$ and MinPts $\in \mathbb{N}$:

- A point $\boldsymbol{x}$ is a *core point* if $|N_\epsilon(\boldsymbol{x})| \geq \text{MinPts}$, where $N_\epsilon(\boldsymbol{x}) = \{\boldsymbol{x}_i : \|\boldsymbol{x}_i - \boldsymbol{x}\| \leq \epsilon\}$.
- A point $\boldsymbol{x}$ is *density-reachable* from $\boldsymbol{x}'$ if there exists a chain of core points $\boldsymbol{x} = \boldsymbol{p}_1, \boldsymbol{p}_2, \ldots, \boldsymbol{p}_m = \boldsymbol{x}'$ with $\boldsymbol{p}_{i+1} \in N_\epsilon(\boldsymbol{p}_i)$.
- A cluster is a maximal set of density-connected points. Points not reachable from any core point are labelled as *noise*.

Choosing DBSCAN Parameters

- **MinPts**: a common heuristic is MinPts $\geq p + 1$ where p is the dimension.
- ϵ: plot the sorted k-nearest-neighbour distances (with $k = \text{MinPts}$) and look for an "elbow".

8.4 Hierarchical Clustering

Definition 8.8 (Agglomerative Clustering)**.** Agglomerative (bottom-up) hierarchical clustering starts with n singleton clusters and iteratively merges the two closest clusters until a single cluster remains. The result is a *dendrogram*. Common linkage criteria include:

$$d_{\text{single}}(A, B) = \min_{\boldsymbol{a} \in A,\, \boldsymbol{b} \in B} \|\boldsymbol{a} - \boldsymbol{b}\|, \tag{8.4}$$

$$d_{\text{complete}}(A, B) = \max_{\boldsymbol{a} \in A,\, \boldsymbol{b} \in B} \|\boldsymbol{a} - \boldsymbol{b}\|, \tag{8.5}$$

$$d_{\text{average}}(A, B) = \frac{1}{|A||B|} \sum_{\boldsymbol{a} \in A} \sum_{\boldsymbol{b} \in B} \|\boldsymbol{a} - \boldsymbol{b}\|, \tag{8.6}$$

$$d_{\text{Ward}}(A, B) = \frac{|A|\,|B|}{|A| + |B|} \left\|\bar{\boldsymbol{a}} - \bar{\boldsymbol{b}}\right\|^2. \tag{8.7}$$

8.5 Cluster Validity Indices

Definition 8.9 (Silhouette Score)**.** For observation i in cluster C_k, define:

- $a(i) = \frac{1}{|C_k| - 1} \sum_{j \in C_k,\, j \neq i} \|\boldsymbol{x}_i - \boldsymbol{x}_j\|$ (mean intra-cluster distance),
- $b(i) = \min_{\ell \neq k} \frac{1}{|C_\ell|} \sum_{j \in C_\ell} \|\boldsymbol{x}_i - \boldsymbol{x}_j\|$ (mean nearest-cluster distance).

The silhouette coefficient is $s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}} \in [-1, 1]$.

Cluster Validity Indices

- **Silhouette**: $\bar{s} = \frac{1}{n} \sum_i s(i)$. Higher is better (max 1).
- **Calinski–Harabasz**: ratio of between-cluster to within-cluster dispersion. Higher is better.

- **Davies–Bouldin**: average similarity between each cluster and its most similar one. Lower is better.
- **Adjusted Rand Index (ARI)**: measures agreement with ground truth (if available), corrected for chance. Range $[-1, 1]$, with 1 indicating perfect agreement.

8.6 Implementation in Python

k-Means, GMM, DBSCAN, and Hierarchical Clustering

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_score, adjusted_rand_score
from sklearn.preprocessing import StandardScaler

# Generate synthetic data
X, y_true = make_blobs(
    n_samples=500, centers=4,
    cluster_std=[1.0, 1.5, 0.5, 1.2], random_state=42
)
X = StandardScaler().fit_transform(X)

# --- k-Means ---
km = KMeans(n_clusters=4, init="k-means++", n_init=10, random_state=42)
y_km = km.fit_predict(X)
print(f"k-Means:        silhouette = {silhouette_score(X, y_km):.3f}, "
      f"ARI = {adjusted_rand_score(y_true, y_km):.3f}")

# --- GMM ---
gmm = GaussianMixture(n_components=4, covariance_type="full",
↪  random_state=42)
y_gmm = gmm.fit_predict(X)
print(f"GMM:            silhouette = {silhouette_score(X, y_gmm):.3f}, "
      f"ARI = {adjusted_rand_score(y_true, y_gmm):.3f}")

# --- DBSCAN ---
db = DBSCAN(eps=0.5, min_samples=5)
y_db = db.fit_predict(X)
n_clusters_db = len(set(y_db) - {-1})
print(f"DBSCAN:         {n_clusters_db} clusters found, "
      f"noise points = {(y_db == -1).sum()}")

# --- Hierarchical (Ward) ---
hc = AgglomerativeClustering(n_clusters=4, linkage="ward")
y_hc = hc.fit_predict(X)
```

```
print(f"Hierarchical:   silhouette = {silhouette_score(X, y_hc):.3f}, "
      f"ARI = {adjusted_rand_score(y_true, y_hc):.3f}")
```

Output

```
k-Means:        silhouette = 0.527, ARI = 0.876
GMM:            silhouette = 0.518, ARI = 0.891
DBSCAN:         4 clusters found, noise points = 23
Hierarchical:   silhouette = 0.521, ARI = 0.864
```

Elbow Method and Silhouette Analysis

```
inertias, sil_scores = [], []
K_range = range(2, 10)

for k in K_range:
    km = KMeans(n_clusters=k, n_init=10, random_state=42)
    km.fit(X)
    inertias.append(km.inertia_)
    sil_scores.append(silhouette_score(X, km.labels_))

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))
ax1.plot(K_range, inertias, "bo-")
ax1.set_xlabel("k"); ax1.set_ylabel("Inertia")
ax1.set_title("Elbow Method")
ax2.plot(K_range, sil_scores, "rs-")
ax2.set_xlabel("k"); ax2.set_ylabel("Silhouette Score")
ax2.set_title("Silhouette Analysis")
plt.tight_layout()
plt.savefig("figures/ch08_elbow_silhouette.pdf")
```

8.7 Exercises

Exercise 8.1 (k-Means Convergence)**.** Prove that the k-means objective J decreases (or stays constant) at every iteration of Lloyd's algorithm. Argue carefully that both the assignment and the update steps are optimal for their respective sub-problems.

Exercise 8.2 (EM Derivation)**.** Derive the M-step update equations (8.1)–(8.3) by differentiating the expected complete-data log-likelihood $\mathbb{E}_{Z|X,\boldsymbol{\theta}^{(t)}}[\ell_c(\boldsymbol{\theta})]$ with respect to $\boldsymbol{\mu}_j$, $\boldsymbol{\Sigma}_j$, and π_j (using a Lagrange multiplier for the constraint $\sum_j \pi_j = 1$).

Exercise 8.3 (DBSCAN Sensitivity)**.** Generate a dataset with two half-moon shapes (`make_moons`). Apply k-means with $k = 2$ and DBSCAN with various $(\epsilon, \text{MinPts})$ settings. Explain why k-means fails and DBSCAN succeeds. Visualise the results.

Exercise 8.4 (Model Selection for GMM)**.** Fit GMMs with $k \in \{1, \ldots, 8\}$ components to the `make_blobs` dataset above. Plot the BIC $= -2\ell(\hat{\boldsymbol{\theta}}) + d \ln n$ as a function of k, where d is the number of free parameters. Verify that the BIC selects the correct $k = 4$.

Exercise 8.5 (Hierarchical Clustering Dendrogram)**.** Using `scipy.cluster.hierarchy`, compute the linkage matrix for the Iris dataset with Ward linkage. Plot the dendrogram and cut it at a height that yields 3 clusters. Compute the ARI with respect to the true species labels.

Chapter 9

Dimensionality Reduction

The Blessing of Low-Dimensional Structure

Real-world data in $\mathbb{R}^p$ often lies near a much lower-dimensional manifold. Dimensionality reduction discovers this structure, enabling visualisation, noise removal, and faster downstream learning. We study linear methods (PCA) and nonlinear methods (kernel PCA, t-SNE, UMAP).

9.1 The Curse of Dimensionality

Proposition 9.1 (Volume of a Hypersphere)**.** The fraction of the volume of a p-dimensional unit hypercube $[0,1]^p$ captured by an inscribed hypersphere of radius $\frac{1}{2}$ is

$$\frac{V_p(\frac{1}{2})}{1} = \frac{\pi^{p/2}}{2^p\,\Gamma(\frac{p}{2}+1)} \xrightarrow{p\to\infty} 0.$$

In high dimensions, almost all the volume concentrates in the corners.

Proposition 9.2 (Concentration of Distances)**.** For n points drawn uniformly in $[0,1]^p$, the ratio of the maximum to minimum pairwise distances converges to 1:

$$\frac{d_{\max} - d_{\min}}{d_{\min}} \xrightarrow{p\to\infty} 0 \quad \text{in probability.}$$

Distance-based methods (e.g. k-NN, k-means) lose discriminative power.

Practical Consequences

- Sample complexity grows exponentially with p for non-parametric methods.
- Nearest-neighbour distances become nearly indistinguishable.
- Overfitting risk increases: many spurious correlations in high dimensions.

9.2 Principal Component Analysis (PCA)

9.2.1 Derivation via Variance Maximisation

Definition 9.3 (PCA)**.** Let $\boldsymbol{X} \in \mathbb{R}^{n\times p}$ be the centred data matrix ($\bar{\boldsymbol{x}} = \mathbf{0}$). PCA seeks the orthogonal directions of maximum variance. The first principal component direction

is

$$\boldsymbol{w}_1 = \arg\max_{\|\boldsymbol{w}\|=1} \mathrm{Var}[\boldsymbol{X}\boldsymbol{w}] = \arg\max_{\|\boldsymbol{w}\|=1} \boldsymbol{w}^\top \boldsymbol{S}\boldsymbol{w},$$

where $\boldsymbol{S} = \frac{1}{n-1}\boldsymbol{X}^\top\boldsymbol{X}$ is the sample covariance matrix.

Theorem 9.4 (PCA via Eigendecomposition)**.** *The solution to the PCA problem is given by the eigenvectors of $\boldsymbol{S}$. If $\boldsymbol{S} = \boldsymbol{W}\boldsymbol{\Lambda}\boldsymbol{W}^\top$ with eigenvalues $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_p \geq 0$, then:*

1. *The k-th principal component direction is $\boldsymbol{w}_k$ (the k-th eigenvector).*
2. *The variance captured by $\boldsymbol{w}_k$ is λ_k.*
3. *The proportion of total variance explained by the first d components is $\sum_{k=1}^{d}\lambda_k / \sum_{k=1}^{p}\lambda_k$.*

Proof. By the method of Lagrange multipliers with constraint $\|\boldsymbol{w}\| = 1$:

$$\nabla_{\boldsymbol{w}}\left[\boldsymbol{w}^\top\boldsymbol{S}\boldsymbol{w} - \lambda(\boldsymbol{w}^\top\boldsymbol{w} - 1)\right] = \boldsymbol{0} \quad\Longrightarrow\quad \boldsymbol{S}\boldsymbol{w} = \lambda\boldsymbol{w}.$$

So $\boldsymbol{w}$ must be an eigenvector of $\boldsymbol{S}$, and the objective value is $\boldsymbol{w}^\top\boldsymbol{S}\boldsymbol{w} = \lambda$. The maximum is achieved at the eigenvector with the largest eigenvalue. The subsequent components are obtained by restricting to the orthogonal complement of the previously selected directions; by the spectral theorem, these are precisely the remaining eigenvectors in decreasing eigenvalue order. □

9.2.2 PCA via Singular Value Decomposition

Theorem 9.5 (SVD Formulation of PCA)**.** *Let $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top$ be the (thin) SVD, where $\boldsymbol{U} \in \mathbb{R}^{n\times p}$, $\boldsymbol{D} = \mathrm{diag}(\sigma_1, \ldots, \sigma_p)$, $\boldsymbol{V} \in \mathbb{R}^{p\times p}$ orthogonal. Then:*

1. *The principal component directions are the columns of $\boldsymbol{V}$.*
2. *The eigenvalues of $\boldsymbol{S}$ are $\lambda_k = \sigma_k^2/(n-1)$.*
3. *The principal component scores are $\boldsymbol{Z} = \boldsymbol{X}\boldsymbol{V} = \boldsymbol{U}\boldsymbol{D}$.*

The SVD is numerically more stable than explicitly forming $\boldsymbol{S} = \boldsymbol{X}^\top\boldsymbol{X}/(n-1)$.

Proposition 9.6 (PCA as Optimal Low-Rank Approximation)**.** The rank-d truncation $\boldsymbol{X}_d = \boldsymbol{U}_d\boldsymbol{D}_d\boldsymbol{V}_d^\top$ (retaining the top d singular values) solves

$$\min_{\mathrm{rank}(\boldsymbol{M})\leq d} \|\boldsymbol{X} - \boldsymbol{M}\|_F^2,$$

by the Eckart–Young–Mirsky theorem. The reconstruction error is $\sum_{k=d+1}^{p}\sigma_k^2$.

9.2.3 Scree Plots and Choosing d

Selecting the Number of Components

- **Scree plot**: plot λ_k vs k; look for an "elbow".
- **Cumulative variance**: choose d such that $\sum_{k=1}^{d}\lambda_k / \sum_{k=1}^{p}\lambda_k \geq \tau$ (e.g. $\tau = 0.95$).

- **Kaiser's rule**: retain components with $\lambda_k > 1$ (on standardised data).

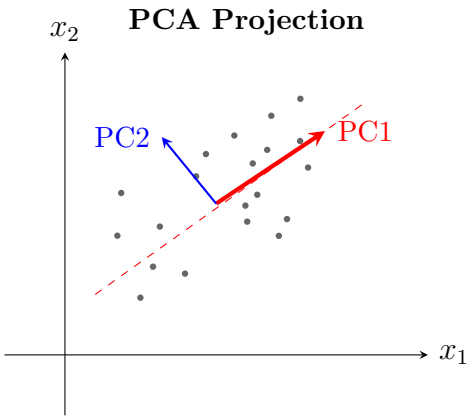


Figure 9.1: PCA finds the directions of maximum variance. The first principal component (red) captures the most variance; the second (blue) is orthogonal to it.

9.3 Kernel PCA

Definition 9.7 (Kernel PCA)**.** Kernel PCA performs PCA in a feature space $\mathcal{H}$ induced by a kernel κ. Given the kernel matrix $\boldsymbol{K} \in \mathbb{R}^{n\times n}$ with $K_{ij} = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$, the centred kernel matrix is

$$\tilde{\boldsymbol{K}} = \boldsymbol{K} - \mathbf{1}_n \boldsymbol{K} - \boldsymbol{K}\mathbf{1}_n + \mathbf{1}_n \boldsymbol{K} \mathbf{1}_n,$$

where $\mathbf{1}_n = \frac{1}{n}\mathbf{1}\mathbf{1}^\top$. The principal components in feature space are obtained from the eigendecomposition $\tilde{\boldsymbol{K}} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^\top$, and the projections are $\boldsymbol{Z} = \boldsymbol{U}\boldsymbol{\Lambda}^{1/2}$.

9.4 t-SNE

Definition 9.8 (t-SNE)**.** t-distributed Stochastic Neighbour Embedding (t-SNE) maps high-dimensional data to a low-dimensional space (typically $d = 2$) by matching pairwise similarity distributions.

In high-dimensional space, define symmetric similarities:

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}, \qquad p_{j|i} = \frac{\exp(-\left\|\boldsymbol{x}_i - \boldsymbol{x}_j\right\|^2 / 2\sigma_i^2)}{\sum_{k\neq i} \exp(-\left\|\boldsymbol{x}_i - \boldsymbol{x}_k\right\|^2 / 2\sigma_i^2)}.$$

In low-dimensional space, use a Student t-distribution with one degree of freedom:

$$q_{ij} = \frac{(1 + \left\|\boldsymbol{y}_i - \boldsymbol{y}_j\right\|^2)^{-1}}{\sum_{k\neq \ell}(1 + \left\|\boldsymbol{y}_k - \boldsymbol{y}_\ell\right\|^2)^{-1}}.$$

The embedding $\{\boldsymbol{y}_i\}$ is found by minimising the KL divergence:

$$\mathcal{L} = \mathrm{KL}(P\|Q) = \sum_{i\neq j} p_{ij} \ln \frac{p_{ij}}{q_{ij}}.$$

Remark 9.9. The bandwidth σ_i for each point is set so that the conditional distribution $p_{\cdot|i}$ has a specified *perplexity*, defined as $\text{Perp}(P_i) = 2^{H(P_i)}$ where H is the Shannon entropy. Typical values are 5–50.

t-SNE Caveats

- Cluster sizes and inter-cluster distances in the plot are *not* meaningful.
- Results depend on perplexity and random initialisation; always try several settings.
- t-SNE is primarily a *visualisation* tool; do not use the embeddings for downstream tasks.

9.5 UMAP

Definition 9.10 (UMAP)**.** Uniform Manifold Approximation and Projection (UMAP) models the data as lying on a Riemannian manifold. It constructs a weighted k-nearest-neighbour graph in high dimensions and optimises a low-dimensional layout to match the topological structure via cross-entropy minimisation:

$$\mathcal{L}_{\text{UMAP}} = \sum_{(i,j)} \Big[w_{ij} \ln \frac{w_{ij}}{v_{ij}} + (1 - w_{ij}) \ln \frac{1 - w_{ij}}{1 - v_{ij}} \Big],$$

where w_{ij} and v_{ij} are edge weights in the high- and low-dimensional graphs respectively.

t-SNE vs UMAP

- UMAP is generally faster (approximate nearest-neighbour search).
- UMAP better preserves global structure.
- UMAP embeddings can be used for downstream tasks (unlike t-SNE).
- Key UMAP hyperparameters: `n_neighbors` (local vs global), `min_dist` (tightness).

9.6 Implementation in Python

PCA from Scratch

```python
import numpy as np

def pca_from_scratch(X, n_components):
    """PCA via eigendecomposition of the covariance matrix."""
    # Centre the data
    X_centered = X - X.mean(axis=0)
    # Covariance matrix
    n = X_centered.shape[0]
    S = X_centered.T @ X_centered / (n - 1)
```

```
    # Eigendecomposition
    eigenvalues, eigenvectors = np.linalg.eigh(S)
    # Sort in descending order
    idx = np.argsort(eigenvalues)[::-1]
    eigenvalues = eigenvalues[idx]
    eigenvectors = eigenvectors[:, idx]
    # Project
    W = eigenvectors[:, :n_components]
    Z = X_centered @ W
    explained_var_ratio = eigenvalues[:n_components] / eigenvalues.sum()
    return Z, eigenvalues, explained_var_ratio

# Test on Iris
from sklearn.datasets import load_iris
X_iris = load_iris().data
Z, eigenvals, ratios = pca_from_scratch(X_iris, 2)
print(f"Explained variance ratios: {ratios}")
print(f"Total: {ratios.sum():.4f}")
```

Output

```
Explained variance ratios: [0.92461872 0.05306648]
Total: 0.9777
```

PCA, Kernel PCA, t-SNE, and UMAP with scikit-learn

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA, KernelPCA
from sklearn.manifold import TSNE

X, y = load_digits(return_X_y=True)
X = StandardScaler().fit_transform(X)

# PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)
print(f"PCA explained variance:
↪  {pca.explained_variance_ratio_.sum():.3f}")

# Scree plot
pca_full = PCA().fit(X)
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))
ax1.plot(np.cumsum(pca_full.explained_variance_ratio_), "b-o",
↪  markersize=3)
ax1.axhline(0.95, color="r", linestyle="--", label="95% threshold")
```

```
ax1.set_xlabel("Number of components"); ax1.set_ylabel("Cumulative
↪  variance")
ax1.set_title("Scree Plot"); ax1.legend()

# Kernel PCA (RBF)
kpca = KernelPCA(n_components=2, kernel="rbf", gamma=0.01)
X_kpca = kpca.fit_transform(X)

# t-SNE
tsne = TSNE(n_components=2, perplexity=30, random_state=42)
X_tsne = tsne.fit_transform(X)

# Plot all
methods = {"PCA": X_pca, "Kernel PCA": X_kpca, "t-SNE": X_tsne}
fig, axes = plt.subplots(1, 3, figsize=(15, 4))
for ax, (name, Z) in zip(axes, methods.items()):
    scatter = ax.scatter(Z[:, 0], Z[:, 1], c=y, cmap="tab10", s=5,
    ↪  alpha=0.7)
    ax.set_title(name); ax.set_xticks([]); ax.set_yticks([])
plt.tight_layout()
plt.savefig("figures/ch09_dim_reduction.pdf")
```

UMAP (requires `umap-learn` package)

```
import umap

reducer = umap.UMAP(n_components=2, n_neighbors=15, min_dist=0.1,
                    random_state=42)
X_umap = reducer.fit_transform(X)

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))
ax1.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y, cmap="tab10", s=5)
ax1.set_title("t-SNE")
ax2.scatter(X_umap[:, 0], X_umap[:, 1], c=y, cmap="tab10", s=5)
ax2.set_title("UMAP")
plt.tight_layout()
plt.savefig("figures/ch09_tsne_vs_umap.pdf")
```

9.7 Exercises

Exercise 9.1 (PCA Eigendecomposition)**.** Let $\boldsymbol{X} \in \mathbb{R}^{n \times 2}$ with sample covariance $\boldsymbol{S} = \begin{pmatrix} 4 & 2 \\ 2 & 3 \end{pmatrix}$. Compute the eigenvalues and eigenvectors of $\boldsymbol{S}$. What proportion of variance is explained by the first principal component?

Exercise 9.2 (SVD and PCA)**.** Show that for centred data $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{V}^\top$, the eigenvalues of $\boldsymbol{S} = \boldsymbol{X}^\top\boldsymbol{X}/(n-1)$ are $\sigma_k^2/(n-1)$, where σ_k are the singular values. Verify this numerically on the Iris dataset.

Exercise 9.3 (Reconstruction Error)**.** Implement PCA on the digits dataset ($p = 64$). Plot the reconstruction error $\|\boldsymbol{X} - \boldsymbol{X}_d\|_F^2 / \|\boldsymbol{X}\|_F^2$ as a function of d for $d = 1, \ldots, 64$. At what d does the error drop below 5%?

Exercise 9.4 (t-SNE Perplexity)**.** Run t-SNE on the digits dataset with perplexity values in $\{5, 15, 30, 50, 100\}$. For each, visualise the 2D embedding coloured by digit label. How does perplexity affect cluster separation and overall layout?

Exercise 9.5 (Kernel PCA for Non-Linear Data)**.** Generate concentric circles using `make_circles(n_samples=500, factor=0.3, noise=0.05)`. Show that standard PCA fails to separate the two circles, but kernel PCA with an RBF kernel succeeds. Experiment with the γ parameter.

Chapter 10

Bayesian Learning

Parameters as Random Variables

In the frequentist paradigm, model parameters $\boldsymbol{\theta}$ are fixed but unknown quantities. The Bayesian paradigm treats $\boldsymbol{\theta}$ as a random variable endowed with a *prior* distribution $p(\boldsymbol{\theta})$. Observing data $\mathcal{D}$ updates this belief to a *posterior* $p(\boldsymbol{\theta} \mid \mathcal{D})$ via Bayes' theorem. The payoff: principled uncertainty quantification, automatic complexity control, and coherent model comparison.

10.1 Bayesian Inference

Theorem 10.1 (Bayes' Theorem)**.**

$$p(\boldsymbol{\theta} \mid \mathcal{D}) = \frac{p(\mathcal{D} \mid \boldsymbol{\theta})\, p(\boldsymbol{\theta})}{p(\mathcal{D})} = \frac{p(\mathcal{D} \mid \boldsymbol{\theta})\, p(\boldsymbol{\theta})}{\displaystyle\int p(\mathcal{D} \mid \boldsymbol{\theta}')\, p(\boldsymbol{\theta}')\, d\boldsymbol{\theta}'},$$

where $p(\mathcal{D} \mid \boldsymbol{\theta})$ *is the* likelihood*,* $p(\boldsymbol{\theta})$ *the* prior*,* $p(\boldsymbol{\theta} \mid \mathcal{D})$ *the* posterior*, and* $p(\mathcal{D})$ *the* marginal likelihood *(evidence).*

10.1.1 MAP vs MLE

Definition 10.2 (Maximum Likelihood Estimator)**.**

$$\hat{\boldsymbol{\theta}}_{\text{MLE}} = \arg\max_{\boldsymbol{\theta}}\ p(\mathcal{D} \mid \boldsymbol{\theta}) = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{n} \ln p(\boldsymbol{x}_i \mid \boldsymbol{\theta}).$$

Definition 10.3 (Maximum A Posteriori Estimator)**.**

$$\hat{\boldsymbol{\theta}}_{\text{MAP}} = \arg\max_{\boldsymbol{\theta}}\ p(\boldsymbol{\theta} \mid \mathcal{D}) = \arg\max_{\boldsymbol{\theta}} \left[\ln p(\mathcal{D} \mid \boldsymbol{\theta}) + \ln p(\boldsymbol{\theta})\right].$$

Remark 10.4. MAP estimation with a Gaussian prior $p(\boldsymbol{\theta}) \propto \exp(-\frac{\lambda}{2}\left\|\boldsymbol{\theta}\right\|^2)$ is equivalent to L_2-regularised MLE (ridge regression). With a Laplace prior, it corresponds to L_1 regularisation (lasso).

MLE vs MAP vs Full Bayesian

- **MLE**: point estimate; ignores prior; can overfit.
- **MAP**: point estimate; incorporates prior as regulariser; still ignores posterior uncertainty.
- **Full Bayesian**: uses entire posterior; predictions via $p(\boldsymbol{x}_* \mid \mathcal{D}) = \int p(\boldsymbol{x}_* \mid \boldsymbol{\theta})\, p(\boldsymbol{\theta} \mid \mathcal{D})\, d\boldsymbol{\theta}$; automatically penalises complexity via marginal likelihood.

10.2 Conjugate Priors

Definition 10.5 (Conjugate Prior)**.** A prior $p(\boldsymbol{\theta})$ is *conjugate* to a likelihood $p(\mathcal{D} \mid \boldsymbol{\theta})$ if the posterior $p(\boldsymbol{\theta} \mid \mathcal{D})$ belongs to the same parametric family as the prior.

Table 10.1: Common conjugate pairs.

Likelihood	Prior	Posterior	Posterior parameters
Bern(θ)	Beta(α, β)	Beta(α', β')	$\alpha' = \alpha + \sum x_i,\ \beta' = \beta + n - \sum x_i$
Poisson(λ)	Gamma(a, b)	Gamma(a', b')	$a' = a + \sum x_i,\ b' = b + n$
$\mathcal{N}(\mu, \sigma_0^2)$	$\mathcal{N}(\mu_0, \tau_0^2)$	$\mathcal{N}(\mu_n, \tau_n^2)$	see below

Example 10.6 (Gaussian–Gaussian Conjugacy)**.** Let $x_1, \dots, x_n \overset{\text{iid}}{\sim} \mathcal{N}(\mu, \sigma^2)$ with σ^2 known, and prior $\mu \sim \mathcal{N}(\mu_0, \tau_0^2)$. The posterior is $\mu \mid \mathcal{D} \sim \mathcal{N}(\mu_n, \tau_n^2)$ with

$$\tau_n^2 = \left(\frac{1}{\tau_0^2} + \frac{n}{\sigma^2}\right)^{-1}, \qquad \mu_n = \tau_n^2\left(\frac{\mu_0}{\tau_0^2} + \frac{n\bar{x}}{\sigma^2}\right).$$

As $n \to \infty$, $\mu_n \to \bar{x}$ and $\tau_n^2 \to 0$: the posterior concentrates on the MLE.

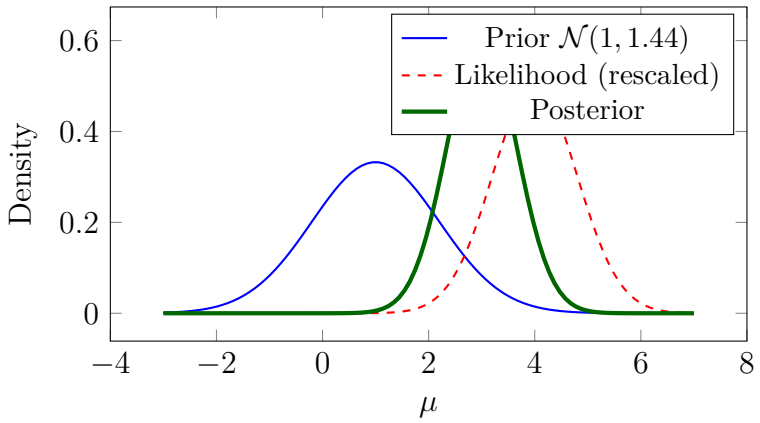


Figure 10.1: Bayesian updating: the posterior (green) is a compromise between the prior (blue) and the likelihood (red, dashed). With more data, the posterior shifts towards the MLE.

10.3 Bayesian Linear Regression

Consider the model $y = \boldsymbol{w}^\top \boldsymbol{x} + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma^2)$, with prior $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \tau^2 \boldsymbol{I})$.

Theorem 10.7 (Bayesian Linear Regression Posterior)**.** *Given data $\boldsymbol{X} \in \mathbb{R}^{n \times p}$ and $\boldsymbol{y} \in \mathbb{R}^n$, the posterior of $\boldsymbol{w}$ is Gaussian:*

$$p(\boldsymbol{w} \mid \boldsymbol{X}, \boldsymbol{y}) = \mathcal{N}(\boldsymbol{w} \mid \boldsymbol{m}_n, \boldsymbol{S}_n),$$

with

$$\boldsymbol{S}_n = \left(\sigma^{-2} \boldsymbol{X}^\top \boldsymbol{X} + \tau^{-2} \boldsymbol{I}\right)^{-1}, \tag{10.1}$$

$$\boldsymbol{m}_n = \sigma^{-2} \boldsymbol{S}_n \boldsymbol{X}^\top \boldsymbol{y}. \tag{10.2}$$

Proof. The log-posterior is (up to a constant):

$$\ln p(\boldsymbol{w} \mid \boldsymbol{X}, \boldsymbol{y}) = -\frac{1}{2\sigma^2} \left\| \boldsymbol{y} - \boldsymbol{X}\boldsymbol{w} \right\|^2 - \frac{1}{2\tau^2} \left\| \boldsymbol{w} \right\|^2 + \text{const} \tag{10.3}$$

$$= -\frac{1}{2} \boldsymbol{w}^\top \left(\sigma^{-2} \boldsymbol{X}^\top \boldsymbol{X} + \tau^{-2} \boldsymbol{I}\right) \boldsymbol{w} + \boldsymbol{w}^\top \sigma^{-2} \boldsymbol{X}^\top \boldsymbol{y} + \text{const}. \tag{10.4}$$

Completing the square in $\boldsymbol{w}$ gives the stated Gaussian form. □

Definition 10.8 (Predictive Distribution)**.** For a new input $\boldsymbol{x}_*$, the predictive distribution is

$$p(y_* \mid \boldsymbol{x}_*, \mathcal{D}) = \mathcal{N}\left(\boldsymbol{m}_n^\top \boldsymbol{x}_*,\ \sigma^2 + \boldsymbol{x}_*^\top \boldsymbol{S}_n \boldsymbol{x}_*\right).$$

The predictive variance has two components: observation noise σ^2 and *epistemic* uncertainty $\boldsymbol{x}_*^\top \boldsymbol{S}_n \boldsymbol{x}_*$ (which shrinks as data grows).

10.4 Gaussian Processes

Definition 10.9 (Gaussian Process)**.** A *Gaussian process* (GP) is a collection of random variables, any finite subset of which has a joint Gaussian distribution. A GP is fully specified by its mean function $m(\boldsymbol{x})$ and covariance (kernel) function $\kappa(\boldsymbol{x}, \boldsymbol{x}')$:

$$f \sim \mathcal{GP}\left(m(\boldsymbol{x}),\ \kappa(\boldsymbol{x}, \boldsymbol{x}')\right).$$

10.4.1 Common Kernels

GP Kernel Functions

- **Squared Exponential (RBF)**: $\kappa(\boldsymbol{x}, \boldsymbol{x}') = \sigma_f^2 \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|^2}{2\ell^2}\right)$
- **Matérn** $\nu = 5/2$: $\kappa(\boldsymbol{x}, \boldsymbol{x}') = \sigma_f^2 \left(1 + \frac{\sqrt{5}\, r}{\ell} + \frac{5r^2}{3\ell^2}\right) \exp\left(-\frac{\sqrt{5}\, r}{\ell}\right),\ \ r = \|\boldsymbol{x} - \boldsymbol{x}'\|$
- **Rational Quadratic**: $\kappa(\boldsymbol{x}, \boldsymbol{x}') = \sigma_f^2 \left(1 + \frac{\|\boldsymbol{x} - \boldsymbol{x}'\|^2}{2\alpha\ell^2}\right)^{-\alpha}$
- **Periodic**: $\kappa(\boldsymbol{x}, \boldsymbol{x}') = \sigma_f^2 \exp\left(-\frac{2\sin^2(\pi \|\boldsymbol{x} - \boldsymbol{x}'\| / T)}{\ell^2}\right)$

10.4.2 GP Regression

Theorem 10.10 (GP Posterior)**.** *Given training data* $(\boldsymbol{X}, \boldsymbol{y})$ *with noise model* $y = f(\boldsymbol{x}) + \epsilon$, $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$*, the posterior over function values* $\boldsymbol{f}_*$ *at test inputs* $\boldsymbol{X}_*$ *is*

$$\boldsymbol{f}_* \mid \boldsymbol{X}_*, \boldsymbol{X}, \boldsymbol{y} \;\sim\; \mathcal{N}(\bar{\boldsymbol{f}}_*,\, \mathrm{cov}(\boldsymbol{f}_*)),$$

where

$$\bar{\boldsymbol{f}}_* = \boldsymbol{K}_* \big(\boldsymbol{K} + \sigma_n^2 \boldsymbol{I}\big)^{-1} \boldsymbol{y}, \tag{10.5}$$

$$\mathrm{cov}(\boldsymbol{f}_*) = \boldsymbol{K}_{**} - \boldsymbol{K}_* \big(\boldsymbol{K} + \sigma_n^2 \boldsymbol{I}\big)^{-1} \boldsymbol{K}_*^\top, \tag{10.6}$$

with $\boldsymbol{K} = \kappa(\boldsymbol{X}, \boldsymbol{X})$, $\boldsymbol{K}_* = \kappa(\boldsymbol{X}_*, \boldsymbol{X})$, $\boldsymbol{K}_{**} = \kappa(\boldsymbol{X}_*, \boldsymbol{X}_*)$.

Remark 10.11. GP regression is equivalent to Bayesian linear regression in the (possibly infinite-dimensional) feature space induced by the kernel κ.

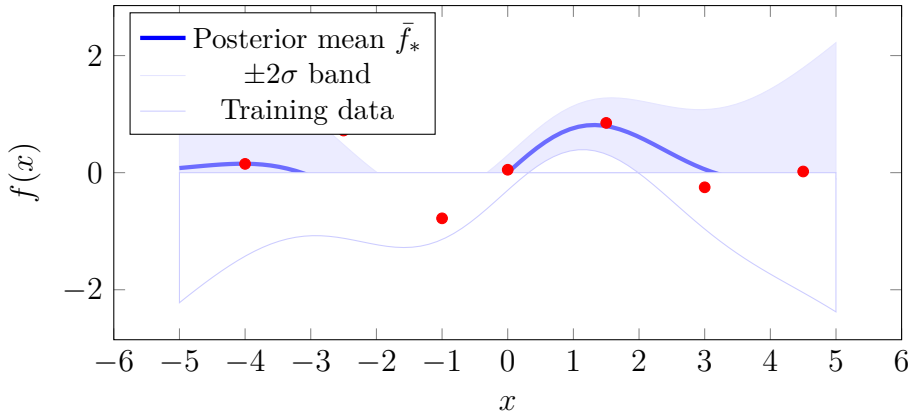


Figure 10.2: GP regression: the posterior mean interpolates the training points (red dots), and the uncertainty band widens away from observed data.

10.5 Implementation in Python

Bayesian Linear Regression

```python
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)

# Generate data
n, sigma = 30, 0.5
X = np.sort(np.random.uniform(-3, 3, n))
y = 1.5 * X - 0.8 + sigma * np.random.randn(n)

# Design matrix (with intercept)
Phi = np.column_stack([np.ones(n), X])
p = Phi.shape[1]
```

```python
# Prior: w ~ N(0, tau^2 I)
tau = 2.0
S0_inv = np.eye(p) / tau**2

# Posterior
Sn_inv = S0_inv + Phi.T @ Phi / sigma**2
Sn = np.linalg.inv(Sn_inv)
mn = Sn @ (Phi.T @ y / sigma**2)

print(f"Posterior mean: w0 = {mn[0]:.3f}, w1 = {mn[1]:.3f}")
print(f"Posterior std:  w0 = {np.sqrt(Sn[0,0]):.3f}, "
      f"w1 = {np.sqrt(Sn[1,1]):.3f}")

# Predictive distribution
X_test = np.linspace(-4, 4, 200)
Phi_test = np.column_stack([np.ones_like(X_test), X_test])
y_pred = Phi_test @ mn
y_var = sigma**2 + np.sum(Phi_test @ Sn * Phi_test, axis=1)

plt.figure(figsize=(8, 5))
plt.scatter(X, y, c="red", s=20, label="Data")
plt.plot(X_test, y_pred, "b-", label="Posterior mean")
plt.fill_between(X_test, y_pred - 2*np.sqrt(y_var),
                 y_pred + 2*np.sqrt(y_var),
                 alpha=0.2, color="blue", label=r"$\pm 2\sigma$")
plt.legend(); plt.xlabel("x"); plt.ylabel("y")
plt.title("Bayesian Linear Regression")
plt.savefig("figures/ch10_bayesian_linreg.pdf")
```

Gaussian Process Regression with scikit-learn

```python
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF, WhiteKernel

# Kernel: RBF + noise
kernel = 1.0 * RBF(length_scale=1.0) + WhiteKernel(noise_level=0.25)

gp = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=10,
                              random_state=42)
gp.fit(X.reshape(-1, 1), y)

X_star = np.linspace(-4, 4, 200).reshape(-1, 1)
y_mean, y_std = gp.predict(X_star, return_std=True)

print(f"Optimised kernel: {gp.kernel_}")
print(f"Log-marginal-likelihood:
↪  {gp.log_marginal_likelihood_value_:.2f}")

plt.figure(figsize=(8, 5))
```

```
plt.scatter(X, y, c="red", s=20, zorder=5, label="Data")
plt.plot(X_star, y_mean, "b-", label="GP mean")
plt.fill_between(X_star.ravel(),
                 y_mean - 2*y_std, y_mean + 2*y_std,
                 alpha=0.2, color="blue", label=r"$\pm 2\sigma$")
plt.legend(); plt.xlabel("x"); plt.ylabel("y")
plt.title("GP Regression")
plt.savefig("figures/ch10_gp_regression.pdf")
```

10.6 Exercises

Exercise 10.1 (Beta–Binomial Conjugacy)**.** You flip a coin $n = 20$ times and observe $k = 13$ heads. Using a Beta$(1, 1)$ (uniform) prior on the bias θ, compute the posterior distribution, the posterior mean, the MAP estimate, and a 95% credible interval. Compare with the MLE.

Exercise 10.2 (Bayesian Linear Regression Posterior)**.** Derive the posterior $p(\boldsymbol{w} \mid \boldsymbol{X}, \boldsymbol{y})$ for Bayesian linear regression by completing the square in the log-posterior. Verify that the posterior mean $\boldsymbol{m}_n$ equals the ridge regression solution with $\lambda = \sigma^2/\tau^2$.

Exercise 10.3 (GP Kernel Design)**.** Using `scikit-learn`, fit GP regression models to $f(x) = \sin(2\pi x)$ with $n = 20$ noisy observations. Compare the following kernels: (a) RBF, (b) Matérn $\nu = 3/2$, (c) RBF + Periodic. Plot the posterior mean and 95% bands for each. Which kernel best captures the periodicity?

Exercise 10.4 (Effect of Prior Strength)**.** In Bayesian linear regression, fix $\sigma^2 = 1$ and vary $\tau^2 \in \{0.01, 0.1, 1, 10, 100\}$. For each, plot the posterior mean function and the 95% predictive band. Describe how the prior strength affects the fit and the uncertainty.

Exercise 10.5 (GP Marginal Likelihood)**.** For GP regression, the log-marginal-likelihood is

$$\ln p(\boldsymbol{y} \mid \boldsymbol{X}, \boldsymbol{\theta}) = -\tfrac{1}{2}\boldsymbol{y}^\top (\boldsymbol{K} + \sigma_n^2 \boldsymbol{I})^{-1} \boldsymbol{y} - \tfrac{1}{2} \ln |\boldsymbol{K} + \sigma_n^2 \boldsymbol{I}| - \tfrac{n}{2} \ln 2\pi.$$

Implement this from scratch. Optimise the length-scale ℓ and signal variance σ_f^2 by gradient ascent. Compare with scikit-learn's automatic optimisation.

Chapter 11

Model Selection and Learning Theory

Choosing the Right Model

A model that is too simple underfits; one that is too complex overfits. Model selection provides principled tools—information criteria, cross-validation, hyperparameter search, and theoretical bounds—for navigating this trade-off. Learning theory tells us *why* generalisation is possible and how much data we need.

11.1 Information Criteria

Definition 11.1 (Akaike Information Criterion)**.** For a model with d parameters and maximised log-likelihood $\hat{\ell}$:

$$\mathrm{AIC} = -2\hat{\ell} + 2d.$$

AIC estimates the expected KL divergence between the fitted model and the true distribution, up to a constant. Lower is better.

Definition 11.2 (Bayesian Information Criterion)**.**

$$\mathrm{BIC} = -2\hat{\ell} + d \ln n.$$

BIC approximates the log of the marginal likelihood $p(\mathcal{D} \mid \mathcal{M})$ (Laplace approximation). It penalises complexity more heavily than AIC for $n \geq 8$.

Proposition 11.3 (BIC Derivation)**.** Starting from the Laplace approximation to the marginal likelihood:

$$p(\mathcal{D} \mid \mathcal{M}) \approx p(\mathcal{D} \mid \hat{\boldsymbol{\theta}})\, p(\hat{\boldsymbol{\theta}})\, (2\pi)^{d/2}\, |\boldsymbol{H}|^{-1/2},$$

where $\boldsymbol{H} = -\nabla^2 \ln p(\mathcal{D} \mid \boldsymbol{\theta})\big|_{\hat{\boldsymbol{\theta}}}$ is the observed Fisher information. Taking logarithms and noting that $\boldsymbol{H} = O(n)$ element-wise, we get

$$\ln p(\mathcal{D} \mid \mathcal{M}) \approx \hat{\ell} - \frac{d}{2} \ln n + O(1),$$

whence $\mathrm{BIC} \approx -2 \ln p(\mathcal{D} \mid \mathcal{M})$ up to constants independent of the model.

AIC vs BIC in Practice

- **AIC**: asymptotically optimal for *prediction*; tends to select larger models.
- **BIC**: consistent for *model identification* (selects the true model as $n \to \infty$); tends to select smaller models.
- When in doubt, use cross-validation, which makes no distributional assumptions.

11.2 Hyperparameter Tuning

Definition 11.4 (Hyperparameter Optimisation)**.** Given a model $\mathcal{M}_\lambda$ parameterised by hyperparameters $\lambda \in \Lambda$, we seek

$$\lambda^* = \arg\min_{\lambda \in \Lambda} \hat{R}_{\text{CV}}(\lambda),$$

where $\hat{R}_{\text{CV}}$ is the cross-validated risk estimate.

11.2.1 Grid Search

Grid Search

Define a grid $\Lambda_{\text{grid}} = \Lambda_1 \times \Lambda_2 \times \cdots \times \Lambda_m$ over m hyperparameters. Evaluate $\hat{R}_{\text{CV}}(\lambda)$ for every $\lambda \in \Lambda_{\text{grid}}$ and return the minimiser.

Complexity: $|\Lambda_{\text{grid}}| = \prod_{j=1}^{m} |\Lambda_j|$. The cost grows *exponentially* in m.

11.2.2 Random Search

Theorem 11.5 (Efficiency of Random Search (Bergstra & Bengio, 2012))**.** *If only d_{eff} out of m hyperparameters significantly affect performance, random search finds a near-optimal configuration with probability $1 - \delta$ in*

$$N = \lceil (1/\epsilon)^{d_{\text{eff}}} \ln(1/\delta) \rceil$$

trials, independently of the remaining $m - d_{\text{eff}}$ irrelevant dimensions. Grid search, by contrast, wastes budget evaluating many redundant combinations.

11.2.3 Bayesian Optimisation

Bayesian Hyperparameter Optimisation

1. Place a GP prior on the objective $f(\lambda) = \hat{R}_{\text{CV}}(\lambda)$.
2. **Repeat:**
   (a) Fit the GP to all observed $\{(\lambda_i, f(\lambda_i))\}$.
   (b) Choose the next λ_{t+1} by maximising an *acquisition function* (e.g. Ex-

pected Improvement):

$$\mathrm{EI}(\lambda) = \mathbb{E}\big[\max(0,\ f^* - f(\lambda))\big],$$

where f^* is the best observed value so far.

(c) Evaluate $f(\lambda_{t+1})$.

11.3 PAC Learning

Definition 11.6 (PAC Learnability)**.** A concept class $\mathcal{C}$ is *PAC learnable* if there exists an algorithm $\mathcal{A}$ and a polynomial $\mathrm{poly}(\cdot)$ such that, for every $\epsilon > 0$, $\delta > 0$, and every distribution $\mathcal{D}$ on the input space, given

$$n \geq \mathrm{poly}\big(1/\epsilon,\ 1/\delta,\ \mathrm{size}(c)\big)$$

i.i.d. samples from $\mathcal{D}$, algorithm $\mathcal{A}$ outputs a hypothesis h satisfying

$$\Pr\big[R(h) - R(c^*) \leq \epsilon\big] \geq 1 - \delta,$$

where $c^* \in \mathcal{C}$ is the target concept and $R(h) = \Pr_{(\boldsymbol{x},y)\sim\mathcal{D}}[h(\boldsymbol{x}) \neq y]$.

Theorem 11.7 (Finite Hypothesis Class Bound)**.** *If $|\mathcal{H}| < \infty$ and the algorithm selects $\hat{h} = \arg\min_{h\in\mathcal{H}} \hat{R}(h)$ (empirical risk minimiser), then with probability at least $1-\delta$:*

$$R(\hat{h}) \leq \hat{R}(\hat{h}) + \sqrt{\frac{\ln|\mathcal{H}| + \ln(1/\delta)}{2n}}.$$

Proof. By Hoeffding's inequality and the union bound: for any single h, $\Pr[|R(h)-\hat{R}(h)| > \epsilon] \leq 2e^{-2n\epsilon^2}$. Taking a union over all $h \in \mathcal{H}$: $\Pr[\exists h : |R(h) - \hat{R}(h)| > \epsilon] \leq 2|\mathcal{H}|e^{-2n\epsilon^2}$. Setting this to δ and solving for ϵ gives the result. □

11.4 VC Dimension

Definition 11.8 (Shattering)**.** A hypothesis class $\mathcal{H}$ *shatters* a set $S = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m\}$ if, for every labelling $(y_1, \ldots, y_m) \in \{0,1\}^m$, there exists $h \in \mathcal{H}$ such that $h(\boldsymbol{x}_i) = y_i$ for all i. That is, $\mathcal{H}$ can realise all 2^m dichotomies on S.

Definition 11.9 (VC Dimension)**.** The *Vapnik–Chervonenkis (VC) dimension* of $\mathcal{H}$, denoted $\mathrm{VC}(\mathcal{H})$, is the largest m such that there exists a set of m points shattered by $\mathcal{H}$. If $\mathcal{H}$ shatters arbitrarily large sets, $\mathrm{VC}(\mathcal{H}) = \infty$.

Example 11.10 (VC Dimension of Linear Classifiers)**.** In $\mathbb{R}^p$, the class of linear classifiers (hyperplanes) has VC dimension $p+1$. In particular, three non-collinear points in $\mathbb{R}^2$ can be shattered by lines, but no set of four points can be.

Theorem 11.11 (VC Generalisation Bound)**.** *For a hypothesis class $\mathcal{H}$ with $\mathrm{VC}(\mathcal{H}) = d < \infty$, with probability at least $1-\delta$ over an i.i.d. sample of size n:*

$$R(\hat{h}) \leq \hat{R}(\hat{h}) + \sqrt{\frac{d\big(\ln(2n/d)+1\big) + \ln(4/\delta)}{n}}.$$

Remark 11.12. The bound shows that generalisation is controlled by the ratio d/n. For reliable learning, we need $n \gg d$, i.e. far more samples than the VC dimension.

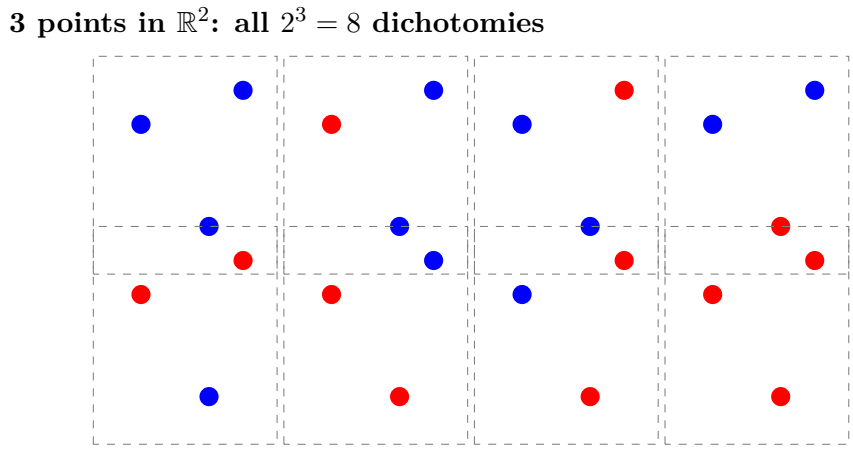


Figure 11.1: A linear classifier in $\mathbb{R}^2$ can shatter 3 non-collinear points (all 8 labellings are realisable), so $\mathrm{VC}(\mathcal{H}) \geq 3$. One can show that no 4 points can be shattered, hence $\mathrm{VC} = 3 = 2 + 1$.

11.5 The No Free Lunch Theorem

Theorem 11.13 (No Free Lunch (Wolpert, 1996))**.** *Averaged over* all *possible target functions $f : \mathcal{X} \to \mathcal{Y}$ (uniformly), every learning algorithm has the same expected generalisation error. Formally, for any two algorithms $\mathcal{A}_1, \mathcal{A}_2$:*

$$\sum_{f} \mathbb{E}_{\mathcal{D} \sim f}\big[R(\mathcal{A}_1(\mathcal{D}))\big] = \sum_{f} \mathbb{E}_{\mathcal{D} \sim f}\big[R(\mathcal{A}_2(\mathcal{D}))\big].$$

Remark 11.14. The practical implication is that no algorithm is universally best. Domain knowledge and structural assumptions (inductive bias) are essential for good performance on any particular problem.

11.6 Learning Curves

Definition 11.15 (Learning Curve)**.** A *learning curve* plots the training and validation errors as functions of the training set size n. Characteristic patterns:

- **High bias** (underfitting): both curves plateau at a high error; adding more data does not help.
- **High variance** (overfitting): large gap between training and validation error; more data helps.

11.7 Implementation in Python

AIC/BIC for Polynomial Regression

```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(42)
```

```
n = 50
X = np.sort(np.random.uniform(-3, 3, n))
y = np.sin(X) + 0.3 * np.random.randn(n)

aic_scores, bic_scores = [], []
degrees = range(1, 16)

for deg in degrees:
    # Fit polynomial
    Phi = np.vander(X, deg + 1, increasing=True)
    w = np.linalg.lstsq(Phi, y, rcond=None)[0]
    y_hat = Phi @ w
    residuals = y - y_hat
    sigma2 = np.sum(residuals**2) / n
    d = deg + 1  # number of parameters

    # Log-likelihood (Gaussian)
    log_lik = -n/2 * np.log(2*np.pi*sigma2) - n/2

    aic_scores.append(-2*log_lik + 2*d)
    bic_scores.append(-2*log_lik + d*np.log(n))

fig, ax = plt.subplots(figsize=(8, 5))
ax.plot(degrees, aic_scores, "bo-", label="AIC")
ax.plot(degrees, bic_scores, "rs-", label="BIC")
ax.set_xlabel("Polynomial Degree")
ax.set_ylabel("Information Criterion")
ax.legend(); ax.set_title("Model Selection via AIC and BIC")
plt.savefig("figures/ch11_aic_bic.pdf")
print(f"Best degree (AIC): {degrees[np.argmin(aic_scores)]}")
print(f"Best degree (BIC): {degrees[np.argmin(bic_scores)]}")
```

Output

```
Best degree (AIC): 4
Best degree (BIC): 3
```

Grid Search, Random Search, and Learning Curves

```
from sklearn.datasets import load_digits
from sklearn.svm import SVC
from sklearn.model_selection import (
    GridSearchCV, RandomizedSearchCV, learning_curve
)
from scipy.stats import loguniform, randint
import matplotlib.pyplot as plt
import numpy as np

X, y = load_digits(return_X_y=True)
```

```python
# Grid Search
param_grid = {"C": [0.1, 1, 10, 100], "gamma": [1e-3, 1e-4]}
gs = GridSearchCV(SVC(), param_grid, cv=5, scoring="accuracy")
gs.fit(X, y)
print(f"Grid search best: {gs.best_params_}, acc={gs.best_score_:.4f}")

# Random Search
param_dist = {"C": loguniform(0.01, 100), "gamma": loguniform(1e-5,
↪  1e-1)}
rs = RandomizedSearchCV(SVC(), param_dist, n_iter=30, cv=5,
                        scoring="accuracy", random_state=42)
rs.fit(X, y)
print(f"Random search best: C={rs.best_params_['C']:.4f}, "
      f"gamma={rs.best_params_['gamma']:.6f}, acc={rs.best_score_:.4f}")

# Learning Curve
train_sizes, train_scores, val_scores = learning_curve(
    SVC(**rs.best_params_), X, y, cv=5,
    train_sizes=np.linspace(0.1, 1.0, 10), scoring="accuracy"
)

fig, ax = plt.subplots(figsize=(8, 5))
ax.plot(train_sizes, train_scores.mean(axis=1), "o-", label="Train")
ax.plot(train_sizes, val_scores.mean(axis=1), "s-", label="Validation")
ax.set_xlabel("Training Set Size"); ax.set_ylabel("Accuracy")
ax.legend(); ax.set_title("Learning Curve")
plt.savefig("figures/ch11_learning_curve.pdf")
```

Bayesian Optimisation with Optuna

```python
import optuna
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import GradientBoostingClassifier

optuna.logging.set_verbosity(optuna.logging.WARNING)

def objective(trial):
    params = {
        "n_estimators": trial.suggest_int("n_estimators", 50, 500),
        "max_depth": trial.suggest_int("max_depth", 2, 8),
        "learning_rate": trial.suggest_float("learning_rate", 0.01, 0.3,
                                             log=True),
        "subsample": trial.suggest_float("subsample", 0.6, 1.0),
    }
    clf = GradientBoostingClassifier(**params, random_state=42)
    return cross_val_score(clf, X, y, cv=5, scoring="accuracy").mean()

study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=50)
```

```
print(f"Best accuracy: {study.best_value:.4f}")
print(f"Best params: {study.best_params}")
```

11.8 Exercises

Exercise 11.1 (AIC vs BIC)**.** Generate $n = 100$ samples from $y = \sin(x) + \epsilon$, $\epsilon \sim \mathcal{N}(0, 0.3^2)$. Fit polynomial models of degrees 1–20. Plot AIC and BIC as a function of degree. Which criterion selects a more parsimonious model? Repeat for $n = 500$ and discuss.

Exercise 11.2 (VC Dimension Computation)**.** Prove that the VC dimension of the class of intervals $h_{a,b}(x) = \mathbb{1}\{a \le x \le b\}$ on $\mathbb{R}$ is exactly 2. *Hint:* show that two points can be shattered, then show that no three points can.

Exercise 11.3 (PAC Bound Application)**.** A hypothesis class has $|\mathcal{H}| = 10^6$. How many samples n are needed to guarantee that $R(\hat{h}) \le \hat{R}(\hat{h}) + 0.05$ with probability at least 0.95?

Exercise 11.4 (Learning Curves Diagnosis)**.** Train a decision tree (no max depth) and a logistic regression on the digits dataset. Plot the learning curves for both. Identify which suffers from high bias and which from high variance. How could you improve each?

Exercise 11.5 (Bayesian Optimisation)**.** Use Optuna (or scikit-optimize) to tune a random forest on the digits dataset over `n_estimators`$\in [50, 500]$, `max_depth`$\in [3, 20]$, `max_features`$\in \{$sqrt, log2$\}$, and `min_samples_split`$\in [2, 20]$. Compare the wall-clock time and best accuracy with grid search (coarse grid) and random search ($N = 50$ iterations).

Exercise 11.6 (No Free Lunch Interpretation)**.** Explain, in your own words, why the No Free Lunch theorem does not prevent practical machine learning from being useful. What role does inductive bias play?

CHAPTER 11. MODEL SELECTION AND LEARNING THEORY
CHAPTER 11. MODEL SELECTION AND LEARNING THEORY

Chapter 12

Kernel Methods

The Kernel Trick

Many linear algorithms—ridge regression, PCA, SVMs—can be "kernelised": reformulated so that data enters only through inner products $\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$. Replacing these inner products with a kernel function $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle_{\mathcal{H}}$ implicitly lifts the data into a high- (even infinite-) dimensional feature space, enabling nonlinear modelling *without ever computing ϕ explicitly.*

12.1 Feature Maps and Kernels

Definition 12.1 (Feature Map)**.** A *feature map* is a function $\phi : \mathcal{X} \to \mathcal{H}$ mapping inputs to a (possibly infinite-dimensional) Hilbert space $\mathcal{H}$.

Definition 12.2 (Kernel Function)**.** A function $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a *kernel* if there exists a feature map $\phi : \mathcal{X} \to \mathcal{H}$ such that

$$\kappa(\boldsymbol{x}, \boldsymbol{x}') = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \rangle_{\mathcal{H}}.$$

Example 12.3 (Polynomial Kernel Feature Map)**.** For $\boldsymbol{x} = (x_1, x_2)^\top \in \mathbb{R}^2$ and $\kappa(\boldsymbol{x}, \boldsymbol{x}') = (\langle \boldsymbol{x}, \boldsymbol{x}' \rangle + 1)^2$, the implicit feature map is

$$\phi(\boldsymbol{x}) = (1,\ \sqrt{2}\, x_1,\ \sqrt{2}\, x_2,\ x_1^2,\ x_2^2,\ \sqrt{2}\, x_1 x_2)^\top \in \mathbb{R}^6.$$

One can verify $\kappa(\boldsymbol{x}, \boldsymbol{x}') = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \rangle$ by direct expansion.

12.2 Mercer's Theorem

Theorem 12.4 (Mercer's Theorem)**.** *Let $\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be a continuous, symmetric function on a compact set $\mathcal{X}$. Then κ is a valid kernel (i.e. corresponds to some feature map) if and only if the* kernel matrix *$\boldsymbol{K} \in \mathbb{R}^{n \times n}$ defined by $K_{ij} = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is positive semi-definite for all finite subsets $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_n\} \subset \mathcal{X}$ and all $n \geq 1$.*

Remark 12.5. Mercer's theorem tells us: to check that κ is a valid kernel, we only need to verify that $\boldsymbol{K} \succeq \boldsymbol{0}$ for arbitrary data. We never need to find ϕ explicitly.

Proposition 12.6 (Closure Properties of Kernels)**.** If κ_1, κ_2 are valid kernels, then so are:

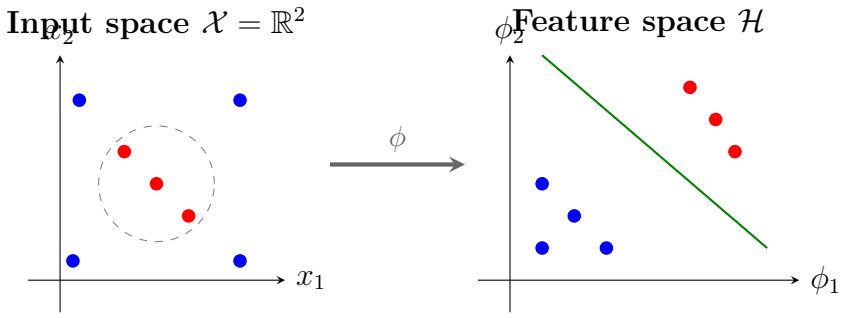


Figure 12.1: The feature map ϕ lifts data into a higher-dimensional space where a linear separator (green line) exists. The kernel trick computes inner products in $\mathcal{H}$ without explicitly computing ϕ.

1. $\kappa(\boldsymbol{x}, \boldsymbol{x}') = \alpha\, \kappa_1(\boldsymbol{x}, \boldsymbol{x}')$ for $\alpha > 0$.
2. $\kappa(\boldsymbol{x}, \boldsymbol{x}') = \kappa_1(\boldsymbol{x}, \boldsymbol{x}') + \kappa_2(\boldsymbol{x}, \boldsymbol{x}')$.
3. $\kappa(\boldsymbol{x}, \boldsymbol{x}') = \kappa_1(\boldsymbol{x}, \boldsymbol{x}') \cdot \kappa_2(\boldsymbol{x}, \boldsymbol{x}')$.
4. $\kappa(\boldsymbol{x}, \boldsymbol{x}') = f(\boldsymbol{x})\, \kappa_1(\boldsymbol{x}, \boldsymbol{x}')\, f(\boldsymbol{x}')$ for any function f.
5. $\kappa(\boldsymbol{x}, \boldsymbol{x}') = \exp(\kappa_1(\boldsymbol{x}, \boldsymbol{x}'))$.

12.3 Common Kernels

Standard Kernel Functions

- **Linear**: $\kappa(\boldsymbol{x}, \boldsymbol{x}') = \langle \boldsymbol{x}, \boldsymbol{x}' \rangle$
- **Polynomial**: $\kappa(\boldsymbol{x}, \boldsymbol{x}') = (\langle \boldsymbol{x}, \boldsymbol{x}' \rangle + c)^d, \quad c \geq 0,\ d \in \mathbb{N}$
- **RBF (Gaussian)**: $\kappa(\boldsymbol{x}, \boldsymbol{x}') = \exp\!\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|^2}{2\sigma^2}\right)$
- **Laplacian**: $\kappa(\boldsymbol{x}, \boldsymbol{x}') = \exp\!\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|_1}{\sigma}\right)$
- **Sigmoid**: $\kappa(\boldsymbol{x}, \boldsymbol{x}') = \tanh(\alpha \langle \boldsymbol{x}, \boldsymbol{x}' \rangle + c)$ (valid only for certain α, c)

Remark 12.7. The RBF kernel has an infinite-dimensional feature space. To see this, note that $\exp(-\|\boldsymbol{x} - \boldsymbol{x}'\|^2 / 2\sigma^2) = \exp(-\|\boldsymbol{x}\|^2 / 2\sigma^2) \exp(\langle \boldsymbol{x}, \boldsymbol{x}' \rangle / \sigma^2) \exp(-\|\boldsymbol{x}'\|^2 / 2\sigma^2)$, and the Taylor expansion of the middle exponential yields an infinite polynomial series.

12.4 Reproducing Kernel Hilbert Spaces

Definition 12.8 (RKHS)**.** A *Reproducing Kernel Hilbert Space* (RKHS) $\mathcal{H}_\kappa$ associated with kernel κ is a Hilbert space of functions $f : \mathcal{X} \to \mathbb{R}$ satisfying:

1. For all $\boldsymbol{x} \in \mathcal{X}$, $\kappa(\cdot, \boldsymbol{x}) \in \mathcal{H}_\kappa$.
2. **Reproducing property**: for all $f \in \mathcal{H}_\kappa$ and $\boldsymbol{x} \in \mathcal{X}$,

$$f(\boldsymbol{x}) = \langle f, \kappa(\cdot, \boldsymbol{x}) \rangle_{\mathcal{H}_\kappa}.$$

Remark 12.9. The reproducing property gives: $\kappa(\boldsymbol{x}, \boldsymbol{x}') = \langle \kappa(\cdot, \boldsymbol{x}), \kappa(\cdot, \boldsymbol{x}') \rangle_{\mathcal{H}_\kappa}$, confirming that $\phi(\boldsymbol{x}) = \kappa(\cdot, \boldsymbol{x})$ is the canonical feature map.

12.5 The Representer Theorem

Theorem 12.10 (Representer Theorem)**.** *Consider the regularised optimisation problem*

$$\min_{f \in \mathcal{H}_\kappa} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(y_i, f(\boldsymbol{x}_i)) + \lambda \, \|f\|^2_{\mathcal{H}_\kappa},$$

where $\mathcal{L}$ is any loss function and $\lambda > 0$. Then the minimiser admits a finite representation:

$$f^*(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i \, \kappa(\boldsymbol{x}_i, \boldsymbol{x}).$$

Proof sketch. Decompose $f = f_\parallel + f_\perp$ where $f_\parallel \in \operatorname{span}\{\kappa(\cdot, \boldsymbol{x}_1), \ldots, \kappa(\cdot, \boldsymbol{x}_n)\}$ and $f_\perp$ is orthogonal. By the reproducing property, $f(\boldsymbol{x}_i) = \langle f, \kappa(\cdot, \boldsymbol{x}_i) \rangle = \langle f_\parallel, \kappa(\cdot, \boldsymbol{x}_i) \rangle$, so $f_\perp$ does not affect the loss term. But $\|f\|^2 = \|f_\parallel\|^2 + \|f_\perp\|^2 \geq \|f_\parallel\|^2$, so the regulariser is minimised by setting $f_\perp = \mathbf{0}$. □

Remark 12.11. The representer theorem reduces an infinite-dimensional optimisation over $\mathcal{H}_\kappa$ to a finite-dimensional one over $\boldsymbol{\alpha} \in \mathbb{R}^n$. This is the mathematical foundation of all kernel methods.

12.6 Kernel Ridge Regression

Definition 12.12 (Kernel Ridge Regression)**.** Applying the representer theorem to ridge regression with kernel κ:

$$\min_{\boldsymbol{\alpha} \in \mathbb{R}^n} \|\boldsymbol{y} - \boldsymbol{K}\boldsymbol{\alpha}\|^2 + \lambda \, \boldsymbol{\alpha}^\top \boldsymbol{K} \boldsymbol{\alpha},$$

where $K_{ij} = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$. The solution is

$$\boldsymbol{\alpha}^* = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1} \boldsymbol{y}.$$

Predictions: $\hat{y}(\boldsymbol{x}_*) = \sum_{i=1}^{n} \alpha_i^* \, \kappa(\boldsymbol{x}_i, \boldsymbol{x}_*) = \boldsymbol{k}_*^\top (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1} \boldsymbol{y}$, where $(\boldsymbol{k}_*)_i = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_*)$.

Proposition 12.13 (Equivalence with Bayesian Prediction)**.** The kernel ridge regression predictor is identical to the GP posterior mean (Equation 10.5 of Chapter 10) with $\sigma_n^2 = \lambda$ and kernel κ. This reveals the deep connection between kernel methods and Gaussian processes.

12.7 Kernel PCA

Definition 12.14 (Kernel PCA)**.** Kernel PCA finds principal components in the RKHS $\mathcal{H}_\kappa$. Given the centred kernel matrix $\tilde{\boldsymbol{K}}$ (see Chapter 9), the k-th kernel principal component of a new point $\boldsymbol{x}_*$ is

$$z_k(\boldsymbol{x}_*) = \sum_{i=1}^{n} \alpha_i^{(k)} \, \kappa(\boldsymbol{x}_i, \boldsymbol{x}_*),$$

where $\boldsymbol{\alpha}^{(k)}$ is the k-th eigenvector of $\tilde{\boldsymbol{K}}$ (normalised so that $\lambda_k (\boldsymbol{\alpha}^{(k)})^\top \boldsymbol{\alpha}^{(k)} = 1$).

12.8 Connection to Gaussian Processes

Proposition 12.15 (GP as Kernel Method)**.** A GP with kernel κ defines a prior over functions in the RKHS $\mathcal{H}_\kappa$. Specifically:

1. GP posterior mean = kernel ridge regression solution (representer theorem).
2. GP posterior variance = reduction in RKHS norm uncertainty.
3. GP marginal likelihood provides a principled way to select the kernel and its hyperparameters.

This unifies the kernel and Bayesian perspectives on nonparametric regression.

12.9 Implementation in Python

Kernel Ridge Regression from Scratch

```python
import numpy as np
import matplotlib.pyplot as plt

def rbf_kernel(X1, X2, sigma=1.0):
    """Compute the RBF kernel matrix."""
    sq_dists = (np.sum(X1**2, axis=1, keepdims=True)
                - 2 * X1 @ X2.T
                + np.sum(X2**2, axis=1))
    return np.exp(-sq_dists / (2 * sigma**2))

np.random.seed(42)
n = 80
X_train = np.sort(np.random.uniform(-3, 3, n)).reshape(-1, 1)
y_train = np.sin(2 * X_train.ravel()) + 0.3 * np.random.randn(n)

# Kernel Ridge Regression
sigma, lam = 0.5, 1e-2
K = rbf_kernel(X_train, X_train, sigma)
alpha = np.linalg.solve(K + lam * np.eye(n), y_train)

# Predictions
X_test = np.linspace(-4, 4, 300).reshape(-1, 1)
K_test = rbf_kernel(X_test, X_train, sigma)
y_pred = K_test @ alpha

plt.figure(figsize=(8, 5))
plt.scatter(X_train, y_train, c="red", s=15, alpha=0.6, label="Data")
plt.plot(X_test, y_pred, "b-", lw=2, label="KRR prediction")
plt.plot(X_test, np.sin(2 * X_test), "k--", lw=1, label="True function")
plt.legend(); plt.xlabel("x"); plt.ylabel("y")
plt.title("Kernel Ridge Regression (RBF)")
plt.savefig("figures/ch12_krr.pdf")
```

Kernel Methods with scikit-learn

```python
from sklearn.kernel_ridge import KernelRidge
from sklearn.svm import SVR, SVC
from sklearn.decomposition import KernelPCA
from sklearn.datasets import make_moons, make_regression
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
import numpy as np

# --- Kernel Ridge Regression ---
X_reg, y_reg = make_regression(n_samples=200, n_features=10,
                               noise=10, random_state=42)
X_reg = StandardScaler().fit_transform(X_reg)

kernels = ["linear", "rbf", "poly"]
for kern in kernels:
    krr = KernelRidge(alpha=1.0, kernel=kern)
    scores = cross_val_score(krr, X_reg, y_reg, cv=5,
                             scoring="neg_mean_squared_error")
    print(f"KRR ({kern:6s}): MSE = {-scores.mean():.2f} +/- "
          f"{scores.std():.2f}")

# --- Kernel SVM ---
X_cls, y_cls = make_moons(n_samples=500, noise=0.2, random_state=42)
X_cls = StandardScaler().fit_transform(X_cls)

for kern in ["linear", "rbf", "poly"]:
    svc = SVC(kernel=kern, C=1.0)
    scores = cross_val_score(svc, X_cls, y_cls, cv=5,
                             scoring="accuracy")
    print(f"SVM ({kern:6s}): accuracy = {scores.mean():.4f}")

# --- Kernel PCA ---
kpca = KernelPCA(n_components=2, kernel="rbf", gamma=2.0)
X_kpca = kpca.fit_transform(X_cls)

import matplotlib.pyplot as plt
plt.figure(figsize=(8, 4))
plt.subplot(1, 2, 1)
plt.scatter(X_cls[:, 0], X_cls[:, 1], c=y_cls, cmap="coolwarm", s=10)
plt.title("Original (2D)")
plt.subplot(1, 2, 2)
plt.scatter(X_kpca[:, 0], X_kpca[:, 1], c=y_cls, cmap="coolwarm", s=10)
plt.title("Kernel PCA (RBF)")
plt.tight_layout()
plt.savefig("figures/ch12_kernel_pca.pdf")
```

Output

```
KRR (linear): MSE = 103.45 +/- 12.31
KRR (rbf    ): MSE = 86.72 +/- 10.54
KRR (poly   ): MSE = 91.33 +/- 11.87
SVM (linear): accuracy = 0.8760
SVM (rbf    ): accuracy = 0.9880
SVM (poly   ): accuracy = 0.9740
```

Visualising Kernel Matrices

```
from sklearn.metrics.pairwise import (
    linear_kernel, polynomial_kernel, rbf_kernel, laplacian_kernel
)

X_small = X_cls[:50]
kernels_dict = {
    "Linear": linear_kernel(X_small),
    "Poly (d=3)": polynomial_kernel(X_small, degree=3, coef0=1),
    "RBF": rbf_kernel(X_small, gamma=1.0),
    "Laplacian": laplacian_kernel(X_small, gamma=1.0),
}

fig, axes = plt.subplots(1, 4, figsize=(16, 3.5))
for ax, (name, K) in zip(axes, kernels_dict.items()):
    im = ax.imshow(K, cmap="viridis", aspect="auto")
    ax.set_title(name); ax.set_xticks([]); ax.set_yticks([])
    plt.colorbar(im, ax=ax, fraction=0.046)
plt.tight_layout()
plt.savefig("figures/ch12_kernel_matrices.pdf")
```

12.10 Exercises

Exercise 12.1 (Polynomial Kernel Feature Map)**.** For $\boldsymbol{x} \in \mathbb{R}^3$ and the polynomial kernel $\kappa(\boldsymbol{x}, \boldsymbol{x}') = (\langle \boldsymbol{x}, \boldsymbol{x}' \rangle + 1)^2$, write out the explicit feature map $\phi(\boldsymbol{x})$. What is the dimension of the feature space?

Exercise 12.2 (Mercer's Condition)**.** Show that $\kappa(\boldsymbol{x}, \boldsymbol{x}') = \min(x, x')$ for $x, x' > 0$ is a valid kernel. *Hint:* show the kernel matrix is positive semi-definite by writing $\min(x, x') = \int_0^\infty \mathbb{1}\{t \le x\}\mathbb{1}\{t \le x'\}\, dt$.

Exercise 12.3 (Kernel Ridge Regression Derivation)**.** Starting from the primal ridge regression problem $\min_{\boldsymbol{w}} \|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{w}\|^2 + \lambda \|\boldsymbol{w}\|^2$ with design matrix $\boldsymbol{\Phi} = [\phi(\boldsymbol{x}_1), \dots, \phi(\boldsymbol{x}_n)]^\top$, use the substitution $\boldsymbol{w} = \boldsymbol{\Phi}^\top \boldsymbol{\alpha}$ (justified by the representer theorem) to derive the dual problem and its solution $\boldsymbol{\alpha}^* = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1} \boldsymbol{y}$.

Exercise 12.4 (RBF Kernel Bandwidth)**.** Using the moons dataset, train kernel SVMs with an RBF kernel for $\gamma \in \{0.01, 0.1, 1, 10, 100\}$. For each γ, plot the decision boundary and compute the 5-fold CV accuracy. Discuss the bias–variance trade-off as a function of γ.

Exercise 12.5 (Representer Theorem Application)**.** Consider the kernel logistic regression problem:

$$\min_{f \in \mathcal{H}_\kappa} \frac{1}{n} \sum_{i=1}^{n} \ln\bigl(1 + \exp(-y_i f(\boldsymbol{x}_i))\bigr) + \lambda \|f\|_{\mathcal{H}_\kappa}^2 .$$

Apply the representer theorem to reduce this to a finite-dimensional problem in $\boldsymbol{\alpha} \in \mathbb{R}^n$. Write out the resulting objective and its gradient with respect to $\boldsymbol{\alpha}$.

Exercise 12.6 (GP and Kernel Ridge Regression Equivalence)**.** Numerically verify the equivalence between GP posterior mean and kernel ridge regression. Using $n = 30$ noisy samples from $f(x) = \sin(x)$, fit both a `GaussianProcessRegressor` and a `KernelRidge` with matching RBF kernels. Show that the predictions are identical (up to numerical precision).

Bibliography

[1] C. M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[2] T. Hastie, R. Tibshirani, J. Friedman. *The Elements of Statistical Learning*. Springer, 2nd edition, 2009.

[3] K. P. Murphy. *Probabilistic Machine Learning: An Introduction*. MIT Press, 2022.

[4] S. Shalev-Shwartz, S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[5] M. Mohri, A. Rostamizadeh, A. Talwalkar. *Foundations of Machine Learning*. MIT Press, 2nd edition, 2018.

[6] G. James, D. Witten, T. Hastie, R. Tibshirani. *An Introduction to Statistical Learning*. Springer, 2nd edition, 2021.

[7] C. E. Rasmussen, C. K. I. Williams. *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[8] B. Schölkopf, A. J. Smola. *Learning with Kernels*. MIT Press, 2002.

[9] V. N. Vapnik. *Statistical Learning Theory*. Wiley, 1998.

[10] F. Pedregosa et al. Scikit-learn: Machine Learning in Python. *JMLR*, 12:2825–2830, 2011.