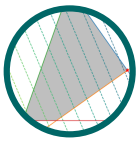
7

Continuous Optimization



Since machine learning algorithms are implemented on a computer, the mathematical formulations are expressed as numerical optimization methods. This chapter describes the basic numerical methods for training machine learning models. Training a machine learning model often boils down to finding a good set of parameters. The notion of "good" is determined by the objective function or the probabilistic model, which we will see examples of in the second part of this book. Given an objective function, finding the best value is done using optimization algorithms.

Since we consider data and models in $\mathbb{R}^D$, the optimization problems we face are *continuous* optimization problems, as opposed to *combinatorial* optimization problems for discrete variables.

This chapter covers two main branches of continuous optimization (Figure 7.1): unconstrained and constrained optimization. We will assume in this chapter that our objective function is differentiable (see Chapter 5), hence we have access to a gradient at each location in the space to help us find the optimum value. By convention, most objective functions in machine learning are intended to be minimized, that is, the best value is the minimum value. Intuitively finding the best value is like finding the valleys of the objective function, and the gradients point us uphill. The idea is to move downhill (opposite to the gradient) and hope to find the deepest point. For unconstrained optimization, this is the only concept we need, but there are several design choices, which we discuss in Section 7.1. For constrained optimization, we need to introduce other concepts to manage the constraints (Section 7.2). We will also introduce a special class of problems (convex optimization problems in Section 7.3) where we can make statements about reaching the global optimum.

global minimum

Consider the function in Figure 7.2. The function has a *global minimum* around $x = -4.5$, with a function value of approximately -47. Since the function is "smooth," the gradients can be used to help find the minimum by indicating whether we should take a step to the right or left. This assumes that we are in the correct bowl, as there exists another *local minimum* around $x = 0.7$. Recall that we can solve for all the stationary points of a function by calculating its derivative and setting it to zero. For

local minimum

Stationary points are the real roots of the derivative, that is, points that have zero gradient.

$$\ell(x) = x^4 + 7x^3 + 5x^2 - 17x + 3\,, \tag{7.1}$$

we obtain the corresponding gradient as

$$\frac{\mathrm{d}\ell(x)}{\mathrm{d}x} = 4x^3 + 21x^2 + 10x - 17\,. \tag{7.2}$$

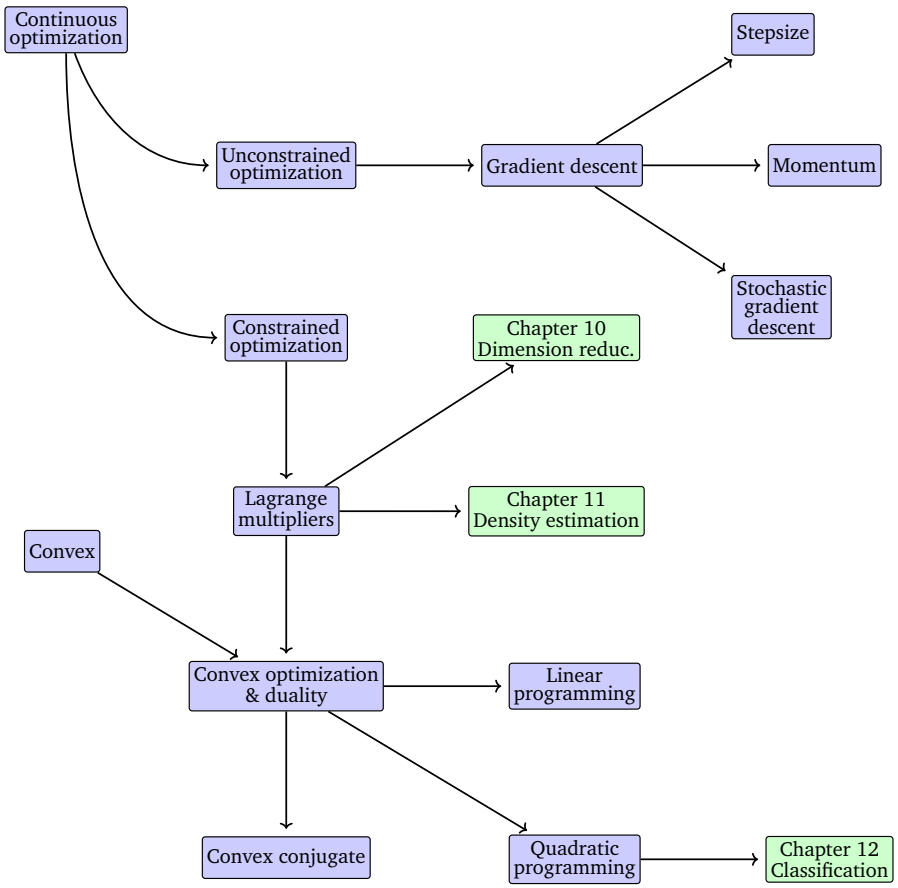


Figure 7.1 A mind map of the concepts related to optimization, as presented in this chapter. There are two main ideas: gradient descent and convex optimization.

Since this is a cubic equation, it has in general three solutions when set to zero. In the example, two of them are minimums and one is a maximum (around $x = -1.4$). To check whether a stationary point is a minimum or maximum, we need to take the derivative a second time and check whether the second derivative is positive or negative at the stationary point. In our case, the second derivative is

$$\frac{\mathrm{d}^2\ell(x)}{\mathrm{d}x^2} = 12x^2 + 42x + 10\,. \tag{7.3}$$

By substituting our visually estimated values of $x = -4.5, -1.4, 0.7$, we will observe that as expected the middle point is a maximum $\left(\frac{\mathrm{d}^2\ell(x)}{\mathrm{d}x^2} < 0\right)$ and the other two stationary points are minimums.

Note that we have avoided analytically solving for values of x in the previous discussion, although for low-order polynomials such as the preceding we could do so. In general, we are unable to find analytic solutions, and hence we need to start at some value, say $x_0 = -6$, and follow the negative gradient. The negative gradient indicates that we should go

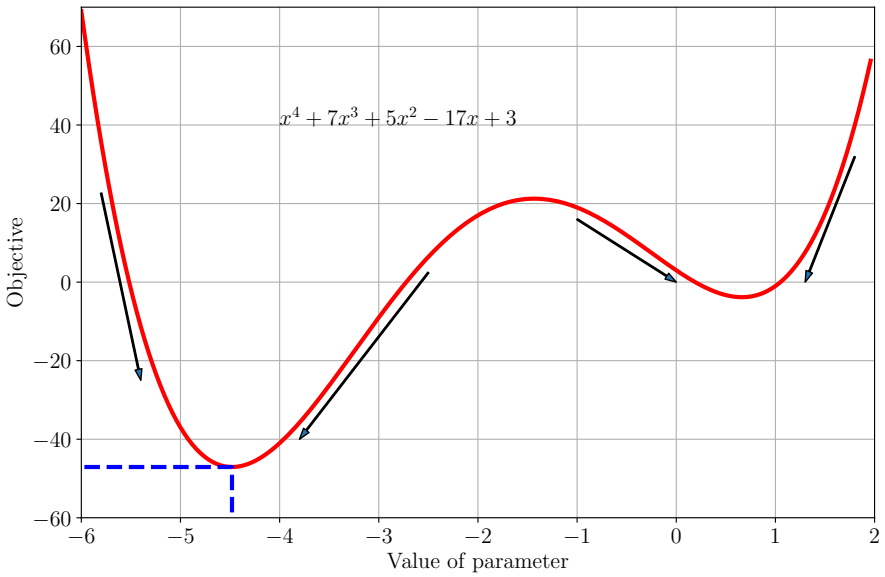


Figure 7.2 Example objective function. Negative gradients are indicated by arrows, and the global minimum is indicated by the dashed blue line.

right, but not how far (this is called the step-size). Furthermore, if we had started at the right side (e.g., $x_0 = 0$) the negative gradient would have led us to the wrong minimum. Figure 7.2 illustrates the fact that for $x > -1$, the negative gradient points toward the minimum on the right of the figure, which has a larger objective value.

According to the Abel–Ruffini theorem, there is in general no algebraic solution for polynomials of degree 5 or more (Abel, 1826).

In Section 7.3, we will learn about a class of functions, called convex functions, that do not exhibit this tricky dependency on the starting point of the optimization algorithm. For convex functions, all local minimums are global minimum. It turns out that many machine learning objective functions are designed such that they are convex, and we will see an example in Chapter 12.

For convex functions all local minima are global minimum.

The discussion in this chapter so far was about a one-dimensional function, where we are able to visualize the ideas of gradients, descent directions, and optimal values. In the rest of this chapter we develop the same ideas in high dimensions. Unfortunately, we can only visualize the concepts in one dimension, but some concepts do not generalize directly to higher dimensions, therefore some care needs to be taken when reading.

7.1 Optimization Using Gradient Descent

We now consider the problem of solving for the minimum of a real-valued function

$$\min_{\boldsymbol{x}} f(\boldsymbol{x})\,, \tag{7.4}$$

where $f : \mathbb{R}^d \to \mathbb{R}$ is an objective function that captures the machine learning problem at hand. We assume that our function f is differentiable, and we are unable to analytically find a solution in closed form.

Gradient descent is a first-order optimization algorithm. To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient of the function at the current point. Recall from Section 5.1 that the gradient points in the direction of the steepest ascent. Another useful intuition is to consider the set of lines where the function is at a certain value ($f(\boldsymbol{x}) = c$ for some value $c \in \mathbb{R}$), which are known as the contour lines. The gradient points in a direction that is orthogonal to the contour lines of the function we wish to optimize.

We use the convention of row vectors for gradients.

Let us consider multivariate functions. Imagine a surface (described by the function $f(\boldsymbol{x})$) with a ball starting at a particular location $\boldsymbol{x}_0$. When the ball is released, it will move downhill in the direction of steepest descent. Gradient descent exploits the fact that $f(\boldsymbol{x}_0)$ decreases fastest if one moves from $\boldsymbol{x}_0$ in the direction of the negative gradient $-((\nabla f)(\boldsymbol{x}_0))^\top$ of f at $\boldsymbol{x}_0$. We assume in this book that the functions are differentiable, and refer the reader to more general settings in Section 7.4. Then, if

$$\boldsymbol{x}_1 = \boldsymbol{x}_0 - \gamma((\nabla f)(\boldsymbol{x}_0))^\top \tag{7.5}$$

for a small *step-size* $\gamma \geqslant 0$, then $f(\boldsymbol{x}_1) \leqslant f(\boldsymbol{x}_0)$. Note that we use the transpose for the gradient since otherwise the dimensions will not work out.

This observation allows us to define a simple gradient descent algorithm: If we want to find a local optimum $f(\boldsymbol{x}_*)$ of a function $f : \mathbb{R}^n \to \mathbb{R},\ \boldsymbol{x} \mapsto f(\boldsymbol{x})$, we start with an initial guess $\boldsymbol{x}_0$ of the parameters we wish to optimize and then iterate according to

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i - \gamma_i((\nabla f)(\boldsymbol{x}_i))^\top . \tag{7.6}$$

For suitable step-size γ_i, the sequence $f(\boldsymbol{x}_0) \geqslant f(\boldsymbol{x}_1) \geqslant \ldots$ converges to a local minimum.

Example 7.1

Consider a quadratic function in two dimensions

$$f\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \frac{1}{2}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^\top \begin{bmatrix} 2 & 1 \\ 1 & 20 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} - \begin{bmatrix} 5 \\ 3 \end{bmatrix}^\top \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \tag{7.7}$$

with gradient

$$\nabla f\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^\top \begin{bmatrix} 2 & 1 \\ 1 & 20 \end{bmatrix} - \begin{bmatrix} 5 \\ 3 \end{bmatrix}^\top . \tag{7.8}$$

Starting at the initial location $\boldsymbol{x}_0 = [-3, -1]^\top$, we iteratively apply (7.6) to obtain a sequence of estimates that converge to the minimum value

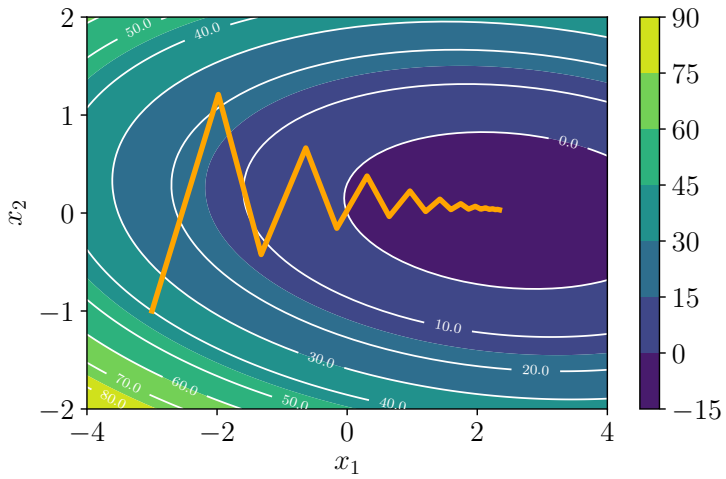


Figure 7.3 Gradient descent on a two-dimensional quadratic surface (shown as a heatmap). See Example 7.1 for a description.

(illustrated in Figure 7.3). We can see (both from the figure and by plugging $\boldsymbol{x}_0$ into (7.8) with $\gamma = 0.085$) that the negative gradient at $\boldsymbol{x}_0$ points north and east, leading to $\boldsymbol{x}_1 = [-1.98, 1.21]^\top$. Repeating that argument gives us $\boldsymbol{x}_2 = [-1.32, -0.42]^\top$, and so on.

Remark. Gradient descent can be relatively slow close to the minimum: Its asymptotic rate of convergence is inferior to many other methods. Using the ball rolling down the hill analogy, when the surface is a long, thin valley, the problem is poorly conditioned (Trefethen and Bau III, 1997). For poorly conditioned convex problems, gradient descent increasingly "zigzags" as the gradients point nearly orthogonally to the shortest direction to a minimum point; see Figure 7.3. ♢

7.1.1 Step-size

The step-size is also called the learning rate.

As mentioned earlier, choosing a good step-size is important in gradient descent. If the step-size is too small, gradient descent can be slow. If the step-size is chosen too large, gradient descent can overshoot, fail to converge, or even diverge. We will discuss the use of momentum in the next section. It is a method that smoothes out erratic behavior of gradient updates and dampens oscillations.

Adaptive gradient methods rescale the step-size at each iteration, depending on local properties of the function. There are two simple heuristics (Toussaint, 2012):

- When the function value increases after a gradient step, the step-size was too large. Undo the step and decrease the step-size.
- When the function value decreases the step could have been larger. Try to increase the step-size.

Although the "undo" step seems to be a waste of resources, using this heuristic guarantees monotonic convergence.

Example 7.2 (Solving a Linear Equation System)
When we solve linear equations of the form $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, in practice we solve $\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} = \boldsymbol{0}$ approximately by finding $\boldsymbol{x}_*$ that minimizes the squared error

$$\|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|^2 = (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})^\top (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) \tag{7.9}$$

if we use the Euclidean norm. The gradient of (7.9) with respect to $\boldsymbol{x}$ is

$$\nabla_{\boldsymbol{x}} = 2(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})^\top \boldsymbol{A}\,. \tag{7.10}$$

We can use this gradient directly in a gradient descent algorithm. However, for this particular special case, it turns out that there is an analytic solution, which can be found by setting the gradient to zero. We will see more on solving squared error problems in Chapter 9.

Remark. When applied to the solution of linear systems of equations $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, gradient descent may converge slowly. The speed of convergence of gradient descent is dependent on the *condition number* $\kappa = \frac{\sigma(\boldsymbol{A})_{\max}}{\sigma(\boldsymbol{A})_{\min}}$, which is the ratio of the maximum to the minimum singular value (Section 4.5) of $\boldsymbol{A}$. The condition number essentially measures the ratio of the most curved direction versus the least curved direction, which corresponds to our imagery that poorly conditioned problems are long, thin valleys: They are very curved in one direction, but very flat in the other. Instead of directly solving $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$, one could instead solve $\boldsymbol{P}^{-1}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) = \boldsymbol{0}$, where $\boldsymbol{P}$ is called the *preconditioner*. The goal is to design $\boldsymbol{P}^{-1}$ such that $\boldsymbol{P}^{-1}\boldsymbol{A}$ has a better condition number, but at the same time $\boldsymbol{P}^{-1}$ is easy to compute. For further information on gradient descent, preconditioning, and convergence we refer to Boyd and Vandenberghe (2004, chapter 9). ♢

condition number

preconditioner

7.1.2 Gradient Descent With Momentum

As illustrated in Figure 7.3, the convergence of gradient descent may be very slow if the curvature of the optimization surface is such that there are regions that are poorly scaled. The curvature is such that the gradient descent steps hops between the walls of the valley and approaches the optimum in small steps. The proposed tweak to improve convergence is to give gradient descent some memory.

Goh (2017) wrote an intuitive blog post on gradient descent with momentum.

Gradient descent with momentum (Rumelhart et al., 1986) is a method that introduces an additional term to remember what happened in the previous iteration. This memory dampens oscillations and smoothes out the gradient updates. Continuing the ball analogy, the momentum term emulates the phenomenon of a heavy ball that is reluctant to change directions. The idea is to have a gradient update with memory to implement

a moving average. The momentum-based method remembers the update $\Delta\boldsymbol{x}_i$ at each iteration i and determines the next update as a linear combination of the current and previous gradients

$$\boldsymbol{x}_{i+1} = \boldsymbol{x}_i - \gamma_i((\nabla f)(\boldsymbol{x}_i))^\top + \alpha\Delta\boldsymbol{x}_i \tag{7.11}$$

$$\Delta\boldsymbol{x}_i = \boldsymbol{x}_i - \boldsymbol{x}_{i-1} = \alpha\Delta\boldsymbol{x}_{i-1} - \gamma_{i-1}((\nabla f)(\boldsymbol{x}_{i-1}))^\top , \tag{7.12}$$

where $\alpha \in [0, 1]$. Sometimes we will only know the gradient approximately. In such cases, the momentum term is useful since it averages out different noisy estimates of the gradient. One particularly useful way to obtain an approximate gradient is by using a stochastic approximation, which we discuss next.

7.1.3 Stochastic Gradient Descent

Computing the gradient can be very time consuming. However, often it is possible to find a "cheap" approximation of the gradient. Approximating the gradient is still useful as long as it points in roughly the same direction as the true gradient.

stochastic gradient descent

Stochastic gradient descent (often shortened as SGD) is a stochastic approximation of the gradient descent method for minimizing an objective function that is written as a sum of differentiable functions. The word stochastic here refers to the fact that we acknowledge that we do not know the gradient precisely, but instead only know a noisy approximation to it. By constraining the probability distribution of the approximate gradients, we can still theoretically guarantee that SGD will converge.

In machine learning, given $n = 1, \ldots, N$ data points, we often consider objective functions that are the sum of the losses L_n incurred by each example n. In mathematical notation, we have the form

$$L(\boldsymbol{\theta}) = \sum_{n=1}^{N} L_n(\boldsymbol{\theta}) , \tag{7.13}$$

where $\boldsymbol{\theta}$ is the vector of parameters of interest, i.e., we want to find $\boldsymbol{\theta}$ that minimizes L. An example from regression (Chapter 9) is the negative log-likelihood, which is expressed as a sum over log-likelihoods of individual examples so that

$$L(\boldsymbol{\theta}) = -\sum_{n=1}^{N} \log p(y_n|\boldsymbol{x}_n, \boldsymbol{\theta}) , \tag{7.14}$$

where $\boldsymbol{x}_n \in \mathbb{R}^D$ are the training inputs, y_n are the training targets, and $\boldsymbol{\theta}$ are the parameters of the regression model.

Standard gradient descent, as introduced previously, is a "batch" optimization method, i.e., optimization is performed using the full training set

by updating the vector of parameters according to

$$\boldsymbol{\theta}_{i+1} = \boldsymbol{\theta}_i - \gamma_i (\nabla L(\boldsymbol{\theta}_i))^\top = \boldsymbol{\theta}_i - \gamma_i \sum_{n=1}^{N} (\nabla L_n(\boldsymbol{\theta}_i))^\top \tag{7.15}$$

for a suitable step-size parameter γ_i. Evaluating the sum gradient may require expensive evaluations of the gradients from all individual functions L_n. When the training set is enormous and/or no simple formulas exist, evaluating the sums of gradients becomes very expensive.

Consider the term $\sum_{n=1}^{N} (\nabla L_n(\boldsymbol{\theta}_i))$ in (7.15). We can reduce the amount of computation by taking a sum over a smaller set of L_n. In contrast to batch gradient descent, which uses all L_n for $n = 1, \ldots, N$, we randomly choose a subset of L_n for mini-batch gradient descent. In the extreme case, we randomly select only a single L_n to estimate the gradient. The key insight about why taking a subset of data is sensible is to realize that for gradient descent to converge, we only require that the gradient is an unbiased estimate of the true gradient. In fact the term $\sum_{n=1}^{N} (\nabla L_n(\boldsymbol{\theta}_i))$ in (7.15) is an empirical estimate of the expected value (Section 6.4.1) of the gradient. Therefore, any other unbiased empirical estimate of the expected value, for example using any subsample of the data, would suffice for convergence of gradient descent.

Remark. When the learning rate decreases at an appropriate rate, and subject to relatively mild assumptions, stochastic gradient descent converges almost surely to local minimum (Bottou, 1998). ♢

Why should one consider using an approximate gradient? A major reason is practical implementation constraints, such as the size of central processing unit (CPU)/graphics processing unit (GPU) memory or limits on computational time. We can think of the size of the subset used to estimate the gradient in the same way that we thought of the size of a sample when estimating empirical means (Section 6.4.1). Large mini-batch sizes will provide accurate estimates of the gradient, reducing the variance in the parameter update. Furthermore, large mini-batches take advantage of highly optimized matrix operations in vectorized implementations of the cost and gradient. The reduction in variance leads to more stable convergence, but each gradient calculation will be more expensive.

In contrast, small mini-batches are quick to estimate. If we keep the mini-batch size small, the noise in our gradient estimate will allow us to get out of some bad local optima, which we may otherwise get stuck in. In machine learning, optimization methods are used for training by minimizing an objective function on the training data, but the overall goal is to improve generalization performance (Chapter 8). Since the goal in machine learning does not necessarily need a precise estimate of the minimum of the objective function, approximate gradients using mini-batch approaches have been widely used. Stochastic gradient descent is very effective in large-scale machine learning problems (Bottou et al., 2018),

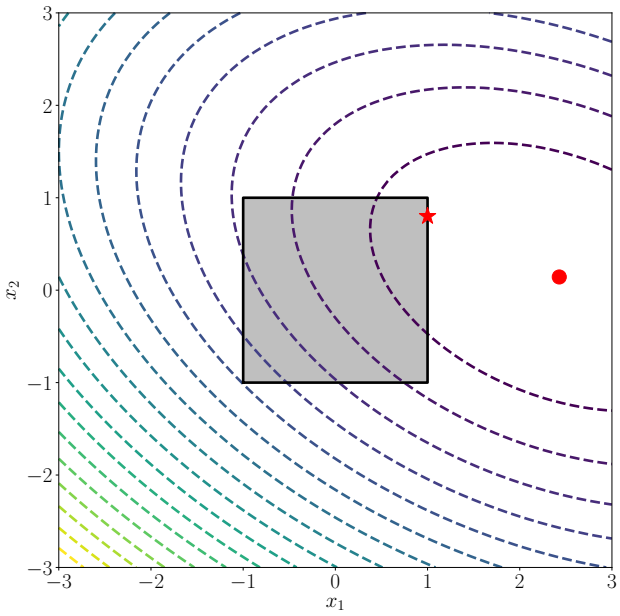


Figure 7.4 Illustration of constrained optimization. The unconstrained problem (indicated by the contour lines) has a minimum on the right side (indicated by the circle). The box constraints ($-1 \leqslant x \leqslant 1$ and $-1 \leqslant y \leqslant 1$) require that the optimal solution is within the box, resulting in an optimal value indicated by the star.

such as training deep neural networks on millions of images (Dean et al., 2012), topic models (Hoffman et al., 2013), reinforcement learning (Mnih et al., 2015), or training of large-scale Gaussian process models (Hensman et al., 2013; Gal et al., 2014).

7.2 Constrained Optimization and Lagrange Multipliers

In the previous section, we considered the problem of solving for the minimum of a function

$$\min_{\boldsymbol{x}} f(\boldsymbol{x})\,, \tag{7.16}$$

where $f : \mathbb{R}^D \to \mathbb{R}$.

In this section, we have additional constraints. That is, for real-valued functions $g_i : \mathbb{R}^D \to \mathbb{R}$ for $i = 1, \ldots, m$, we consider the constrained optimization problem (see Figure 7.4 for an illustration)

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}) \\ \text{subject to} \quad & g_i(\boldsymbol{x}) \leqslant 0 \quad \text{for all} \quad i = 1, \ldots, m\,. \end{aligned} \tag{7.17}$$

It is worth pointing out that the functions f and g_i could be non-convex in general, and we will consider the convex case in the next section.

One obvious, but not very practical, way of converting the constrained problem (7.17) into an unconstrained one is to use an indicator function

$$J(\boldsymbol{x}) = f(\boldsymbol{x}) + \sum_{i=1}^{m} \mathbf{1}(g_i(\boldsymbol{x}))\,, \tag{7.18}$$

where $\mathbf{1}(z)$ is an infinite step function

$$\mathbf{1}(z) = \begin{cases} 0 & \text{if } z \leqslant 0 \\ \infty & \text{otherwise} \end{cases} . \tag{7.19}$$

This gives infinite penalty if the constraint is not satisfied, and hence would provide the same solution. However, this infinite step function is equally difficult to optimize. We can overcome this difficulty by introducing *Lagrange multipliers*. The idea of Lagrange multipliers is to replace the step function with a linear function.

Lagrange multiplier

Lagrangian

We associate to problem (7.17) the *Lagrangian* by introducing the Lagrange multipliers $\lambda_i \geqslant 0$ corresponding to each inequality constraint respectively (Boyd and Vandenberghe, 2004, chapter 4) so that

$$\mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = f(\boldsymbol{x}) + \sum_{i=1}^{m} \lambda_i g_i(\boldsymbol{x}) \tag{7.20a}$$

$$= f(\boldsymbol{x}) + \boldsymbol{\lambda}^\top \boldsymbol{g}(\boldsymbol{x}) , \tag{7.20b}$$

where in the last line we have concatenated all constraints $g_i(\boldsymbol{x})$ into a vector $\boldsymbol{g}(\boldsymbol{x})$, and all the Lagrange multipliers into a vector $\boldsymbol{\lambda} \in \mathbb{R}^m$.

We now introduce the idea of Lagrangian duality. In general, duality in optimization is the idea of converting an optimization problem in one set of variables $\boldsymbol{x}$ (called the primal variables), into another optimization problem in a different set of variables $\boldsymbol{\lambda}$ (called the dual variables). We introduce two different approaches to duality: In this section, we discuss Lagrangian duality; in Section 7.3.3, we discuss Legendre-Fenchel duality.

Definition 7.1. The problem in (7.17)

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}) \\ \text{subject to} \quad & g_i(\boldsymbol{x}) \leqslant 0 \quad \text{for all} \quad i = 1, \dots, m \end{aligned} \tag{7.21}$$

primal problem

is known as the *primal problem*, corresponding to the primal variables x.

Lagrangian dual problem

The associated *Lagrangian dual problem* is given by

$$\begin{aligned} \max_{\boldsymbol{\lambda} \in \mathbb{R}^m} \quad & \mathfrak{D}(\boldsymbol{\lambda}) \\ \text{subject to} \quad & \boldsymbol{\lambda} \geqslant \mathbf{0} , \end{aligned} \tag{7.22}$$

where $\boldsymbol{\lambda}$ are the dual variables and $\mathfrak{D}(\boldsymbol{\lambda}) = \min_{\boldsymbol{x} \in \mathbb{R}^d} \mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda})$.

Remark. In the discussion of Definition 7.1, we use two concepts that are also of independent interest (Boyd and Vandenberghe, 2004).

minimax inequality

First is the *minimax inequality*, which says that for any function with two arguments $\varphi(\boldsymbol{x}, \boldsymbol{y})$, the maximin is less than the minimax, i.e.,

$$\max_{\boldsymbol{y}} \min_{\boldsymbol{x}} \varphi(\boldsymbol{x}, \boldsymbol{y}) \leqslant \min_{\boldsymbol{x}} \max_{\boldsymbol{y}} \varphi(\boldsymbol{x}, \boldsymbol{y}) . \tag{7.23}$$

This inequality can be proved by considering the inequality

$$\text{For all } \boldsymbol{x}, \boldsymbol{y} \qquad \min_{\boldsymbol{x}} \varphi(\boldsymbol{x}, \boldsymbol{y}) \leqslant \max_{\boldsymbol{y}} \varphi(\boldsymbol{x}, \boldsymbol{y}) \,. \tag{7.24}$$

Note that taking the maximum over $\boldsymbol{y}$ of the left-hand side of (7.24) maintains the inequality since the inequality is true for all $\boldsymbol{y}$. Similarly, we can take the minimum over $\boldsymbol{x}$ of the right-hand side of (7.24) to obtain (7.23).

The second concept is *weak duality*, which uses (7.23) to show that primal values are always greater than or equal to dual values. This is described in more detail in (7.27). ♢

weak duality

Recall that the difference between $J(\boldsymbol{x})$ in (7.18) and the Lagrangian in (7.20b) is that we have relaxed the indicator function to a linear function. Therefore, when $\lambda \geqslant 0$, the Lagrangian $\mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda})$ is a lower bound of $J(\boldsymbol{x})$. Hence, the maximum of $\mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda})$ with respect to $\boldsymbol{\lambda}$ is

$$J(\boldsymbol{x}) = \max_{\boldsymbol{\lambda} \geqslant \boldsymbol{0}} \mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda}) \,. \tag{7.25}$$

Recall that the original problem was minimizing $J(\boldsymbol{x})$,

$$\min_{\boldsymbol{x} \in \mathbb{R}^d} \max_{\boldsymbol{\lambda} \geqslant \boldsymbol{0}} \mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda}) \,. \tag{7.26}$$

By the minimax inequality (7.23), it follows that swapping the order of the minimum and maximum results in a smaller value, i.e.,

$$\min_{\boldsymbol{x} \in \mathbb{R}^d} \max_{\boldsymbol{\lambda} \geqslant \boldsymbol{0}} \mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda}) \geqslant \max_{\boldsymbol{\lambda} \geqslant \boldsymbol{0}} \min_{\boldsymbol{x} \in \mathbb{R}^d} \mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda}) \,. \tag{7.27}$$

This is also known as *weak duality*. Note that the inner part of the right-hand side is the dual objective function $\mathfrak{D}(\boldsymbol{\lambda})$ and the definition follows.

weak duality

In contrast to the original optimization problem, which has constraints, $\min_{\boldsymbol{x} \in \mathbb{R}^d} \mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda})$ is an unconstrained optimization problem for a given value of $\boldsymbol{\lambda}$. If solving $\min_{\boldsymbol{x} \in \mathbb{R}^d} \mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda})$ is easy, then the overall problem is easy to solve. We can see this by observing from (7.20b) that $\mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda})$ is affine with respect to $\boldsymbol{\lambda}$. Therefore $\min_{\boldsymbol{x} \in \mathbb{R}^d} \mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda})$ is a pointwise minimum of affine functions of $\boldsymbol{\lambda}$, and hence $\mathfrak{D}(\boldsymbol{\lambda})$ is concave even though $f(\cdot)$ and $g_i(\cdot)$ may be nonconvex. The outer problem, maximization over $\boldsymbol{\lambda}$, is the maximum of a concave function and can be efficiently computed.

Assuming $f(\cdot)$ and $g_i(\cdot)$ are differentiable, we find the Lagrange dual problem by differentiating the Lagrangian with respect to $\boldsymbol{x}$, setting the differential to zero, and solving for the optimal value. We will discuss two concrete examples in Sections 7.3.1 and 7.3.2, where $f(\cdot)$ and $g_i(\cdot)$ are convex.

Remark (Equality Constraints). Consider (7.17) with additional equality constraints

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f(\boldsymbol{x}) \\ \text{subject to} \quad & g_i(\boldsymbol{x}) \leqslant 0 \quad \text{for all} \quad i = 1, \ldots, m \\ & h_j(\boldsymbol{x}) = 0 \quad \text{for all} \quad j = 1, \ldots, n \,. \end{aligned} \tag{7.28}$$

We can model equality constraints by replacing them with two inequality constraints. That is for each equality constraint $h_j(\boldsymbol{x}) = 0$ we equivalently replace it by two constraints $h_j(\boldsymbol{x}) \leqslant 0$ and $h_j(\boldsymbol{x}) \geqslant 0$. It turns out that the resulting Lagrange multipliers are then unconstrained.

Therefore, we constrain the Lagrange multipliers corresponding to the inequality constraints in (7.28) to be non-negative, and leave the Lagrange multipliers corresponding to the equality constraints unconstrained. ♢

7.3 Convex Optimization

We focus our attention of a particularly useful class of optimization problems, where we can guarantee global optimality. When $f(\cdot)$ is a convex function, and when the constraints involving $g(\cdot)$ and $h(\cdot)$ are convex sets, this is called a *convex optimization problem*. In this setting, we have *strong duality*: The optimal solution of the dual problem is the same as the optimal solution of the primal problem. The distinction between convex functions and convex sets are often not strictly presented in machine learning literature, but one can often infer the implied meaning from context.

convex optimization problem

strong duality

convex set

Definition 7.2. A set $\mathcal{C}$ is a *convex set* if for any $x, y \in \mathcal{C}$ and for any scalar θ with $0 \leqslant \theta \leqslant 1$, we have

$$\theta x + (1-\theta)y \in \mathcal{C}\,. \tag{7.29}$$

Figure 7.5 Example of a convex set.

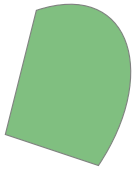
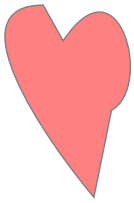


Figure 7.6 Example of a nonconvex set.



Convex sets are sets such that a straight line connecting any two elements of the set lie inside the set. Figures 7.5 and 7.6 illustrate convex and nonconvex sets, respectively.

Convex functions are functions such that a straight line between any two points of the function lie above the function. Figure 7.2 shows a nonconvex function, and Figure 7.3 shows a convex function. Another convex function is shown in Figure 7.7.

Definition 7.3. Let function $f : \mathbb{R}^D \to \mathbb{R}$ be a function whose domain is a convex set. The function f is a *convex function* if for all $\boldsymbol{x}, \boldsymbol{y}$ in the domain of f, and for any scalar θ with $0 \leqslant \theta \leqslant 1$, we have

$$f(\theta\boldsymbol{x} + (1-\theta)\boldsymbol{y}) \leqslant \theta f(\boldsymbol{x}) + (1-\theta)f(\boldsymbol{y})\,. \tag{7.30}$$

Remark. A *concave function* is the negative of a convex function. ♢

convex function

concave function

The constraints involving $g(\cdot)$ and $h(\cdot)$ in (7.28) truncate functions at a scalar value, resulting in sets. Another relation between convex functions and convex sets is to consider the set obtained by "filling in" a convex function. A convex function is a bowl-like object, and we imagine pouring water into it to fill it up. This resulting filled-in set, called the *epigraph* of the convex function, is a convex set.

epigraph

If a function $f : \mathbb{R}^n \to \mathbb{R}$ is differentiable, we can specify convexity in

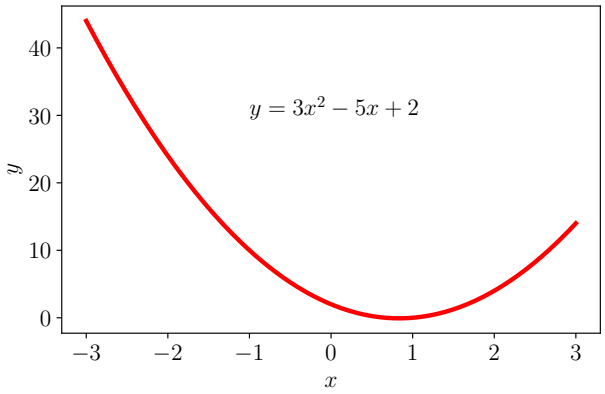


Figure 7.7 Example of a convex function.

terms of its gradient $\nabla_{\boldsymbol{x}} f(\boldsymbol{x})$ (Section 5.2). A function $f(\boldsymbol{x})$ is convex if and only if for any two points $\boldsymbol{x}, \boldsymbol{y}$ it holds that

$$f(\boldsymbol{y}) \geqslant f(\boldsymbol{x}) + \nabla_{\boldsymbol{x}} f(\boldsymbol{x})^\top (\boldsymbol{y} - \boldsymbol{x}) \,. \tag{7.31}$$

If we further know that a function $f(\boldsymbol{x})$ is twice differentiable, that is, the Hessian (5.147) exists for all values in the domain of $\boldsymbol{x}$, then the function $f(\boldsymbol{x})$ is convex if and only if $\nabla^2_{\boldsymbol{x}} f(\boldsymbol{x})$ is positive semidefinite (Boyd and Vandenberghe, 2004).

Example 7.3

The negative entropy $f(x) = x \log_2 x$ is convex for $x > 0$. A visualization of the function is shown in Figure 7.8, and we can see that the function is convex. To illustrate the previous definitions of convexity, let us check the calculations for two points $x = 2$ and $x = 4$. Note that to prove convexity of $f(x)$ we would need to check for all points $x \in \mathbb{R}$.

Recall Definition 7.3. Consider a point midway between the two points (that is $\theta = 0.5$); then the left-hand side is $f(0.5 \cdot 2 + 0.5 \cdot 4) = 3 \log_2 3 \approx 4.75$. The right-hand side is $0.5(2 \log_2 2) + 0.5(4 \log_2 4) = 1 + 4 = 5$. And therefore the definition is satisfied.

Since $f(x)$ is differentiable, we can alternatively use (7.31). Calculating the derivative of $f(x)$, we obtain

$$\nabla_x (x \log_2 x) = 1 \cdot \log_2 x + x \cdot \frac{1}{x \log_e 2} = \log_2 x + \frac{1}{\log_e 2} \,. \tag{7.32}$$

Using the same two test points $x = 2$ and $x = 4$, the left-hand side of (7.31) is given by $f(4) = 8$. The right-hand side is

$$f(\boldsymbol{x}) + \nabla_{\boldsymbol{x}}^\top (\boldsymbol{y} - \boldsymbol{x}) = f(2) + \nabla f(2) \cdot (4 - 2) \tag{7.33a}$$

$$= 2 + (1 + \frac{1}{\log_e 2}) \cdot 2 \approx 6.9 \,. \tag{7.33b}$$

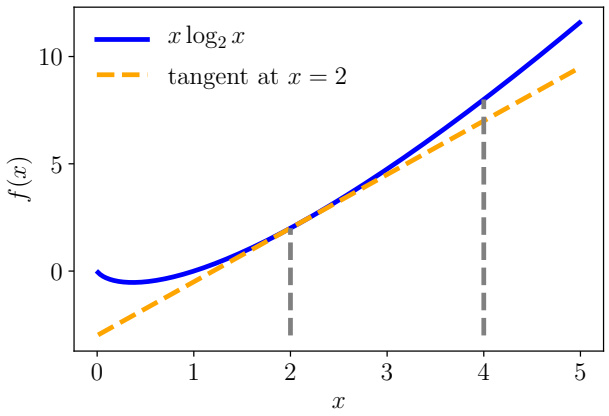


Figure 7.8 The negative entropy function (which is convex) and its tangent at $x = 2$.

We can check that a function or set is convex from first principles by recalling the definitions. In practice, we often rely on operations that preserve convexity to check that a particular function or set is convex. Although the details are vastly different, this is again the idea of closure that we introduced in Chapter 2 for vector spaces.

Example 7.4

A nonnegative weighted sum of convex functions is convex. Observe that if f is a convex function, and $\alpha \geqslant 0$ is a nonnegative scalar, then the function αf is convex. We can see this by multiplying α to both sides of the equation in Definition 7.3, and recalling that multiplying a nonnegative number does not change the inequality.

If f_1 and f_2 are convex functions, then we have by the definition

$$f_1(\theta \boldsymbol{x} + (1-\theta)\boldsymbol{y}) \leqslant \theta f_1(\boldsymbol{x}) + (1-\theta) f_1(\boldsymbol{y}) \tag{7.34}$$

$$f_2(\theta \boldsymbol{x} + (1-\theta)\boldsymbol{y}) \leqslant \theta f_2(\boldsymbol{x}) + (1-\theta) f_2(\boldsymbol{y})\,. \tag{7.35}$$

Summing up both sides gives us

$$\begin{aligned} &f_1(\theta \boldsymbol{x} + (1-\theta)\boldsymbol{y}) + f_2(\theta \boldsymbol{x} + (1-\theta)\boldsymbol{y}) \\ &\leqslant \theta f_1(\boldsymbol{x}) + (1-\theta) f_1(\boldsymbol{y}) + \theta f_2(\boldsymbol{x}) + (1-\theta) f_2(\boldsymbol{y})\,, \end{aligned} \tag{7.36}$$

where the right-hand side can be rearranged to

$$\theta(f_1(\boldsymbol{x}) + f_2(\boldsymbol{x})) + (1-\theta)(f_1(\boldsymbol{y}) + f_2(\boldsymbol{y}))\,, \tag{7.37}$$

completing the proof that the sum of convex functions is convex.

Combining the preceding two facts, we see that $\alpha f_1(\boldsymbol{x}) + \beta f_2(\boldsymbol{x})$ is convex for $\alpha, \beta \geqslant 0$. This closure property can be extended using a similar argument for nonnegative weighted sums of more than two convex functions.

Remark. The inequality in (7.30) is sometimes called *Jensen's inequality*. In fact, a whole class of inequalities for taking nonnegative weighted sums of convex functions are all called Jensen's inequality. ♢

Jensen's inequality

In summary, a constrained optimization problem is called a *convex optimization problem* if

convex optimization problem

$$\begin{aligned} &\min_{\boldsymbol{x}} f(\boldsymbol{x}) \\ \text{subject to } &g_i(\boldsymbol{x}) \leqslant 0 \quad \text{for all} \quad i = 1, \dots, m \\ &h_j(\boldsymbol{x}) = 0 \quad \text{for all} \quad j = 1, \dots, n\,, \end{aligned} \tag{7.38}$$

where all functions $f(\boldsymbol{x})$ and $g_i(\boldsymbol{x})$ are convex functions, and all $h_j(\boldsymbol{x}) = 0$ are convex sets. In the following, we will describe two classes of convex optimization problems that are widely used and well understood.

7.3.1 Linear Programming

Consider the special case when all the preceding functions are linear, i.e.,

$$\begin{aligned} \min_{\boldsymbol{x} \in \mathbb{R}^d} \quad & \boldsymbol{c}^\top \boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} \leqslant \boldsymbol{b}\,, \end{aligned} \tag{7.39}$$

where $\boldsymbol{A} \in \mathbb{R}^{m \times d}$ and $\boldsymbol{b} \in \mathbb{R}^m$. This is known as a *linear program*. It has d variables and m linear constraints. The Lagrangian is given by

linear program

Linear programs are one of the most widely used approaches in industry.

$$\mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = \boldsymbol{c}^\top \boldsymbol{x} + \boldsymbol{\lambda}^\top (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b})\,, \tag{7.40}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^m$ is the vector of non-negative Lagrange multipliers. Rearranging the terms corresponding to $\boldsymbol{x}$ yields

$$\mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda}) = (\boldsymbol{c} + \boldsymbol{A}^\top \boldsymbol{\lambda})^\top \boldsymbol{x} - \boldsymbol{\lambda}^\top \boldsymbol{b}\,. \tag{7.41}$$

Taking the derivative of $\mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda})$ with respect to $\boldsymbol{x}$ and setting it to zero gives us

$$\boldsymbol{c} + \boldsymbol{A}^\top \boldsymbol{\lambda} = \boldsymbol{0}\,. \tag{7.42}$$

Therefore, the dual Lagrangian is $\mathfrak{D}(\boldsymbol{\lambda}) = -\boldsymbol{\lambda}^\top \boldsymbol{b}$. Recall we would like to maximize $\mathfrak{D}(\boldsymbol{\lambda})$. In addition to the constraint due to the derivative of $\mathfrak{L}(\boldsymbol{x}, \boldsymbol{\lambda})$ being zero, we also have the fact that $\boldsymbol{\lambda} \geqslant \boldsymbol{0}$, resulting in the following dual optimization problem

It is convention to minimize the primal and maximize the dual.

$$\begin{aligned} \max_{\boldsymbol{\lambda} \in \mathbb{R}^m} \quad & -\boldsymbol{b}^\top \boldsymbol{\lambda} \\ \text{subject to} \quad & \boldsymbol{c} + \boldsymbol{A}^\top \boldsymbol{\lambda} = \boldsymbol{0} \\ & \boldsymbol{\lambda} \geqslant \boldsymbol{0}\,. \end{aligned} \tag{7.43}$$

This is also a linear program, but with m variables. We have the choice of solving the primal (7.39) or the dual (7.43) program depending on

whether m or d is larger. Recall that d is the number of variables and m is the number of constraints in the primal linear program.

Example 7.5 (Linear Program)
Consider the linear program

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathbb{R}^2} \quad & -\begin{bmatrix}5\\3\end{bmatrix}^\top \begin{bmatrix}x_1\\x_2\end{bmatrix} \\ \text{subject to} \quad & \begin{bmatrix}2 & 2\\2 & -4\\-2 & 1\\0 & -1\\0 & 1\end{bmatrix} \begin{bmatrix}x_1\\x_2\end{bmatrix} \leqslant \begin{bmatrix}33\\8\\5\\-1\\8\end{bmatrix} \end{aligned} \tag{7.44}$$

with two variables. This program is also shown in Figure 7.9. The objective function is linear, resulting in linear contour lines. The constraint set in standard form is translated into the legend. The optimal value must lie in the shaded (feasible) region, and is indicated by the star.

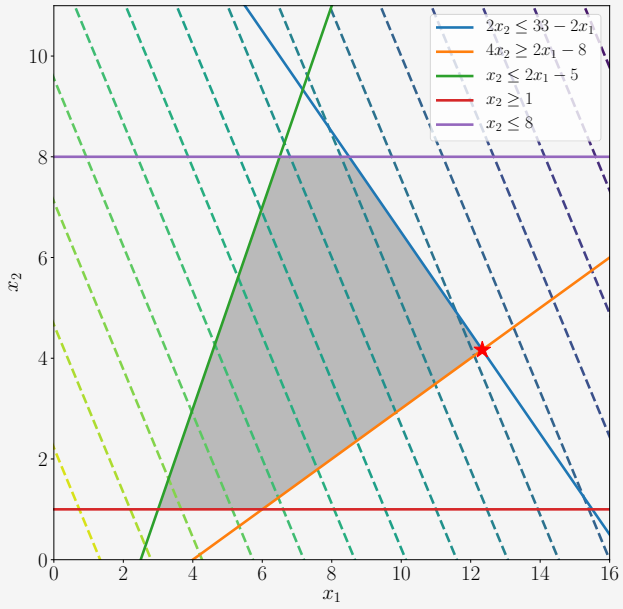


Figure 7.9 Illustration of a linear program. The unconstrained problem (indicated by the contour lines) has a minimum on the right side. The optimal value given the constraints are shown by the star.

7.3.2 Quadratic Programming

Consider the case of a convex quadratic objective function, where the constraints are affine, i.e.,

$$\begin{aligned} \min_{\boldsymbol{x}\in\mathbb{R}^d} \quad & \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{c}^\top \boldsymbol{x} \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} \leqslant \boldsymbol{b}\,, \end{aligned} \tag{7.45}$$

where $\boldsymbol{A} \in \mathbb{R}^{m\times d}$, $\boldsymbol{b} \in \mathbb{R}^m$, and $\boldsymbol{c} \in \mathbb{R}^d$. The square symmetric matrix $\boldsymbol{Q} \in \mathbb{R}^{d\times d}$ is positive definite, and therefore the objective function is convex. This is known as a *quadratic program*. Observe that it has d variables and m linear constraints.

Example 7.6 (Quadratic Program)
Consider the quadratic program

$$\min_{\boldsymbol{x}\in\mathbb{R}^2} \quad \frac{1}{2}\begin{bmatrix}x_1\\x_2\end{bmatrix}^\top \begin{bmatrix}2 & 1\\1 & 4\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix} + \begin{bmatrix}5\\3\end{bmatrix}^\top \begin{bmatrix}x_1\\x_2\end{bmatrix} \tag{7.46}$$

$$\text{subject to} \quad \begin{bmatrix}1 & 0\\-1 & 0\\0 & 1\\0 & -1\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix} \leqslant \begin{bmatrix}1\\1\\1\\1\end{bmatrix} \tag{7.47}$$

of two variables. The program is also illustrated in Figure 7.4. The objective function is quadratic with a positive semidefinite matrix $\boldsymbol{Q}$, resulting in elliptical contour lines. The optimal value must lie in the shaded (feasible) region, and is indicated by the star.

The Lagrangian is given by

$$\begin{aligned} \mathfrak{L}(\boldsymbol{x},\boldsymbol{\lambda}) &= \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{Q}\boldsymbol{x} + \boldsymbol{c}^\top \boldsymbol{x} + \boldsymbol{\lambda}^\top(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) && \text{(7.48a)}\\ &= \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{Q}\boldsymbol{x} + (\boldsymbol{c} + \boldsymbol{A}^\top\boldsymbol{\lambda})^\top \boldsymbol{x} - \boldsymbol{\lambda}^\top \boldsymbol{b}\,, && \text{(7.48b)} \end{aligned}$$

where again we have rearranged the terms. Taking the derivative of $\mathfrak{L}(\boldsymbol{x},\boldsymbol{\lambda})$ with respect to $\boldsymbol{x}$ and setting it to zero gives

$$\boldsymbol{Q}\boldsymbol{x} + (\boldsymbol{c} + \boldsymbol{A}^\top\boldsymbol{\lambda}) = \boldsymbol{0}\,. \tag{7.49}$$

Assuming that $\boldsymbol{Q}$ is invertible, we get

$$\boldsymbol{x} = -\boldsymbol{Q}^{-1}(\boldsymbol{c} + \boldsymbol{A}^\top\boldsymbol{\lambda})\,. \tag{7.50}$$

Substituting (7.50) into the primal Lagrangian $\mathfrak{L}(\boldsymbol{x},\boldsymbol{\lambda})$, we get the dual Lagrangian

$$\mathfrak{D}(\boldsymbol{\lambda}) = -\frac{1}{2}(\boldsymbol{c} + \boldsymbol{A}^\top\boldsymbol{\lambda})^\top \boldsymbol{Q}^{-1}(\boldsymbol{c} + \boldsymbol{A}^\top\boldsymbol{\lambda}) - \boldsymbol{\lambda}^\top \boldsymbol{b}\,. \tag{7.51}$$

Therefore, the dual optimization problem is given by

$$\begin{aligned} \max_{\boldsymbol{\lambda} \in \mathbb{R}^m} \quad & -\frac{1}{2}(\boldsymbol{c} + \boldsymbol{A}^\top \boldsymbol{\lambda})^\top \boldsymbol{Q}^{-1}(\boldsymbol{c} + \boldsymbol{A}^\top \boldsymbol{\lambda}) - \boldsymbol{\lambda}^\top \boldsymbol{b} \\ \text{subject to} \quad & \boldsymbol{\lambda} \geqslant \boldsymbol{0}\,. \end{aligned} \tag{7.52}$$

We will see an application of quadratic programming in machine learning in Chapter 12.

7.3.3 Legendre–Fenchel Transform and Convex Conjugate

Let us revisit the idea of duality from Section 7.2, without considering constraints. One useful fact about a convex set is that it can be equivalently described by its supporting hyperplanes. A hyperplane is called a *supporting hyperplane* of a convex set if it intersects the convex set, and the convex set is contained on just one side of it. Recall that we can fill up a convex function to obtain the epigraph, which is a convex set. Therefore, we can also describe convex functions in terms of their supporting hyperplanes. Furthermore, observe that the supporting hyperplane just touches the convex function, and is in fact the tangent to the function at that point. And recall that the tangent of a function $f(\boldsymbol{x})$ at a given point $\boldsymbol{x}_0$ is the evaluation of the gradient of that function at that point $\left.\frac{\mathrm{d}f(\boldsymbol{x})}{\mathrm{d}\boldsymbol{x}}\right|_{\boldsymbol{x}=\boldsymbol{x}_0}$. In summary, because convex sets can be equivalently described by their supporting hyperplanes, convex functions can be equivalently described by a function of their gradient. The *Legendre transform* formalizes this concept.

supporting hyperplane

Legendre transform

Physics students are often introduced to the Legendre transform as relating the Lagrangian and the Hamiltonian in classical mechanics.

Legendre-Fenchel transform

convex conjugate

We begin with the most general definition, which unfortunately has a counter-intuitive form, and look at special cases to relate the definition to the intuition described in the preceding paragraph. The *Legendre-Fenchel transform* is a transformation (in the sense of a Fourier transform) from a convex differentiable function $f(\boldsymbol{x})$ to a function that depends on the tangents $s(\boldsymbol{x}) = \nabla_{\boldsymbol{x}} f(\boldsymbol{x})$. It is worth stressing that this is a transformation of the function $f(\cdot)$ and not the variable $\boldsymbol{x}$ or the function evaluated at $\boldsymbol{x}$. The Legendre-Fenchel transform is also known as the *convex conjugate* (for reasons we will see soon) and is closely related to duality (Hiriart-Urruty and Lemaréchal, 2001, chapter 5).

convex conjugate

Definition 7.4. The *convex conjugate* of a function $f : \mathbb{R}^D \to \mathbb{R}$ is a function f^* defined by

$$f^*(\boldsymbol{s}) = \sup_{\boldsymbol{x} \in \mathbb{R}^D} \left(\langle \boldsymbol{s}, \boldsymbol{x} \rangle - f(\boldsymbol{x}) \right) . \tag{7.53}$$

Note that the preceding convex conjugate definition does not need the function f to be convex nor differentiable. In Definition 7.4, we have used a general inner product (Section 3.2) but in the rest of this section we

will consider the standard dot product between finite-dimensional vectors ($\langle \boldsymbol{s}, \boldsymbol{x} \rangle = \boldsymbol{s}^\top \boldsymbol{x}$) to avoid too many technical details.

This derivation is easiest to understand by drawing the reasoning as it progresses.

To understand Definition 7.4 in a geometric fashion, consider a nice simple one-dimensional convex and differentiable function, for example $f(x) = x^2$. Note that since we are looking at a one-dimensional problem, hyperplanes reduce to a line. Consider a line $y = sx + c$. Recall that we are able to describe convex functions by their supporting hyperplanes, so let us try to describe this function $f(x)$ by its supporting lines. Fix the gradient of the line $s \in \mathbb{R}$ and for each point $(x_0, f(x_0))$ on the graph of f, find the minimum value of c such that the line still intersects $(x_0, f(x_0))$. Note that the minimum value of c is the place where a line with slope s "just touches" the function $f(x) = x^2$. The line passing through $(x_0, f(x_0))$ with gradient s is given by

$$y - f(x_0) = s(x - x_0)\,. \tag{7.54}$$

The y-intercept of this line is $-sx_0 + f(x_0)$. The minimum of c for which $y = sx + c$ intersects with the graph of f is therefore

$$\inf_{x_0} -sx_0 + f(x_0)\,. \tag{7.55}$$

The preceding convex conjugate is by convention defined to be the negative of this. The reasoning in this paragraph did not rely on the fact that we chose a one-dimensional convex and differentiable function, and holds for $f : \mathbb{R}^D \to \mathbb{R}$, which are nonconvex and non-differentiable.

The classical Legendre transform is defined on convex differentiable functions in $\mathbb{R}^D$.

Remark. Convex differentiable functions such as the example $f(x) = x^2$ is a nice special case, where there is no need for the supremum, and there is a one-to-one correspondence between a function and its Legendre transform. Let us derive this from first principles. For a convex differentiable function, we know that at x_0 the tangent touches $f(x_0)$ so that

$$f(x_0) = sx_0 + c\,. \tag{7.56}$$

Recall that we want to describe the convex function $f(x)$ in terms of its gradient $\nabla_x f(x)$, and that $s = \nabla_x f(x_0)$. We rearrange to get an expression for $-c$ to obtain

$$-c = sx_0 - f(x_0)\,. \tag{7.57}$$

Note that $-c$ changes with x_0 and therefore with s, which is why we can think of it as a function of s, which we call

$$f^*(s) := sx_0 - f(x_0)\,. \tag{7.58}$$

Comparing (7.58) with Definition 7.4, we see that (7.58) is a special case (without the supremum). ♢

The conjugate function has nice properties; for example, for convex functions, applying the Legendre transform again gets us back to the original function. In the same way that the slope of $f(x)$ is s, the slope of $f^*(s)$

is x. The following two examples show common uses of convex conjugates in machine learning.

Example 7.7 (Convex Conjugates)
To illustrate the application of convex conjugates, consider the quadratic function

$$f(\boldsymbol{y}) = \frac{\lambda}{2}\boldsymbol{y}^\top \boldsymbol{K}^{-1}\boldsymbol{y} \tag{7.59}$$

based on a positive definite matrix $\boldsymbol{K} \in \mathbb{R}^{n\times n}$. We denote the primal variable to be $\boldsymbol{y} \in \mathbb{R}^n$ and the dual variable to be $\boldsymbol{\alpha} \in \mathbb{R}^n$.

Applying Definition 7.4, we obtain the function

$$f^*(\boldsymbol{\alpha}) = \sup_{\boldsymbol{y}\in\mathbb{R}^n} \langle \boldsymbol{y}, \boldsymbol{\alpha}\rangle - \frac{\lambda}{2}\boldsymbol{y}^\top \boldsymbol{K}^{-1}\boldsymbol{y}\,. \tag{7.60}$$

Since the function is differentiable, we can find the maximum by taking the derivative and with respect to $\boldsymbol{y}$ setting it to zero.

$$\frac{\partial\left[\langle \boldsymbol{y}, \boldsymbol{\alpha}\rangle - \frac{\lambda}{2}\boldsymbol{y}^\top \boldsymbol{K}^{-1}\boldsymbol{y}\right]}{\partial \boldsymbol{y}} = (\boldsymbol{\alpha} - \lambda \boldsymbol{K}^{-1}\boldsymbol{y})^\top \tag{7.61}$$

and hence when the gradient is zero we have $\boldsymbol{y} = \frac{1}{\lambda}\boldsymbol{K}\boldsymbol{\alpha}$. Substituting into (7.60) yields

$$f^*(\boldsymbol{\alpha}) = \frac{1}{\lambda}\boldsymbol{\alpha}^\top \boldsymbol{K}\boldsymbol{\alpha} - \frac{\lambda}{2}\left(\frac{1}{\lambda}\boldsymbol{K}\boldsymbol{\alpha}\right)^\top \boldsymbol{K}^{-1}\left(\frac{1}{\lambda}\boldsymbol{K}\boldsymbol{\alpha}\right) = \frac{1}{2\lambda}\boldsymbol{\alpha}^\top \boldsymbol{K}\boldsymbol{\alpha}\,. \tag{7.62}$$

Example 7.8
In machine learning, we often use sums of functions; for example, the objective function of the training set includes a sum of the losses for each example in the training set. In the following, we derive the convex conjugate of a sum of losses $\ell(t)$, where $\ell : \mathbb{R} \to \mathbb{R}$. This also illustrates the application of the convex conjugate to the vector case. Let $\mathcal{L}(\boldsymbol{t}) = \sum_{i=1}^n \ell_i(t_i)$. Then,

$$\mathcal{L}^*(\boldsymbol{z}) = \sup_{\boldsymbol{t}\in\mathbb{R}^n} \langle \boldsymbol{z}, \boldsymbol{t}\rangle - \sum_{i=1}^n \ell_i(t_i) \tag{7.63a}$$

$$= \sup_{\boldsymbol{t}\in\mathbb{R}^n} \sum_{i=1}^n z_i t_i - \ell_i(t_i) \qquad \text{definition of dot product} \tag{7.63b}$$

$$= \sum_{i=1}^n \sup_{t\in\mathbb{R}^n} z_i t_i - \ell_i(t_i) \tag{7.63c}$$

$$= \sum_{i=1}^{n} \ell_i^*(z_i)\,. \qquad \text{definition of conjugate} \tag{7.63d}$$

Recall that in Section 7.2 we derived a dual optimization problem using Lagrange multipliers. Furthermore, for convex optimization problems we have strong duality, that is the solutions of the primal and dual problem match. The Legendre-Fenchel transform described here also can be used to derive a dual optimization problem. Furthermore, when the function is convex and differentiable, the supremum is unique. To further investigate the relation between these two approaches, let us consider a linear equality constrained convex optimization problem.

Example 7.9
Let $f(\boldsymbol{y})$ and $g(\boldsymbol{x})$ be convex functions, and $\boldsymbol{A}$ a real matrix of appropriate dimensions such that $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}$. Then

$$\min_{\boldsymbol{x}} f(\boldsymbol{A}\boldsymbol{x}) + g(\boldsymbol{x}) = \min_{\boldsymbol{A}\boldsymbol{x}=\boldsymbol{y}} f(\boldsymbol{y}) + g(\boldsymbol{x}). \tag{7.64}$$

By introducing the Lagrange multiplier $\boldsymbol{u}$ for the constraints $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{y}$,

$$\min_{\boldsymbol{A}\boldsymbol{x}=\boldsymbol{y}} f(\boldsymbol{y}) + g(\boldsymbol{x}) = \min_{\boldsymbol{x},\boldsymbol{y}} \max_{\boldsymbol{u}} f(\boldsymbol{y}) + g(\boldsymbol{x}) + (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y})^\top \boldsymbol{u} \tag{7.65a}$$

$$= \max_{\boldsymbol{u}} \min_{\boldsymbol{x},\boldsymbol{y}} f(\boldsymbol{y}) + g(\boldsymbol{x}) + (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y})^\top \boldsymbol{u}\,, \tag{7.65b}$$

where the last step of swapping max and min is due to the fact that $f(\boldsymbol{y})$ and $g(\boldsymbol{x})$ are convex functions. By splitting up the dot product term and collecting $\boldsymbol{x}$ and $\boldsymbol{y}$,

$$\max_{\boldsymbol{u}} \min_{\boldsymbol{x},\boldsymbol{y}} f(\boldsymbol{y}) + g(\boldsymbol{x}) + (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y})^\top \boldsymbol{u} \tag{7.66a}$$

$$= \max_{\boldsymbol{u}} \left[\min_{\boldsymbol{y}} -\boldsymbol{y}^\top \boldsymbol{u} + f(\boldsymbol{y})\right] + \left[\min_{\boldsymbol{x}} (\boldsymbol{A}\boldsymbol{x})^\top \boldsymbol{u} + g(\boldsymbol{x})\right] \tag{7.66b}$$

$$= \max_{\boldsymbol{u}} \left[\min_{\boldsymbol{y}} -\boldsymbol{y}^\top \boldsymbol{u} + f(\boldsymbol{y})\right] + \left[\min_{\boldsymbol{x}} \boldsymbol{x}^\top \boldsymbol{A}^\top \boldsymbol{u} + g(\boldsymbol{x})\right] \tag{7.66c}$$

Recall the convex conjugate (Definition 7.4) and the fact that dot products are symmetric,

For general inner products, $\boldsymbol{A}^\top$ is replaced by the adjoint $\boldsymbol{A}^*$.

$$\max_{\boldsymbol{u}} \left[\min_{\boldsymbol{y}} -\boldsymbol{y}^\top \boldsymbol{u} + f(\boldsymbol{y})\right] + \left[\min_{\boldsymbol{x}} \boldsymbol{x}^\top \boldsymbol{A}^\top \boldsymbol{u} + g(\boldsymbol{x})\right] \tag{7.67a}$$

$$= \max_{\boldsymbol{u}} -f^*(\boldsymbol{u}) - g^*(-\boldsymbol{A}^\top \boldsymbol{u})\,. \tag{7.67b}$$

Therefore, we have shown that

$$\min_{\boldsymbol{x}} f(\boldsymbol{A}\boldsymbol{x}) + g(\boldsymbol{x}) = \max_{\boldsymbol{u}} -f^*(\boldsymbol{u}) - g^*(-\boldsymbol{A}^\top \boldsymbol{u})\,. \tag{7.68}$$

The Legendre-Fenchel conjugate turns out to be quite useful for machine learning problems that can be expressed as convex optimization problems. In particular, for convex loss functions that apply independently to each example, the conjugate loss is a convenient way to derive a dual problem.

7.4 Further Reading

Continuous optimization is an active area of research, and we do not try to provide a comprehensive account of recent advances.

From a gradient descent perspective, there are two major weaknesses which each have their own set of literature. The first challenge is the fact that gradient descent is a first-order algorithm, and does not use information about the curvature of the surface. When there are long valleys, the gradient points perpendicularly to the direction of interest. The idea of momentum can be generalized to a general class of acceleration methods (Nesterov, 2018). Conjugate gradient methods avoid the issues faced by gradient descent by taking previous directions into account (Shewchuk, 1994). Second-order methods such as Newton methods use the Hessian to provide information about the curvature. Many of the choices for choosing step-sizes and ideas like momentum arise by considering the curvature of the objective function (Goh, 2017; Bottou et al., 2018). Quasi-Newton methods such as L-BFGS try to use cheaper computational methods to approximate the Hessian (Nocedal and Wright, 2006). Recently there has been interest in other metrics for computing descent directions, resulting in approaches such as mirror descent (Beck and Teboulle, 2003) and natural gradient (Toussaint, 2012).

The second challenge is to handle non-differentiable functions. Gradient methods are not well defined when there are kinks in the function. In these cases, *subgradient methods* can be used (Shor, 1985). For further information and algorithms for optimizing non-differentiable functions, we refer to the book by Bertsekas (1999). There is a vast amount of literature on different approaches for numerically solving continuous optimization problems, including algorithms for constrained optimization problems. Good starting points to appreciate this literature are the books by Luenberger (1969) and Bonnans et al. (2006). A recent survey of continuous optimization is provided by Bubeck (2015).

Hugo Gonçalves' blog is also a good resource for an easier introduction to Legendre–Fenchel transforms: `https://tinyurl.com/ydaal7hj`

Modern applications of machine learning often mean that the size of datasets prohibit the use of batch gradient descent, and hence stochastic gradient descent is the current workhorse of large-scale machine learning methods. Recent surveys of the literature include Hazan (2015) and Bottou et al. (2018).

For duality and convex optimization, the book by Boyd and Vandenberghe (2004) includes lectures and slides online. A more mathematical treatment is provided by Bertsekas (2009), and recent book by one of

the key researchers in the area of optimization is Nesterov (2018). Convex optimization is based upon convex analysis, and the reader interested in more foundational results about convex functions is referred to Rockafellar (1970), Hiriart-Urruty and Lemaréchal (2001), and Borwein and Lewis (2006). Legendre–Fenchel transforms are also covered in the aforementioned books on convex analysis, but a more beginner-friendly presentation is available at Zia et al. (2009). The role of Legendre–Fenchel transforms in the analysis of convex optimization algorithms is surveyed in Polyak (2016).

Exercises

7.1 Consider the univariate function

$$f(x) = x^3 + 6x^2 - 3x - 5.$$

Find its stationary points and indicate whether they are maximum, minimum, or saddle points.

7.2 Consider the update equation for stochastic gradient descent (Equation (7.15)). Write down the update when we use a mini-batch size of one.

7.3 Consider whether the following statements are true or false:

a. The intersection of any two convex sets is convex.
b. The union of any two convex sets is convex.
c. The difference of a convex set A from another convex set B is convex.

7.4 Consider whether the following statements are true or false:

a. The sum of any two convex functions is convex.
b. The difference of any two convex functions is convex.
c. The product of any two convex functions is convex.
d. The maximum of any two convex functions is convex.

7.5 Express the following optimization problem as a standard linear program in matrix notation

$$\max_{\boldsymbol{x}\in\mathbb{R}^2,\ \xi\in\mathbb{R}} \boldsymbol{p}^\top \boldsymbol{x} + \xi$$

subject to the constraints that $\xi \geqslant 0$, $x_0 \leqslant 0$ and $x_1 \leqslant 3$.

7.6 Consider the linear program illustrated in Figure 7.9,

$$\min_{\boldsymbol{x}\in\mathbb{R}^2} -\begin{bmatrix}5\\3\end{bmatrix}^\top \begin{bmatrix}x_1\\x_2\end{bmatrix}$$
$$\text{subject to } \begin{bmatrix}2 & 2\\2 & -4\\-2 & 1\\0 & -1\\0 & 1\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix} \leqslant \begin{bmatrix}33\\8\\5\\-1\\8\end{bmatrix}$$

Derive the dual linear program using Lagrange duality.

7.7 Consider the quadratic program illustrated in Figure 7.4,

$$\min_{\boldsymbol{x}\in\mathbb{R}^2} \frac{1}{2}\begin{bmatrix}x_1\\x_2\end{bmatrix}^\top \begin{bmatrix}2 & 1\\1 & 4\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix} + \begin{bmatrix}5\\3\end{bmatrix}^\top \begin{bmatrix}x_1\\x_2\end{bmatrix}$$
$$\text{subject to } \begin{bmatrix}1 & 0\\-1 & 0\\0 & 1\\0 & -1\end{bmatrix}\begin{bmatrix}x_1\\x_2\end{bmatrix} \leqslant \begin{bmatrix}1\\1\\1\\1\end{bmatrix}$$

Derive the dual quadratic program using Lagrange duality.

7.8 Consider the following convex optimization problem

$$\begin{aligned}\min_{\boldsymbol{w}\in\mathbb{R}^D} \quad & \frac{1}{2}\boldsymbol{w}^\top\boldsymbol{w}\\ \text{subject to} \quad & \boldsymbol{w}^\top\boldsymbol{x} \geqslant 1\,.\end{aligned}$$

Derive the Lagrangian dual by introducing the Lagrange multiplier λ.

7.9 Consider the negative entropy of $\boldsymbol{x} \in \mathbb{R}^D$,

$$f(\boldsymbol{x}) = \sum_{d=1}^{D} x_d \log x_d\,.$$

Derive the convex conjugate function $f^*(s)$, by assuming the standard dot product.

Hint: Take the gradient of an appropriate function and set the gradient to zero.

7.10 Consider the function

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^\top\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}^\top\boldsymbol{x} + c\,,$$

where $\boldsymbol{A}$ is strictly positive definite, which means that it is invertible. Derive the convex conjugate of $f(\boldsymbol{x})$.

Hint: Take the gradient of an appropriate function and set the gradient to zero.

7.11 The hinge loss (which is the loss used by the support vector machine) is given by

$$L(\alpha) = \max\{0, 1-\alpha\}\,,$$

If we are interested in applying gradient methods such as L-BFGS, and do not want to resort to subgradient methods, we need to smooth the kink in the hinge loss. Compute the convex conjugate of the hinge loss $L^*(\beta)$ where β is the dual variable. Add a ℓ_2 proximal term, and compute the conjugate of the resulting function

$$L^*(\beta) + \frac{\gamma}{2}\beta^2\,,$$

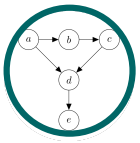
where γ is a given hyperparameter.

Part II

Central Machine Learning Problems

8

When Models Meet Data



In the first part of the book, we introduced the mathematics that form the foundations of many machine learning methods. The hope is that a reader would be able to learn the rudimentary forms of the language of mathematics from the first part, which we will now use to describe and discuss machine learning. The second part of the book introduces four pillars of machine learning:

- Regression (Chapter 9)
- Dimensionality reduction (Chapter 10)
- Density estimation (Chapter 11)
- Classification (Chapter 12)

The main aim of this part of the book is to illustrate how the mathematical concepts introduced in the first part of the book can be used to design machine learning algorithms that can be used to solve tasks within the remit of the four pillars. We do not intend to introduce advanced machine learning concepts, but instead to provide a set of practical methods that allow the reader to apply the knowledge they gained from the first part of the book. It also provides a gateway to the wider machine learning literature for readers already familiar with the mathematics.

8.1 Data, Models, and Learning

It is worth at this point, to pause and consider the problem that a machine learning algorithm is designed to solve. As discussed in Chapter 1, there are three major components of a machine learning system: data, models, and learning. The main question of machine learning is "What do we mean by good models?". The word *model* has many subtleties, and we will revisit it multiple times in this chapter. It is also not entirely obvious how to objectively define the word "good". One of the guiding principles of machine learning is that good models should perform well on unseen data. This requires us to define some performance metrics, such as accuracy or distance from ground truth, as well as figuring out ways to do well under these performance metrics. This chapter covers a few necessary bits and pieces of mathematical and statistical language that are commonly

model

Table 8.1 Example data from a fictitious human resource database that is not in a numerical format.

Name	Gender	Degree	Postcode	Age	Annual salary
Aditya	M	MSc	W21BG	36	89563
Bob	M	PhD	EC1A1BA	47	123543
Chloé	F	BEcon	SW1A1BH	26	23989
Daisuke	M	BSc	SE207AT	68	138769
Elisabeth	F	MBA	SE10AA	33	113888

used to talk about machine learning models. By doing so, we briefly outline the current best practices for training a model such that the resulting predictor does well on data that we have not yet seen.

As mentioned in Chapter 1, there are two different senses in which we use the phrase "machine learning algorithm": training and prediction. We will describe these ideas in this chapter, as well as the idea of selecting among different models. We will introduce the framework of empirical risk minimization in Section 8.2, the principle of maximum likelihood in Section 8.3, and the idea of probabilistic models in Section 8.4. We briefly outline a graphical language for specifying probabilistic models in Section 8.5 and finally discuss model selection in Section 8.6. The rest of this section expands upon the three main components of machine learning: data, models and learning.

8.1.1 Data as Vectors

We assume that our data can be read by a computer, and represented adequately in a numerical format. Data is assumed to be tabular (Figure 8.1), where we think of each row of the table as representing a particular instance or example, and each column to be a particular feature. In recent years, machine learning has been applied to many types of data that do not obviously come in the tabular numerical format, for example genomic sequences, text and image contents of a webpage, and social media graphs. We do not discuss the important and challenging aspects of identifying good features. Many of these aspects depend on domain expertise and require careful engineering, and, in recent years, they have been put under the umbrella of data science (Stray, 2016; Adhikari and DeNero, 2018).

Data is assumed to be in a tidy format (Wickham, 2014; Codd, 1990).

Even when we have data in tabular format, there are still choices to be made to obtain a numerical representation. For example, in Table 8.1, the gender column (a categorical variable) may be converted into numbers 0 representing "Male" and 1 representing "Female". Alternatively, the gender could be represented by numbers $-1, +1$, respectively (as shown in Table 8.2). Furthermore, it is often important to use domain knowledge when constructing the representation, such as knowing that university degrees progress from bachelor's to master's to PhD or realizing that the postcode provided is not just a string of characters but actually encodes an area in London. In Table 8.2, we converted the data from Table 8.1 to a numerical format, and each postcode is represented as two numbers,

Gender ID	Degree	Latitude (in degrees)	Longitude (in degrees)	Age	Annual Salary (in thousands)
-1	2	51.5073	0.1290	36	89.563
-1	3	51.5074	0.1275	47	123.543
+1	1	51.5071	0.1278	26	23.989
-1	1	51.5075	0.1281	68	138.769
+1	2	51.5074	0.1278	33	113.888

Table 8.2 Example data from a fictitious human resource database (see Table 8.1), converted to a numerical format.

a latitude and longitude. Even numerical data that could potentially be directly read into a machine learning algorithm should be carefully considered for units, scaling, and constraints. Without additional information, one should shift and scale all columns of the dataset such that they have an empirical mean of 0 and an empirical variance of 1. For the purposes of this book, we assume that a domain expert already converted data appropriately, i.e., each input $\boldsymbol{x}_n$ is a D-dimensional vector of real numbers, which are called *features*, *attributes*, or *covariates*. We consider a dataset to be of the form as illustrated by Table 8.2. Observe that we have dropped the Name column of Table 8.1 in the new numerical representation. There are two main reasons why this is desirable: (1) we do not expect the identifier (the Name) to be informative for a machine learning task; and (2) we may wish to anonymize the data to help protect the privacy of the employees.

feature
attribute
covariate

In this part of the book, we will use N to denote the number of examples in a dataset and index the examples with lowercase $n = 1, \dots, N$. We assume that we are given a set of numerical data, represented as an array of vectors (Table 8.2). Each row is a particular individual $\boldsymbol{x}_n$, often referred to as an *example* or *data point* in machine learning. The subscript n refers to the fact that this is the nth example out of a total of N examples in the dataset. Each column represents a particular feature of interest about the example, and we index the features as $d = 1, \dots, D$. Recall that data is represented as vectors, which means that each example (each data point) is a D-dimensional vector. The orientation of the table originates from the database community, but for some machine learning algorithms (e.g., in Chapter 10) it is more convenient to represent examples as column vectors.

example
data point

Let us consider the problem of predicting annual salary from age, based on the data in Table 8.2. This is called a supervised learning problem where we have a *label* y_n (the salary) associated with each example $\boldsymbol{x}_n$ (the age). The label y_n has various other names, including target, response variable, and annotation. A dataset is written as a set of example-label pairs $\{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_n, y_n), \dots, (\boldsymbol{x}_N, y_N)\}$. The table of examples $\{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\}$ is often concatenated, and written as $\boldsymbol{X} \in \mathbb{R}^{N \times D}$. Figure 8.1 illustrates the dataset consisting of the two rightmost columns of Table 8.2, where $x =$ age and $y =$ salary.

label

We use the concepts introduced in the first part of the book to formalize

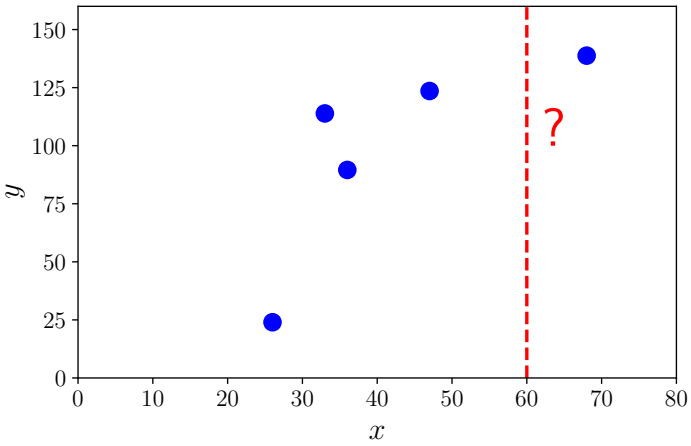


Figure 8.1 Toy data for linear regression. Training data in (x_n, y_n) pairs from the rightmost two columns of Table 8.2. We are interested in the salary of a person aged sixty ($x = 60$) illustrated as a vertical dashed red line, which is not part of the training data.

the machine learning problems such as that in the previous paragraph. Representing data as vectors $\boldsymbol{x}_n$ allows us to use concepts from linear algebra (introduced in Chapter 2). In many machine learning algorithms, we need to additionally be able to compare two vectors. As we will see in Chapters 9 and 12, computing the similarity or distance between two examples allows us to formalize the intuition that examples with similar features should have similar labels. The comparison of two vectors requires that we construct a geometry (explained in Chapter 3) and allows us to optimize the resulting learning problem using techniques from Chapter 7.

Since we have vector representations of data, we can manipulate data to find potentially better representations of it. We will discuss finding good representations in two ways: finding lower-dimensional approximations of the original feature vector, and using nonlinear higher-dimensional combinations of the original feature vector. In Chapter 10, we will see an example of finding a low-dimensional approximation of the original data space by finding the principal components. Finding principal components is closely related to concepts of eigenvalue and singular value decomposition as introduced in Chapter 4. For the high-dimensional representation, we will see an explicit *feature map* $\phi(\cdot)$ that allows us to represent inputs $\boldsymbol{x}_n$ using a higher-dimensional representation $\phi(\boldsymbol{x}_n)$. The main motivation for higher-dimensional representations is that we can construct new features as non-linear combinations of the original features, which in turn may make the learning problem easier. We will discuss the feature map in Section 9.2 and show how this feature map leads to a *kernel* in Section 12.4. In recent years, deep learning methods (Goodfellow et al., 2016) have shown promise in using the data itself to learn new good features and have been very successful in areas, such as computer vision, speech recognition, and natural language processing. We will not cover neural networks in this part of the book, but the reader is referred to

feature map

kernel

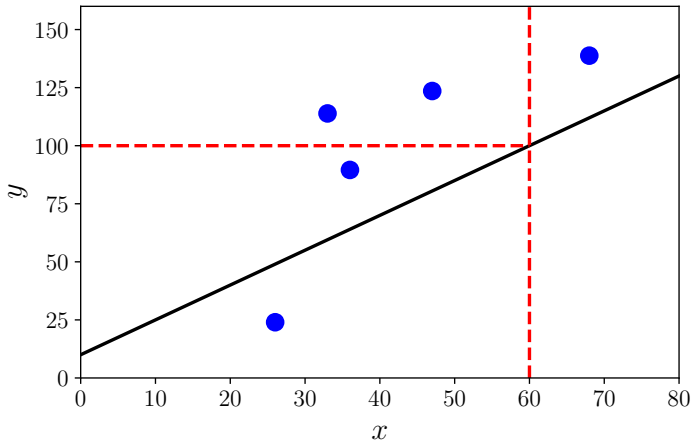


Figure 8.2 Example function (black solid diagonal line) and its prediction at $x = 60$, i.e., $f(60) = 100$.

Section 5.6 for the mathematical description of backpropagation, a key concept for training neural networks.

8.1.2 Models as Functions

Once we have data in an appropriate vector representation, we can get to the business of constructing a predictive function (known as a *predictor*). In Chapter 1, we did not yet have the language to be precise about models. Using the concepts from the first part of the book, we can now introduce what "model" means. We present two major approaches in this book: a predictor as a function, and a predictor as a probabilistic model. We describe the former here and the latter in the next subsection.

predictor

A predictor is a function that, when given a particular input example (in our case, a vector of features), produces an output. For now, consider the output to be a single number, i.e., a real-valued scalar output. This can be written as

$$f : \mathbb{R}^D \to \mathbb{R}\,, \tag{8.1}$$

where the input vector $\boldsymbol{x}$ is D-dimensional (has D features), and the function f then applied to it (written as $f(\boldsymbol{x})$) returns a real number. Figure 8.2 illustrates a possible function that can be used to compute the value of the prediction for input values x.

In this book, we do not consider the general case of all functions, which would involve the need for functional analysis. Instead, we consider the special case of linear functions

$$f(\boldsymbol{x}) = \boldsymbol{\theta}^\top \boldsymbol{x} + \theta_0 \tag{8.2}$$

for unknown $\boldsymbol{\theta}$ and θ_0. This restriction means that the contents of Chapters 2 and 3 suffice for precisely stating the notion of a predictor for the non-probabilistic (in contrast to the probabilistic view described next)

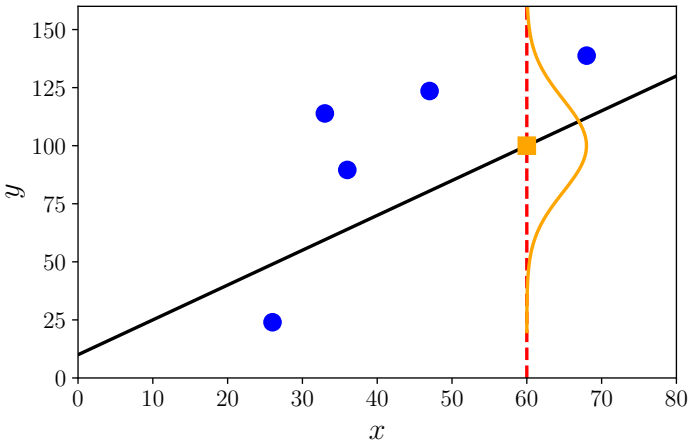


Figure 8.3 Example function (black solid diagonal line) and its predictive uncertainty at $x = 60$ (drawn as a Gaussian).

view of machine learning. Linear functions strike a good balance between the generality of the problems that can be solved and the amount of background mathematics that is needed.

8.1.3 Models as Probability Distributions

We often consider data to be noisy observations of some true underlying effect, and hope that by applying machine learning we can identify the signal from the noise. This requires us to have a language for quantifying the effect of noise. We often would also like to have predictors that express some sort of uncertainty, e.g., to quantify the confidence we have about the value of the prediction for a particular test data point. As we have seen in Chapter 6, probability theory provides a language for quantifying uncertainty. Figure 8.3 illustrates the predictive uncertainty of the function as a Gaussian distribution.

Instead of considering a predictor as a single function, we could consider predictors to be probabilistic models, i.e., models describing the distribution of possible functions. We limit ourselves in this book to the special case of distributions with finite-dimensional parameters, which allows us to describe probabilistic models without needing stochastic processes and random measures. For this special case, we can think about probabilistic models as multivariate probability distributions, which already allow for a rich class of models.

We will introduce how to use concepts from probability (Chapter 6) to define machine learning models in Section 8.4, and introduce a graphical language for describing probabilistic models in a compact way in Section 8.5.

8.1.4 Learning is Finding Parameters

The goal of learning is to find a model and its corresponding parameters such that the resulting predictor will perform well on unseen data. There are conceptually three distinct algorithmic phases when discussing machine learning algorithms:

1. Prediction or inference
2. Training or parameter estimation
3. Hyperparameter tuning or model selection

The prediction phase is when we use a trained predictor on previously unseen test data. In other words, the parameters and model choice is already fixed and the predictor is applied to new vectors representing new input data points. As outlined in Chapter 1 and the previous subsection, we will consider two schools of machine learning in this book, corresponding to whether the predictor is a function or a probabilistic model. When we have a probabilistic model (discussed further in Section 8.4) the prediction phase is called inference.

Remark. Unfortunately, there is no agreed upon naming for the different algorithmic phases. The word "inference" is sometimes also used to mean parameter estimation of a probabilistic model, and less often may be also used to mean prediction for non-probabilistic models. ♢

The training or parameter estimation phase is when we adjust our predictive model based on training data. We would like to find good predictors given training data, and there are two main strategies for doing so: finding the best predictor based on some measure of quality (sometimes called finding a point estimate), or using Bayesian inference. Finding a point estimate can be applied to both types of predictors, but Bayesian inference requires probabilistic models.

For the non-probabilistic model, we follow the principle of *empirical risk minimization*, which we describe in Section 8.2. Empirical risk minimization directly provides an optimization problem for finding good parameters. With a statistical model, the principle of *maximum likelihood* is used to find a good set of parameters (Section 8.3). We can additionally model the uncertainty of parameters using a probabilistic model, which we will look at in more detail in Section 8.4.

empirical risk minimization

maximum likelihood

We use numerical methods to find good parameters that "fit" the data, and most training methods can be thought of as hill-climbing approaches to find the maximum of an objective, for example the maximum of a likelihood. To apply hill-climbing approaches we use the gradients described in Chapter 5 and implement numerical optimization approaches from Chapter 7.

The convention in optimization is to minimize objectives. Hence, there is often an extra minus sign in machine learning objectives.

As mentioned in Chapter 1, we are interested in learning a model based on data such that it performs well on future data. It is not enough for

the model to only fit the training data well, the predictor needs to perform well on unseen data. We simulate the behavior of our predictor on future unseen data using *cross-validation* (Section 8.2.4). As we will see in this chapter, to achieve the goal of performing well on unseen data, we will need to balance between fitting well on training data and finding "simple" explanations of the phenomenon. This trade-off is achieved using regularization (Section 8.2.3) or by adding a prior (Section 8.3.2). In philosophy, this is considered to be neither induction nor deduction, but is called *abduction*. According to the *Stanford Encyclopedia of Philosophy*, abduction is the process of inference to the best explanation (Douven, 2017).

cross-validation

abduction

A good movie title is "AI abduction".

We often need to make high-level modeling decisions about the structure of the predictor, such as the number of components to use or the class of probability distributions to consider. The choice of the number of components is an example of a *hyperparameter*, and this choice can affect the performance of the model significantly. The problem of choosing among different models is called *model selection*, which we describe in Section 8.6. For non-probabilistic models, model selection is often done using *nested cross-validation*, which is described in Section 8.6.1. We also use model selection to choose hyperparameters of our model.

hyperparameter

model selection

nested cross-validation

Remark. The distinction between parameters and hyperparameters is somewhat arbitrary, and is mostly driven by the distinction between what can be numerically optimized versus what needs to use search techniques. Another way to consider the distinction is to consider parameters as the explicit parameters of a probabilistic model, and to consider hyperparameters (higher-level parameters) as parameters that control the distribution of these explicit parameters. ♢

In the following sections, we will look at three flavors of machine learning: empirical risk minimization (Section 8.2), the principle of maximum likelihood (Section 8.3), and probabilistic modeling (Section 8.4).

8.2 Empirical Risk Minimization

After having all the mathematics under our belt, we are now in a position to introduce what it means to learn. The "learning" part of machine learning boils down to estimating parameters based on training data.

In this section, we consider the case of a predictor that is a function, and consider the case of probabilistic models in Section 8.3. We describe the idea of empirical risk minimization, which was originally popularized by the proposal of the support vector machine (described in Chapter 12). However, its general principles are widely applicable and allow us to ask the question of what is learning without explicitly constructing probabilistic models. There are four main design choices, which we will cover in detail in the following subsections:

Section 8.2.1 What is the set of functions we allow the predictor to take?
Section 8.2.2 How do we measure how well the predictor performs on the training data?
Section 8.2.3 How do we construct predictors from only training data that performs well on unseen test data?
Section 8.2.4 What is the procedure for searching over the space of models?

8.2.1 Hypothesis Class of Functions

Assume we are given N examples $\boldsymbol{x}_n \in \mathbb{R}^D$ and corresponding scalar labels $y_n \in \mathbb{R}$. We consider the supervised learning setting, where we obtain pairs $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)$. Given this data, we would like to estimate a predictor $f(\cdot, \boldsymbol{\theta}) : \mathbb{R}^D \to \mathbb{R}$, parametrized by $\boldsymbol{\theta}$. We hope to be able to find a good parameter $\boldsymbol{\theta}^*$ such that we fit the data well, that is,

$$f(\boldsymbol{x}_n, \boldsymbol{\theta}^*) \approx y_n \quad \text{for all} \quad n = 1, \ldots, N \,. \tag{8.3}$$

In this section, we use the notation $\hat{y}_n = f(\boldsymbol{x}_n, \boldsymbol{\theta}^*)$ to represent the output of the predictor.

Remark. For ease of presentation, we will describe empirical risk minimization in terms of supervised learning (where we have labels). This simplifies the definition of the hypothesis class and the loss function. It is also common in machine learning to choose a parametrized class of functions, for example affine functions. ♢

Example 8.1
We introduce the problem of ordinary least-squares regression to illustrate empirical risk minimization. A more comprehensive account of regression is given in Chapter 9. When the label y_n is real-valued, a popular choice of function class for predictors is the set of affine functions. We choose a more compact notation for an affine function by concatenating an additional unit feature $x^{(0)} = 1$ to $\boldsymbol{x}_n$, i.e., $\boldsymbol{x}_n = [1, x_n^{(1)}, x_n^{(2)}, \ldots, x_n^{(D)}]^\top$. The parameter vector is correspondingly $\boldsymbol{\theta} = [\theta_0, \theta_1, \theta_2, \ldots, \theta_D]^\top$, allowing us to write the predictor as a linear function

Affine functions are often referred to as linear functions in machine learning.

$$f(\boldsymbol{x}_n, \boldsymbol{\theta}) = \boldsymbol{\theta}^\top \boldsymbol{x}_n \,. \tag{8.4}$$

This linear predictor is equivalent to the affine model

$$f(\boldsymbol{x}_n, \boldsymbol{\theta}) = \theta_0 + \sum_{d=1}^{D} \theta_d x_n^{(d)} \,. \tag{8.5}$$

The predictor takes the vector of features representing a single example $\boldsymbol{x}_n$ as input and produces a real-valued output, i.e., $f : \mathbb{R}^{D+1} \to \mathbb{R}$. The

previous figures in this chapter had a straight line as a predictor, which means that we have assumed an affine function.

Instead of a linear function, we may wish to consider non-linear functions as predictors. Recent advances in neural networks allow for efficient computation of more complex non-linear function classes.

Given the class of functions, we want to search for a good predictor. We now move on to the second ingredient of empirical risk minimization: how to measure how well the predictor fits the training data.

8.2.2 Loss Function for Training

loss function

Consider the label y_n for a particular example; and the corresponding prediction $\hat{y}_n$ that we make based on $\boldsymbol{x}_n$. To define what it means to fit the data well, we need to specify a *loss function* $\ell(y_n, \hat{y}_n)$ that takes the ground truth label and the prediction as input and produces a non-negative number (referred to as the loss) representing how much error we have made on this particular prediction. Our goal for finding a good parameter vector $\boldsymbol{\theta}^*$ is to minimize the average loss on the set of N training examples.

The expression "error" is often used to mean loss.

independent and identically distributed

One assumption that is commonly made in machine learning is that the set of examples $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)$ is *independent and identically distributed*. The word independent (Section 6.4.5) means that two data points $(\boldsymbol{x}_i, y_i)$ and $(\boldsymbol{x}_j, y_j)$ do not statistically depend on each other, meaning that the empirical mean is a good estimate of the population mean (Section 6.4.1). This implies that we can use the empirical mean of the loss on the training data. For a given *training set* $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$, we introduce the notation of an example matrix $\boldsymbol{X} := [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]^\top \in \mathbb{R}^{N \times D}$ and a label vector $\boldsymbol{y} := [y_1, \ldots, y_N]^\top \in \mathbb{R}^N$. Using this matrix notation the average loss is given by

training set

$$\mathbf{R}_{\text{emp}}(f, \boldsymbol{X}, \boldsymbol{y}) = \frac{1}{N} \sum_{n=1}^{N} \ell(y_n, \hat{y}_n)\,, \tag{8.6}$$

empirical risk

where $\hat{y}_n = f(\boldsymbol{x}_n, \boldsymbol{\theta})$. Equation (8.6) is called the *empirical risk* and depends on three arguments, the predictor f and the data $\boldsymbol{X}, \boldsymbol{y}$. This general strategy for learning is called *empirical risk minimization*.

empirical risk minimization

Example 8.2 (Least-Squares Loss)
Continuing the example of least-squares regression, we specify that we measure the cost of making an error during training using the squared loss $\ell(y_n, \hat{y}_n) = (y_n - \hat{y}_n)^2$. We wish to minimize the empirical risk (8.6),

which is the average of the losses over the data

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^D} \frac{1}{N}\sum_{n=1}^{N}(y_n - f(\boldsymbol{x}_n,\boldsymbol{\theta}))^2, \tag{8.7}$$

where we substituted the predictor $\hat{y}_n = f(\boldsymbol{x}_n,\boldsymbol{\theta})$. By using our choice of a linear predictor $f(\boldsymbol{x}_n,\boldsymbol{\theta}) = \boldsymbol{\theta}^\top\boldsymbol{x}_n$, we obtain the optimization problem

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^D} \frac{1}{N}\sum_{n=1}^{N}(y_n - \boldsymbol{\theta}^\top\boldsymbol{x}_n)^2\,. \tag{8.8}$$

This equation can be equivalently expressed in matrix form

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^D} \frac{1}{N}\left\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}\right\|^2\,. \tag{8.9}$$

This is known as the *least-squares problem*. There exists a closed-form analytic solution for this by solving the normal equations, which we will discuss in Section 9.2.

least-squares problem

We are not interested in a predictor that only performs well on the training data. Instead, we seek a predictor that performs well (has low risk) on unseen test data. More formally, we are interested in finding a predictor f (with parameters fixed) that minimizes the *expected risk*

expected risk

$$\mathbf{R}_{\text{true}}(f) = \mathbb{E}_{\boldsymbol{x},y}[\ell(y, f(\boldsymbol{x}))]\,, \tag{8.10}$$

where y is the label and $f(\boldsymbol{x})$ is the prediction based on the example $\boldsymbol{x}$. The notation $\mathbf{R}_{\text{true}}(f)$ indicates that this is the true risk if we had access to an infinite amount of data. The expectation is over the (infinite) set of all possible data and labels. There are two practical questions that arise from our desire to minimize expected risk, which we address in the following two subsections:

Another phrase commonly used for expected risk is "population risk".

- How should we change our training procedure to generalize well?
- How do we estimate expected risk from (finite) data?

Remark. Many machine learning tasks are specified with an associated performance measure, e.g., accuracy of prediction or root mean squared error. The performance measure could be more complex, be cost sensitive, and capture details about the particular application. In principle, the design of the loss function for empirical risk minimization should correspond directly to the performance measure specified by the machine learning task. In practice, there is often a mismatch between the design of the loss function and the performance measure. This could be due to issues such as ease of implementation or efficiency of optimization. ♢

8.2.3 Regularization to Reduce Overfitting

This section describes an addition to empirical risk minimization that allows it to generalize well (approximately minimizing expected risk). Recall that the aim of training a machine learning predictor is so that we can perform well on unseen data, i.e., the predictor generalizes well. We simulate this unseen data by holding out a proportion of the whole dataset. This hold out set is referred to as the *test set*. Given a sufficiently rich class of functions for the predictor f, we can essentially memorize the training data to obtain zero empirical risk. While this is great to minimize the loss (and therefore the risk) on the training data, we would not expect the predictor to generalize well to unseen data. In practice, we have only a finite set of data, and hence we split our data into a training and a test set. The training set is used to fit the model, and the test set (not seen by the machine learning algorithm during training) is used to evaluate generalization performance. It is important for the user to not cycle back to a new round of training after having observed the test set. We use the subscripts $_{\text{train}}$ and $_{\text{test}}$ to denote the training and test sets, respectively. We will revisit this idea of using a finite dataset to evaluate expected risk in Section 8.2.4.

test set

Even knowing only the performance of the predictor on the test set leaks information (Blum and Hardt, 2015).

overfitting

It turns out that empirical risk minimization can lead to *overfitting*, i.e., the predictor fits too closely to the training data and does not generalize well to new data (Mitchell, 1997). This general phenomenon of having very small average loss on the training set but large average loss on the test set tends to occur when we have little data and a complex hypothesis class. For a particular predictor f (with parameters fixed), the phenomenon of overfitting occurs when the risk estimate from the training data $\mathbf{R}_{\text{emp}}(f, \boldsymbol{X}_{\text{train}}, \boldsymbol{y}_{\text{train}})$ underestimates the expected risk $\mathbf{R}_{\text{true}}(f)$. Since we estimate the expected risk $\mathbf{R}_{\text{true}}(f)$ by using the empirical risk on the test set $\mathbf{R}_{\text{emp}}(f, \boldsymbol{X}_{\text{test}}, \boldsymbol{y}_{\text{test}})$ if the test risk is much larger than the training risk, this is an indication of overfitting. We revisit the idea of overfitting in Section 8.3.3.

Therefore, we need to somehow bias the search for the minimizer of empirical risk by introducing a penalty term, which makes it harder for the optimizer to return an overly flexible predictor. In machine learning, the penalty term is referred to as *regularization*. Regularization is a way to compromise between accurate solution of empirical risk minimization and the size or complexity of the solution.

regularization

Example 8.3 (Regularized Least Squares)
Regularization is an approach that discourages complex or extreme solutions to an optimization problem. The simplest regularization strategy is

to replace the least-squares problem

$$\min_{\boldsymbol{\theta}} \frac{1}{N} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}\|^2 \,. \tag{8.11}$$

in the previous example with the "regularized" problem by adding a penalty term involving only $\boldsymbol{\theta}$:

$$\min_{\boldsymbol{\theta}} \frac{1}{N} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}\|^2 + \lambda \|\boldsymbol{\theta}\|^2 \,. \tag{8.12}$$

The additional term $\|\boldsymbol{\theta}\|^2$ is called the *regularizer*, and the parameter λ is the *regularization parameter*. The regularization parameter trades off minimizing the loss on the training set and the magnitude of the parameters $\boldsymbol{\theta}$. It often happens that the magnitude of the parameter values becomes relatively large if we run into overfitting (Bishop, 2006).

regularizer

regularization parameter

The regularization term is sometimes called the *penalty term*, which biases the vector $\boldsymbol{\theta}$ to be closer to the origin. The idea of regularization also appears in probabilistic models as the prior probability of the parameters. Recall from Section 6.6 that for the posterior distribution to be of the same form as the prior distribution, the prior and the likelihood need to be conjugate. We will revisit this idea in Section 8.3.2. We will see in Chapter 12 that the idea of the regularizer is equivalent to the idea of a large margin.

penalty term

8.2.4 Cross-Validation to Assess the Generalization Performance

We mentioned in the previous section that we measure the generalization error by estimating it by applying the predictor on test data. This data is also sometimes referred to as the *validation set*. The validation set is a subset of the available training data that we keep aside. A practical issue with this approach is that the amount of data is limited, and ideally we would use as much of the data available to train the model. This would require us to keep our validation set $\mathcal{V}$ small, which then would lead to a noisy estimate (with high variance) of the predictive performance. One solution to these contradictory objectives (large training set, large validation set) is to use *cross-validation*. K-fold cross-validation effectively partitions the data into K chunks, $K-1$ of which form the training set $\mathcal{R}$, and the last chunk serves as the validation set $\mathcal{V}$ (similar to the idea outlined previously). Cross-validation iterates through (ideally) all combinations of assignments of chunks to $\mathcal{R}$ and $\mathcal{V}$; see Figure 8.4. This procedure is repeated for all K choices for the validation set, and the performance of the model from the K runs is averaged.

validation set

cross-validation

We partition our dataset into two sets $\mathcal{D} = \mathcal{R} \cup \mathcal{V}$, such that they do not overlap ($\mathcal{R} \cap \mathcal{V} = \emptyset$), where $\mathcal{V}$ is the validation set, and train our model on $\mathcal{R}$. After training, we assess the performance of the predictor f on the

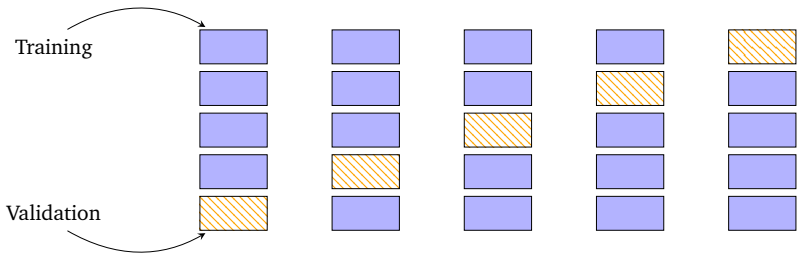


Figure 8.4 K-fold cross-validation. The dataset is divided into $K = 5$ chunks, $K - 1$ of which serve as the training set (blue) and one as the validation set (orange hatch).

validation set $\mathcal{V}$ (e.g., by computing root mean square error (RMSE) of the trained model on the validation set). More precisely, for each partition k the training data $\mathcal{R}^{(k)}$ produces a predictor $f^{(k)}$, which is then applied to validation set $\mathcal{V}^{(k)}$ to compute the empirical risk $R(f^{(k)}, \mathcal{V}^{(k)})$. We cycle through all possible partitionings of validation and training sets and compute the average generalization error of the predictor. Cross-validation approximates the expected generalization error

$$\mathbb{E}_{\mathcal{V}}[R(f, \mathcal{V})] \approx \frac{1}{K} \sum_{k=1}^{K} R(f^{(k)}, \mathcal{V}^{(k)}) \,, \tag{8.13}$$

where $R(f^{(k)}, \mathcal{V}^{(k)})$ is the risk (e.g., RMSE) on the validation set $\mathcal{V}^{(k)}$ for predictor $f^{(k)}$. The approximation has two sources: first, due to the finite training set, which results in not the best possible $f^{(k)}$; and second, due to the finite validation set, which results in an inaccurate estimation of the risk $R(f^{(k)}, \mathcal{V}^{(k)})$. A potential disadvantage of K-fold cross-validation is the computational cost of training the model K times, which can be burdensome if the training cost is computationally expensive. In practice, it is often not sufficient to look at the direct parameters alone. For example, we need to explore multiple complexity parameters (e.g., multiple regularization parameters), which may not be direct parameters of the model. Evaluating the quality of the model, depending on these hyperparameters, may result in a number of training runs that is exponential in the number of model parameters. One can use nested cross-validation (Section 8.6.1) to search for good hyperparameters.

embarrassingly parallel

However, cross-validation is an *embarrassingly parallel* problem, i.e., little effort is needed to separate the problem into a number of parallel tasks. Given sufficient computing resources (e.g., cloud computing, server farms), cross-validation does not require longer than a single performance assessment.

In this section, we saw that empirical risk minimization is based on the following concepts: the hypothesis class of functions, the loss function and regularization. In Section 8.3, we will see the effect of using a probability distribution to replace the idea of loss functions and regularization.

8.2.5 Further Reading

Due to the fact that the original development of empirical risk minimization (Vapnik, 1998) was couched in heavily theoretical language, many of the subsequent developments have been theoretical. The area of study is called *statistical learning theory* (Vapnik, 1999; Evgeniou et al., 2000; Hastie et al., 2001; von Luxburg and Schölkopf, 2011). A recent machine learning textbook that builds on the theoretical foundations and develops efficient learning algorithms is Shalev-Shwartz and Ben-David (2014).

statistical learning theory

The concept of regularization has its roots in the solution of ill-posed inverse problems (Neumaier, 1998). The approach presented here is called *Tikhonov regularization*, and there is a closely related constrained version called Ivanov regularization. Tikhonov regularization has deep relationships to the bias-variance trade-off and feature selection (Bühlmann and Van De Geer, 2011). An alternative to cross-validation is bootstrap and jackknife (Efron and Tibshirani, 1993; Davidson and Hinkley, 1997; Hall, 1992).

Tikhonov regularization

Thinking about empirical risk minimization (Section 8.2) as "probability free" is incorrect. There is an underlying unknown probability distribution $p(\boldsymbol{x}, y)$ that governs the data generation. However, the approach of empirical risk minimization is agnostic to that choice of distribution. This is in contrast to standard statistical approaches that explicitly require the knowledge of $p(\boldsymbol{x}, y)$. Furthermore, since the distribution is a joint distribution on both examples $\boldsymbol{x}$ and labels y, the labels can be nondeterministic. In contrast to standard statistics we do not need to specify the noise distribution for the labels y.

8.3 Parameter Estimation

In Section 8.2, we did not explicitly model our problem using probability distributions. In this section, we will see how to use probability distributions to model our uncertainty due to the observation process and our uncertainty in the parameters of our predictors. In Section 8.3.1, we introduce the likelihood, which is analogous to the concept of loss functions (Section 8.2.2) in empirical risk minimization. The concept of priors (Section 8.3.2) is analogous to the concept of regularization (Section 8.2.3).

8.3.1 Maximum Likelihood Estimation

The idea behind *maximum likelihood estimation* (MLE) is to define a function of the parameters that enables us to find a model that fits the data well. The estimation problem is focused on the *likelihood* function, or more precisely its negative logarithm. For data represented by a random variable $\boldsymbol{x}$ and for a family of probability densities $p(\boldsymbol{x} \mid \boldsymbol{\theta})$ parametrized by $\boldsymbol{\theta}$, the *negative log-likelihood* is given by

maximum likelihood estimation

likelihood

negative log-likelihood

$$\mathcal{L}_{\boldsymbol{x}}(\boldsymbol{\theta}) = -\log p(\boldsymbol{x} \mid \boldsymbol{\theta}) \,. \tag{8.14}$$

The notation $\mathcal{L}_{\boldsymbol{x}}(\boldsymbol{\theta})$ emphasizes the fact that the parameter $\boldsymbol{\theta}$ is varying and the data $\boldsymbol{x}$ is fixed. We very often drop the reference to $\boldsymbol{x}$ when writing the negative log-likelihood, as it is really a function of $\boldsymbol{\theta}$, and write it as $\mathcal{L}(\boldsymbol{\theta})$ when the random variable representing the uncertainty in the data is clear from the context.

Let us interpret what the probability density $p(\boldsymbol{x} \mid \boldsymbol{\theta})$ is modeling for a fixed value of $\boldsymbol{\theta}$. It is a distribution that models the uncertainty of the data for a given parameter setting. For a given dataset $\boldsymbol{x}$, the likelihood allows us to express preferences about different settings of the parameters $\boldsymbol{\theta}$, and we can choose the setting that more "likely" has generated the data.

In a complementary view, if we consider the data to be fixed (because it has been observed), and we vary the parameters $\boldsymbol{\theta}$, what does $\mathcal{L}(\boldsymbol{\theta})$ tell us? It tells us how likely a particular setting of $\boldsymbol{\theta}$ is for the observations $\boldsymbol{x}$. Based on this second view, the maximum likelihood estimator gives us the most likely parameter $\boldsymbol{\theta}$ for the set of data.

We consider the supervised learning setting, where we obtain pairs $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)$ with $\boldsymbol{x}_n \in \mathbb{R}^D$ and labels $y_n \in \mathbb{R}$. We are interested in constructing a predictor that takes a feature vector $\boldsymbol{x}_n$ as input and produces a prediction y_n (or something close to it), i.e., given a vector $\boldsymbol{x}_n$ we want the probability distribution of the label y_n. In other words, we specify the conditional probability distribution of the labels given the examples for the particular parameter setting $\boldsymbol{\theta}$.

Example 8.4

The first example that is often used is to specify that the conditional probability of the labels given the examples is a Gaussian distribution. In other words, we assume that we can explain our observation uncertainty by independent Gaussian noise (refer to Section 6.5) with zero mean, $\varepsilon_n \sim \mathcal{N}\big(0,\, \sigma^2\big)$. We further assume that the linear model $\boldsymbol{x}_n^\top \boldsymbol{\theta}$ is used for prediction. This means we specify a Gaussian likelihood for each example label pair $(\boldsymbol{x}_n, y_n)$,

$$p(y_n \mid \boldsymbol{x}_n, \boldsymbol{\theta}) = \mathcal{N}\big(y_n \mid \boldsymbol{x}_n^\top \boldsymbol{\theta},\, \sigma^2\big) \,. \tag{8.15}$$

An illustration of a Gaussian likelihood for a given parameter $\boldsymbol{\theta}$ is shown in Figure 8.3. We will see in Section 9.2 how to explicitly expand the preceding expression out in terms of the Gaussian distribution.

independent and identically distributed

We assume that the set of examples $(x_1, y_1), \ldots, (x_N, y_N)$ are *independent and identically distributed* (i.i.d.). The word "independent" (Section 6.4.5) implies that the likelihood involving the whole dataset ($\mathcal{Y} = \{y_1, \ldots, y_N\}$ and $\mathcal{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$) factorizes into a product of the likelihoods of

each individual example

$$p(\mathcal{Y} \mid \mathcal{X}, \boldsymbol{\theta}) = \prod_{n=1}^{N} p(y_n \mid \boldsymbol{x}_n, \boldsymbol{\theta}) \,, \tag{8.16}$$

where $p(y_n \mid \boldsymbol{x}_n, \boldsymbol{\theta})$ is a particular distribution (which was Gaussian in Example 8.4). The expression "identically distributed" means that each term in the product (8.16) is of the same distribution, and all of them share the same parameters. It is often easier from an optimization viewpoint to compute functions that can be decomposed into sums of simpler functions. Hence, in machine learning we often consider the negative log-likelihood

Recall $\log(ab) = \log(a) + \log(b)$

$$\mathcal{L}(\boldsymbol{\theta}) = -\log p(\mathcal{Y} \mid \mathcal{X}, \boldsymbol{\theta}) = -\sum_{n=1}^{N} \log p(y_n \mid \boldsymbol{x}_n, \boldsymbol{\theta}) \,. \tag{8.17}$$

While it is temping to interpret the fact that $\boldsymbol{\theta}$ is on the right of the conditioning in $p(y_n|\boldsymbol{x}_n, \boldsymbol{\theta})$ (8.15), and hence should be interpreted as observed and fixed, this interpretation is incorrect. The negative log-likelihood $\mathcal{L}(\boldsymbol{\theta})$ is a function of $\boldsymbol{\theta}$. Therefore, to find a good parameter vector $\boldsymbol{\theta}$ that explains the data $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)$ well, minimize the negative log-likelihood $\mathcal{L}(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$.

Remark. The negative sign in (8.17) is a historical artifact that is due to the convention that we want to maximize likelihood, but numerical optimization literature tends to study minimization of functions. ♢

Example 8.5

Continuing on our example of Gaussian likelihoods (8.15), the negative log-likelihood can be rewritten as

$$\begin{aligned}
\mathcal{L}(\boldsymbol{\theta}) &= -\sum_{n=1}^{N} \log p(y_n \mid \boldsymbol{x}_n, \boldsymbol{\theta}) = -\sum_{n=1}^{N} \log \mathcal{N}\big(y_n \mid \boldsymbol{x}_n^\top \boldsymbol{\theta},\, \sigma^2\big) && \text{(8.18a)} \\
&= -\sum_{n=1}^{N} \log \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_n - \boldsymbol{x}_n^\top \boldsymbol{\theta})^2}{2\sigma^2}\right) && \text{(8.18b)} \\
&= -\sum_{n=1}^{N} \log \exp\left(-\frac{(y_n - \boldsymbol{x}_n^\top \boldsymbol{\theta})^2}{2\sigma^2}\right) - \sum_{n=1}^{N} \log \frac{1}{\sqrt{2\pi\sigma^2}} && \text{(8.18c)} \\
&= \frac{1}{2\sigma^2} \sum_{n=1}^{N} (y_n - \boldsymbol{x}_n^\top \boldsymbol{\theta})^2 - \sum_{n=1}^{N} \log \frac{1}{\sqrt{2\pi\sigma^2}} \,. && \text{(8.18d)}
\end{aligned}$$

As σ is given, the second term in (8.18d) is constant, and minimizing $\mathcal{L}(\boldsymbol{\theta})$ corresponds to solving the least-squares problem (compare with (8.8)) expressed in the first term.

It turns out that for Gaussian likelihoods the resulting optimization

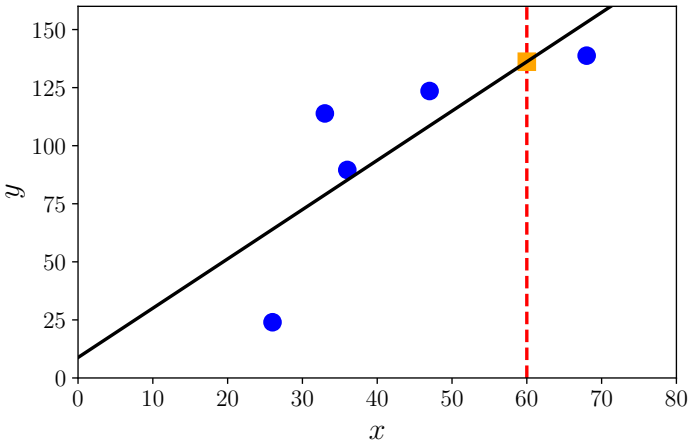


Figure 8.5 For the given data, the maximum likelihood estimate of the parameters results in the black diagonal line. The orange square shows the value of the maximum likelihood prediction at $x = 60$.

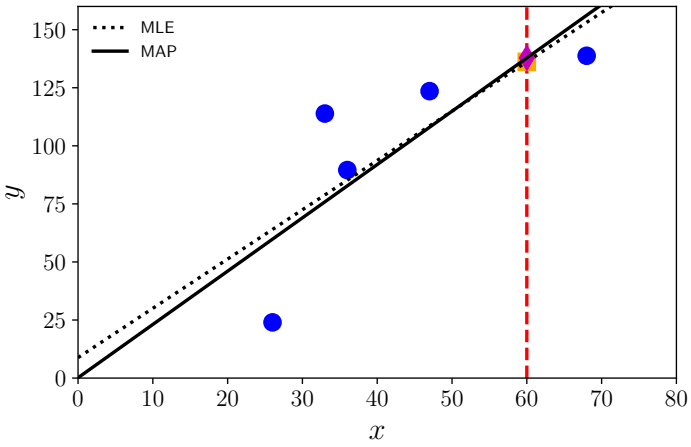


Figure 8.6 Comparing the predictions with the maximum likelihood estimate and the MAP estimate at $x = 60$. The prior biases the slope to be less steep and the intercept to be closer to zero. In this example, the bias that moves the intercept closer to zero actually increases the slope.

problem corresponding to maximum likelihood estimation has a closed-form solution. We will see more details on this in Chapter 9. Figure 8.5 shows a regression dataset and the function that is induced by the maximum-likelihood parameters. Maximum likelihood estimation may suffer from overfitting (Section 8.3.3), analogous to unregularized empirical risk minimization (Section 8.2.3). For other likelihood functions, i.e., if we model our noise with non-Gaussian distributions, maximum likelihood estimation may not have a closed-form analytic solution. In this case, we resort to numerical optimization methods discussed in Chapter 7.

8.3.2 Maximum A Posteriori Estimation

If we have prior knowledge about the distribution of the parameters $\boldsymbol{\theta}$, we can multiply an additional term to the likelihood. This additional term is a prior probability distribution on parameters $p(\boldsymbol{\theta})$. For a given prior, after

observing some data $\boldsymbol{x}$, how should we update the distribution of $\boldsymbol{\theta}$? In other words, how should we represent the fact that we have more specific knowledge of $\boldsymbol{\theta}$ after observing data $\boldsymbol{x}$? Bayes' theorem, as discussed in Section 6.3, gives us a principled tool to update our probability distributions of random variables. It allows us to compute a *posterior* distribution $p(\boldsymbol{\theta} \mid \boldsymbol{x})$ (the more specific knowledge) on the parameters $\boldsymbol{\theta}$ from general *prior* statements (prior distribution) $p(\boldsymbol{\theta})$ and the function $p(\boldsymbol{x} \mid \boldsymbol{\theta})$ that links the parameters $\boldsymbol{\theta}$ and the observed data $\boldsymbol{x}$ (called the *likelihood*):

posterior

prior

likelihood

$$p(\boldsymbol{\theta} \mid \boldsymbol{x}) = \frac{p(\boldsymbol{x} \mid \boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\boldsymbol{x})} . \tag{8.19}$$

Recall that we are interested in finding the parameter $\boldsymbol{\theta}$ that maximizes the posterior. Since the distribution $p(\boldsymbol{x})$ does not depend on $\boldsymbol{\theta}$, we can ignore the value of the denominator for the optimization and obtain

$$p(\boldsymbol{\theta} \mid \boldsymbol{x}) \propto p(\boldsymbol{x} \mid \boldsymbol{\theta})p(\boldsymbol{\theta}) . \tag{8.20}$$

The preceding proportion relation hides the density of the data $p(\boldsymbol{x})$, which may be difficult to estimate. Instead of estimating the minimum of the negative log-likelihood, we now estimate the minimum of the negative log-posterior, which is referred to as *maximum a posteriori estimation* (*MAP estimation*). An illustration of the effect of adding a zero-mean Gaussian prior is shown in Figure 8.6.

maximum a posteriori estimation

MAP estimation

Example 8.6

In addition to the assumption of Gaussian likelihood in the previous example, we assume that the parameter vector is distributed as a multivariate Gaussian with zero mean, i.e., $p(\boldsymbol{\theta}) = \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$, where $\boldsymbol{\Sigma}$ is the covariance matrix (Section 6.5). Note that the conjugate prior of a Gaussian is also a Gaussian (Section 6.6.1), and therefore we expect the posterior distribution to also be a Gaussian. We will see the details of maximum a posteriori estimation in Chapter 9.

The idea of including prior knowledge about where good parameters lie is widespread in machine learning. An alternative view, which we saw in Section 8.2.3, is the idea of regularization, which introduces an additional term that biases the resulting parameters to be close to the origin. Maximum a posteriori estimation can be considered to bridge the non-probabilistic and probabilistic worlds as it explicitly acknowledges the need for a prior distribution but it still only produces a point estimate of the parameters.

Remark. The maximum likelihood estimate $\boldsymbol{\theta}_{\text{ML}}$ possesses the following properties (Lehmann and Casella, 1998; Efron and Hastie, 2016):

- Asymptotic consistency: The MLE converges to the true value in the

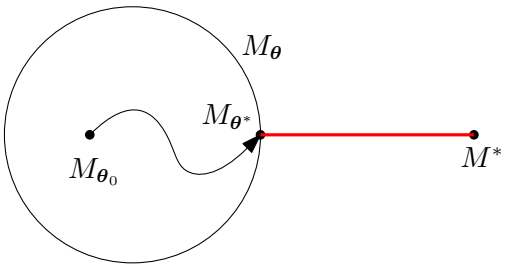


Figure 8.7 Model fitting. In a parametrized class $M_{\boldsymbol{\theta}}$ of models, we optimize the model parameters $\boldsymbol{\theta}$ to minimize the distance to the true (unknown) model M^*.

limit of infinitely many observations, plus a random error that is approximately normal.
- The size of the samples necessary to achieve these properties can be quite large.
- The error's variance decays in $1/N$, where N is the number of data points.
- Especially, in the "small" data regime, maximum likelihood estimation can lead to *overfitting*.

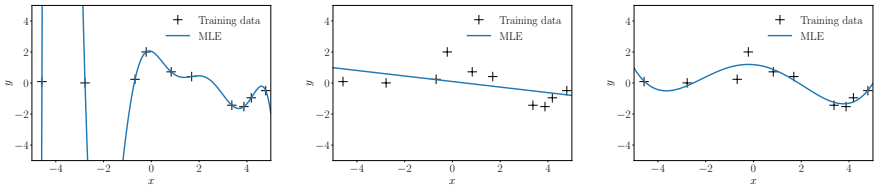
♢

The principle of maximum likelihood estimation (and maximum a posteriori estimation) uses probabilistic modeling to reason about the uncertainty in the data and model parameters. However, we have not yet taken probabilistic modeling to its full extent. In this section, the resulting training procedure still produces a point estimate of the predictor, i.e., training returns one single set of parameter values that represent the best predictor. In Section 8.4, we will take the view that the parameter values should also be treated as random variables, and instead of estimating "best" values of that distribution, we will use the full parameter distribution when making predictions.

8.3.3 Model Fitting

Consider the setting where we are given a dataset, and we are interested in fitting a parametrized model to the data. When we talk about "fitting", we typically mean optimizing/learning model parameters so that they minimize some loss function, e.g., the negative log-likelihood. With maximum likelihood (Section 8.3.1) and maximum a posteriori estimation (Section 8.3.2), we already discussed two commonly used algorithms for model fitting.

The parametrization of the model defines a model class $M_{\boldsymbol{\theta}}$ with which we can operate. For example, in a linear regression setting, we may define the relationship between inputs x and (noise-free) observations y to be $y = ax + b$, where $\boldsymbol{\theta} := \{a, b\}$ are the model parameters. In this case, the model parameters $\boldsymbol{\theta}$ describe the family of affine functions, i.e., straight lines with slope a, which are offset from 0 by b. Assume the data comes



(a) Overfitting (b) Underfitting. (c) Fitting well.

Figure 8.8 Fitting (by maximum likelihood) of different model classes to a regression dataset.

from a model M^*, which is unknown to us. For a given training dataset, we optimize $\boldsymbol{\theta}$ so that $M_{\boldsymbol{\theta}}$ is as close as possible to M^*, where the "closeness" is defined by the objective function we optimize (e.g., squared loss on the training data). Figure 8.7 illustrates a setting where we have a small model class (indicated by the circle $M_{\boldsymbol{\theta}}$), and the data generation model M^* lies outside the set of considered models. We begin our parameter search at $M_{\boldsymbol{\theta}_0}$. After the optimization, i.e., when we obtain the best possible parameters $\boldsymbol{\theta}^*$, we distinguish three different cases: (i) overfitting, (ii) underfitting, and (iii) fitting well. We will give a high-level intuition of what these three concepts mean.

overfitting

Roughly speaking, *overfitting* refers to the situation where the parametrized model class is too rich to model the dataset generated by M^*, i.e., $M_{\boldsymbol{\theta}}$ could model much more complicated datasets. For instance, if the dataset was generated by a linear function, and we define $M_{\boldsymbol{\theta}}$ to be the class of seventh-order polynomials, we could model not only linear functions, but also polynomials of degree two, three, etc. Models that overfit typically have a large number of parameters. An observation we often make is that the overly flexible model class $M_{\boldsymbol{\theta}}$ uses all its modeling power to reduce the training error. If the training data is noisy, it will therefore find some useful signal in the noise itself. This will cause enormous problems when we predict away from the training data. Figure 8.8(a) gives an example of overfitting in the context of regression where the model parameters are learned by means of maximum likelihood (see Section 8.3.1). We will discuss overfitting in regression more in Section 9.2.2.

One way to detect overfitting in practice is to observe that the model has low training risk but high test risk during cross validation (Section 8.2.4).

underfitting

When we run into *underfitting*, we encounter the opposite problem where the model class $M_{\boldsymbol{\theta}}$ is not rich enough. For example, if our dataset was generated by a sinusoidal function, but $\boldsymbol{\theta}$ only parametrizes straight lines, the best optimization procedure will not get us close to the true model. However, we still optimize the parameters and find the best straight line that models the dataset. Figure 8.8(b) shows an example of a model that underfits because it is insufficiently flexible. Models that underfit typically have few parameters.

The third case is when the parametrized model class is about right. Then, our model fits well, i.e., it neither overfits nor underfits. This means our model class is just rich enough to describe the dataset we are given. Figure 8.8(c) shows a model that fits the given dataset fairly well. Ideally,

this is the model class we would want to work with since it has good generalization properties.

In practice, we often define very rich model classes $M_{\boldsymbol{\theta}}$ with many parameters, such as deep neural networks. To mitigate the problem of overfitting, we can use regularization (Section 8.2.3) or priors (Section 8.3.2). We will discuss how to choose the model class in Section 8.6.

8.3.4 Further Reading

When considering probabilistic models, the principle of maximum likelihood estimation generalizes the idea of least-squares regression for linear models, which we will discuss in detail in Chapter 9. When restricting the predictor to have linear form with an additional nonlinear function φ applied to the output, i.e.,

$$p(y_n|\boldsymbol{x}_n, \boldsymbol{\theta}) = \varphi(\boldsymbol{\theta}^\top \boldsymbol{x}_n)\,, \tag{8.21}$$

we can consider other models for other prediction tasks, such as binary classification or modeling count data (McCullagh and Nelder, 1989). An alternative view of this is to consider likelihoods that are from the exponential family (Section 6.6). The class of models, which have linear dependence between parameters and data, and have potentially nonlinear transformation φ (called a *link function*), is referred to as *generalized linear models* (Agresti, 2002, chapter 4).

link function
generalized linear model

Maximum likelihood estimation has a rich history, and was originally proposed by Sir Ronald Fisher in the 1930s. We will expand upon the idea of a probabilistic model in Section 8.4. One debate among researchers who use probabilistic models, is the discussion between Bayesian and frequentist statistics. As mentioned in Section 6.1.1, it boils down to the definition of probability. Recall from Section 6.1 that one can consider probability to be a generalization (by allowing uncertainty) of logical reasoning (Cheeseman, 1985; Jaynes, 2003). The method of maximum likelihood estimation is frequentist in nature, and the interested reader is pointed to Efron and Hastie (2016) for a balanced view of both Bayesian and frequentist statistics.

There are some probabilistic models where maximum likelihood estimation may not be possible. The reader is referred to more advanced statistical textbooks, e.g., Casella and Berger (2002), for approaches, such as method of moments, M-estimation, and estimating equations.

8.4 Probabilistic Modeling and Inference

In machine learning, we are frequently concerned with the interpretation and analysis of data, e.g., for prediction of future events and decision making. To make this task more tractable, we often build models that describe the *generative process* that generates the observed data.

generative process

For example, we can describe the outcome of a coin-flip experiment ("heads" or "tails") in two steps. First, we define a parameter μ, which describes the probability of "heads" as the parameter of a Bernoulli distribution (Chapter 6); second, we can sample an outcome $x \in \{\text{head}, \text{tail}\}$ from the Bernoulli distribution $p(x \mid \mu) = \text{Ber}(\mu)$. The parameter μ gives rise to a specific dataset $\mathcal{X}$ and depends on the coin used. Since μ is unknown in advance and can never be observed directly, we need mechanisms to learn something about μ given observed outcomes of coin-flip experiments. In the following, we will discuss how probabilistic modeling can be used for this purpose.

8.4.1 Probabilistic Models

A probabilistic model is specified by the joint distribution of all random variables.

Probabilistic models represent the uncertain aspects of an experiment as probability distributions. The benefit of using probabilistic models is that they offer a unified and consistent set of tools from probability theory (Chapter 6) for modeling, inference, prediction, and model selection.

In probabilistic modeling, the joint distribution $p(\boldsymbol{x}, \boldsymbol{\theta})$ of the observed variables $\boldsymbol{x}$ and the hidden parameters $\boldsymbol{\theta}$ is of central importance: It encapsulates information from the following:

- The prior and the likelihood (product rule, Section 6.3).
- The marginal likelihood $p(\boldsymbol{x})$, which will play an important role in model selection (Section 8.6), can be computed by taking the joint distribution and integrating out the parameters (sum rule, Section 6.3).
- The posterior, which can be obtained by dividing the joint by the marginal likelihood.

Only the joint distribution has this property. Therefore, a probabilistic model is specified by the joint distribution of all its random variables.

8.4.2 Bayesian Inference

Parameter estimation can be phrased as an optimization problem.

A key task in machine learning is to take a model and the data to uncover the values of the model's hidden variables $\boldsymbol{\theta}$ given the observed variables $\boldsymbol{x}$. In Section 8.3.1, we already discussed two ways for estimating model parameters $\boldsymbol{\theta}$ using maximum likelihood or maximum a posteriori estimation. In both cases, we obtain a single-best value for $\boldsymbol{\theta}$ so that the key algorithmic problem of parameter estimation is solving an optimization problem. Once these point estimates $\boldsymbol{\theta}^*$ are known, we use them to make predictions. More specifically, the predictive distribution will be $p(\boldsymbol{x} \mid \boldsymbol{\theta}^*)$, where we use $\boldsymbol{\theta}^*$ in the likelihood function.

As discussed in Section 6.3, focusing solely on some statistic of the posterior distribution (such as the parameter $\boldsymbol{\theta}^*$ that maximizes the posterior) leads to loss of information, which can be critical in a system that

uses the prediction $p(\boldsymbol{x} \mid \boldsymbol{\theta}^*)$ to make decisions. These decision-making systems typically have different objective functions than the likelihood, a squared-error loss or a mis-classification error. Therefore, having the full posterior distribution around can be extremely useful and leads to more robust decisions. *Bayesian inference* is about finding this posterior distribution (Gelman et al., 2004). For a dataset $\mathcal{X}$, a parameter prior $p(\boldsymbol{\theta})$, and a likelihood function, the posterior

Bayesian inference is about learning the distribution of random variables.

Bayesian inference

$$p(\boldsymbol{\theta} \mid \mathcal{X}) = \frac{p(\mathcal{X} \mid \boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{X})}, \qquad p(\mathcal{X}) = \int p(\mathcal{X} \mid \boldsymbol{\theta})p(\boldsymbol{\theta})\mathrm{d}\boldsymbol{\theta}, \tag{8.22}$$

is obtained by applying Bayes' theorem. The key idea is to exploit Bayes' theorem to invert the relationship between the parameters $\boldsymbol{\theta}$ and the data $\mathcal{X}$ (given by the likelihood) to obtain the posterior distribution $p(\boldsymbol{\theta} \mid \mathcal{X})$.

Bayesian inference inverts the relationship between parameters and the data.

The implication of having a posterior distribution on the parameters is that it can be used to propagate uncertainty from the parameters to the data. More specifically, with a distribution $p(\boldsymbol{\theta})$ on the parameters our predictions will be

$$p(\boldsymbol{x}) = \int p(\boldsymbol{x} \mid \boldsymbol{\theta})p(\boldsymbol{\theta})\mathrm{d}\boldsymbol{\theta} = \mathbb{E}_{\boldsymbol{\theta}}[p(\boldsymbol{x} \mid \boldsymbol{\theta})], \tag{8.23}$$

and they no longer depend on the model parameters $\boldsymbol{\theta}$, which have been marginalized/integrated out. Equation (8.23) reveals that the prediction is an average over all plausible parameter values $\boldsymbol{\theta}$, where the plausibility is encapsulated by the parameter distribution $p(\boldsymbol{\theta})$.

Having discussed parameter estimation in Section 8.3 and Bayesian inference here, let us compare these two approaches to learning. Parameter estimation via maximum likelihood or MAP estimation yields a consistent point estimate $\boldsymbol{\theta}^*$ of the parameters, and the key computational problem to be solved is optimization. In contrast, Bayesian inference yields a (posterior) distribution, and the key computational problem to be solved is integration. Predictions with point estimates are straightforward, whereas predictions in the Bayesian framework require solving another integration problem; see (8.23). However, Bayesian inference gives us a principled way to incorporate prior knowledge, account for side information, and incorporate structural knowledge, all of which is not easily done in the context of parameter estimation. Moreover, the propagation of parameter uncertainty to the prediction can be valuable in decision-making systems for risk assessment and exploration in the context of data-efficient learning (Deisenroth et al., 2015; Kamthe and Deisenroth, 2018).

While Bayesian inference is a mathematically principled framework for learning about parameters and making predictions, there are some practical challenges that come with it because of the integration problems we need to solve; see (8.22) and (8.23). More specifically, if we do not choose a conjugate prior on the parameters (Section 6.6.1), the integrals in (8.22) and (8.23) are not analytically tractable, and we cannot compute the pos-

terior, the predictions, or the marginal likelihood in closed form. In these cases, we need to resort to approximations. Here, we can use stochastic approximations, such as Markov chain Monte Carlo (MCMC) (Gilks et al., 1996), or deterministic approximations, such as the Laplace approximation (Bishop, 2006; Barber, 2012; Murphy, 2012), variational inference (Jordan et al., 1999; Blei et al., 2017), or expectation propagation (Minka, 2001a).

Despite these challenges, Bayesian inference has been successfully applied to a variety of problems, including large-scale topic modeling (Hoffman et al., 2013), click-through-rate prediction (Graepel et al., 2010), data-efficient reinforcement learning in control systems (Deisenroth et al., 2015), online ranking systems (Herbrich et al., 2007), and large-scale recommender systems. There are generic tools, such as Bayesian optimization (Brochu et al., 2009; Snoek et al., 2012; Shahriari et al., 2016), that are very useful ingredients for an efficient search of meta parameters of models or algorithms.

Remark. In the machine learning literature, there can be a somewhat arbitrary separation between (random) "variables" and "parameters". While parameters are estimated (e.g., via maximum likelihood), variables are usually marginalized out. In this book, we are not so strict with this separation because, in principle, we can place a prior on any parameter and integrate it out, which would then turn the parameter into a random variable according to the aforementioned separation. ♢

8.4.3 Latent-Variable Models

latent variable

In practice, it is sometimes useful to have additional *latent variables* $\boldsymbol{z}$ (besides the model parameters $\boldsymbol{\theta}$) as part of the model (Moustaki et al., 2015). These latent variables are different from the model parameters $\boldsymbol{\theta}$ as they do not parametrize the model explicitly. Latent variables may describe the data-generating process, thereby contributing to the interpretability of the model. They also often simplify the structure of the model and allow us to define simpler and richer model structures. Simplification of the model structure often goes hand in hand with a smaller number of model parameters (Paquet, 2008; Murphy, 2012). Learning in latent-variable models (at least via maximum likelihood) can be done in a principled way using the expectation maximization (EM) algorithm (Dempster et al., 1977; Bishop, 2006). Examples, where such latent variables are helpful, are principal component analysis for dimensionality reduction (Chapter 10), Gaussian mixture models for density estimation (Chapter 11), hidden Markov models (Maybeck, 1979) or dynamical systems (Ghahramani and Roweis, 1999; Ljung, 1999) for time-series modeling, and meta learning and task generalization (Hausman et al., 2018; Sæmundsson et al., 2018). Although the introduction of these latent variables

may make the model structure and the generative process easier, learning in latent-variable models is generally hard, as we will see in Chapter 11.

Since latent-variable models also allow us to define the process that generates data from parameters, let us have a look at this generative process. Denoting data by $\boldsymbol{x}$, the model parameters by $\boldsymbol{\theta}$ and the latent variables by $\boldsymbol{z}$, we obtain the conditional distribution

$$p(\boldsymbol{x} \mid \boldsymbol{z}, \boldsymbol{\theta}) \tag{8.24}$$

that allows us to generate data for any model parameters and latent variables. Given that $\boldsymbol{z}$ are latent variables, we place a prior $p(\boldsymbol{z})$ on them.

As the models we discussed previously, models with latent variables can be used for parameter learning and inference within the frameworks we discussed in Sections 8.3 and 8.4.2. To facilitate learning (e.g., by means of maximum likelihood estimation or Bayesian inference), we follow a two-step procedure. First, we compute the likelihood $p(\boldsymbol{x} \mid \boldsymbol{\theta})$ of the model, which does not depend on the latent variables. Second, we use this likelihood for parameter estimation or Bayesian inference, where we use exactly the same expressions as in Sections 8.3 and 8.4.2, respectively.

Since the likelihood function $p(\boldsymbol{x} \mid \boldsymbol{\theta})$ is the predictive distribution of the data given the model parameters, we need to marginalize out the latent variables so that

$$p(\boldsymbol{x} \mid \boldsymbol{\theta}) = \int p(\boldsymbol{x} \mid \boldsymbol{z}, \boldsymbol{\theta}) p(\boldsymbol{z}) \mathrm{d}\boldsymbol{z}\,, \tag{8.25}$$

where $p(\boldsymbol{x} \mid \boldsymbol{z}, \boldsymbol{\theta})$ is given in (8.24) and $p(\boldsymbol{z})$ is the prior on the latent variables. Note that the likelihood must not depend on the latent variables $\boldsymbol{z}$, but it is only a function of the data $\boldsymbol{x}$ and the model parameters $\boldsymbol{\theta}$.

The likelihood is a function of the data and the model parameters, but is independent of the latent variables.

The likelihood in (8.25) directly allows for parameter estimation via maximum likelihood. MAP estimation is also straightforward with an additional prior on the model parameters $\boldsymbol{\theta}$ as discussed in Section 8.3.2. Moreover, with the likelihood (8.25) Bayesian inference (Section 8.4.2) in a latent-variable model works in the usual way: We place a prior $p(\boldsymbol{\theta})$ on the model parameters and use Bayes' theorem to obtain a posterior distribution

$$p(\boldsymbol{\theta} \mid \mathcal{X}) = \frac{p(\mathcal{X} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta})}{p(\mathcal{X})} \tag{8.26}$$

over the model parameters given a dataset $\mathcal{X}$. The posterior in (8.26) can be used for predictions within a Bayesian inference framework; see (8.23).

One challenge we have in this latent-variable model is that the likelihood $p(\mathcal{X} \mid \boldsymbol{\theta})$ requires the marginalization of the latent variables according to (8.25). Except when we choose a conjugate prior $p(\boldsymbol{z})$ for $p(\boldsymbol{x} \mid \boldsymbol{z}, \boldsymbol{\theta})$, the marginalization in (8.25) is not analytically tractable, and we need to resort to approximations (Bishop, 2006; Paquet, 2008; Murphy, 2012; Moustaki et al., 2015).

Similar to the parameter posterior (8.26) we can compute a posterior on the latent variables according to

$$p(\boldsymbol{z} \mid \mathcal{X}) = \frac{p(\mathcal{X} \mid \boldsymbol{z})p(\boldsymbol{z})}{p(\mathcal{X})}, \qquad p(\mathcal{X} \mid \boldsymbol{z}) = \int p(\mathcal{X} \mid \boldsymbol{z}, \boldsymbol{\theta})p(\boldsymbol{\theta})\mathrm{d}\boldsymbol{\theta}, \quad (8.27)$$

where $p(\boldsymbol{z})$ is the prior on the latent variables and $p(\mathcal{X} \mid \boldsymbol{z})$ requires us to integrate out the model parameters $\boldsymbol{\theta}$.

Given the difficulty of solving integrals analytically, it is clear that marginalizing out both the latent variables and the model parameters at the same time is not possible in general (Bishop, 2006; Murphy, 2012). A quantity that is easier to compute is the posterior distribution on the latent variables, but conditioned on the model parameters, i.e.,

$$p(\boldsymbol{z} \mid \mathcal{X}, \boldsymbol{\theta}) = \frac{p(\mathcal{X} \mid \boldsymbol{z}, \boldsymbol{\theta})p(\boldsymbol{z})}{p(\mathcal{X} \mid \boldsymbol{\theta})}, \quad (8.28)$$

where $p(\boldsymbol{z})$ is the prior on the latent variables and $p(\mathcal{X} \mid \boldsymbol{z}, \boldsymbol{\theta})$ is given in (8.24).

In Chapters 10 and 11, we derive the likelihood functions for PCA and Gaussian mixture models, respectively. Moreover, we compute the posterior distributions (8.28) on the latent variables for both PCA and Gaussian mixture models.

Remark. In the following chapters, we may not be drawing such a clear distinction between latent variables $\boldsymbol{z}$ and uncertain model parameters $\boldsymbol{\theta}$ and call the model parameters "latent" or "hidden" as well because they are unobserved. In Chapters 10 and 11, where we use the latent variables $\boldsymbol{z}$, we will pay attention to the difference as we will have two different types of hidden variables: model parameters $\boldsymbol{\theta}$ and latent variables $\boldsymbol{z}$. $\diamondsuit$

We can exploit the fact that all the elements of a probabilistic model are random variables to define a unified language for representing them. In Section 8.5, we will see a concise graphical language for representing the structure of probabilistic models. We will use this graphical language to describe the probabilistic models in the subsequent chapters.

8.4.4 Further Reading

Probabilistic models in machine learning (Bishop, 2006; Barber, 2012; Murphy, 2012) provide a way for users to capture uncertainty about data and predictive models in a principled fashion. Ghahramani (2015) presents a short review of probabilistic models in machine learning. Given a probabilistic model, we may be lucky enough to be able to compute parameters of interest analytically. However, in general, analytic solutions are rare, and computational methods such as sampling (Gilks et al., 1996; Brooks et al., 2011) and variational inference (Jordan et al., 1999; Blei et al.,

2017) are used. Moustaki et al. (2015) and Paquet (2008) provide a good overview of Bayesian inference in latent-variable models.

In recent years, several programming languages have been proposed that aim to treat the variables defined in software as random variables corresponding to probability distributions. The objective is to be able to write complex functions of probability distributions, while under the hood the compiler automatically takes care of the rules of Bayesian inference. This rapidly changing field is called *probabilistic programming*.

probabilistic programming

8.5 Directed Graphical Models

In this section, we introduce a graphical language for specifying a probabilistic model, called the *directed graphical model*. It provides a compact and succinct way to specify probabilistic models, and allows the reader to visually parse dependencies between random variables. A graphical model visually captures the way in which the joint distribution over all random variables can be decomposed into a product of factors depending only on a subset of these variables. In Section 8.4, we identified the joint distribution of a probabilistic model as the key quantity of interest because it comprises information about the prior, the likelihood, and the posterior. However, the joint distribution by itself can be quite complicated, and it does not tell us anything about structural properties of the probabilistic model. For example, the joint distribution $p(a, b, c)$ does not tell us anything about independence relations. This is the point where graphical models come into play. This section relies on the concepts of independence and conditional independence, as described in Section 6.4.5.

directed graphical model

Directed graphical models are also known as Bayesian networks.

graphical model

In a *graphical model*, nodes are random variables. In Figure 8.9(a), the nodes represent the random variables a, b, c. Edges represent probabilistic relations between variables, e.g., conditional probabilities.

Remark. Not every distribution can be represented in a particular choice of graphical model. A discussion of this can be found in Bishop (2006). ♢

Probabilistic graphical models have some convenient properties:

- They are a simple way to visualize the structure of a probabilistic model.
- They can be used to design or motivate new kinds of statistical models.
- Inspection of the graph alone gives us insight into properties, e.g., conditional independence.
- Complex computations for inference and learning in statistical models can be expressed in terms of graphical manipulations.

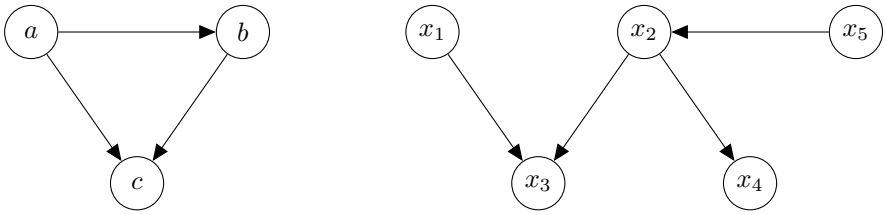
8.5.1 Graph Semantics

directed graphical model/Bayesian network

Directed graphical models/Bayesian networks are a method for representing conditional dependencies in a probabilistic model. They provide a visual

description of the conditional probabilities, hence, providing a simple language for describing complex interdependence. The modular description also entails computational simplification. Directed links (arrows) between two nodes (random variables) indicate conditional probabilities. For example, the arrow between a and b in Figure 8.9(a) gives the conditional probability $p(b \mid a)$ of b given a.

With additional assumptions, the arrows can be used to indicate causal relationships (Pearl, 2009).



(a) Fully connected. (b) Not fully connected.

Figure 8.9 Examples of directed graphical models.

Directed graphical models can be derived from joint distributions if we know something about their factorization.

Example 8.7
Consider the joint distribution

$$p(a, b, c) = p(c \mid a, b)p(b \mid a)p(a) \tag{8.29}$$

of three random variables a, b, c. The factorization of the joint distribution in (8.29) tells us something about the relationship between the random variables:

- c depends directly on a and b.
- b depends directly on a.
- a depends neither on b nor on c.

For the factorization in (8.29), we obtain the directed graphical model in Figure 8.9(a).

In general, we can construct the corresponding directed graphical model from a factorized joint distribution as follows:

1. Create a node for all random variables.
2. For each conditional distribution, we add a directed link (arrow) to the graph from the nodes corresponding to the variables on which the distribution is conditioned.

The graph layout depends on the factorization of the joint distribution.

The graph layout depends on the choice of factorization of the joint distribution.

We discussed how to get from a known factorization of the joint distribution to the corresponding directed graphical model. Now, we will do

exactly the opposite and describe how to extract the joint distribution of a set of random variables from a given graphical model.

Example 8.8
Looking at the graphical model in Figure 8.9(b), we exploit two properties:

- The joint distribution $p(x_1, \ldots, x_5)$ we seek is the product of a set of conditionals, one for each node in the graph. In this particular example, we will need five conditionals.
- Each conditional depends only on the parents of the corresponding node in the graph. For example, x_4 will be conditioned on x_2.

These two properties yield the desired factorization of the joint distribution

$$p(x_1, x_2, x_3, x_4, x_5) = p(x_1)p(x_5)p(x_2 \mid x_5)p(x_3 \mid x_1, x_2)p(x_4 \mid x_2)\,. \quad (8.30)$$

In general, the joint distribution $p(\boldsymbol{x}) = p(x_1, \ldots, x_K)$ is given as

$$p(\boldsymbol{x}) = \prod_{k=1}^{K} p(x_k \mid \mathrm{Pa}_k)\,, \quad (8.31)$$

where Pa_k means "the parent nodes of x_k". Parent nodes of x_k are nodes that have arrows pointing to x_k.

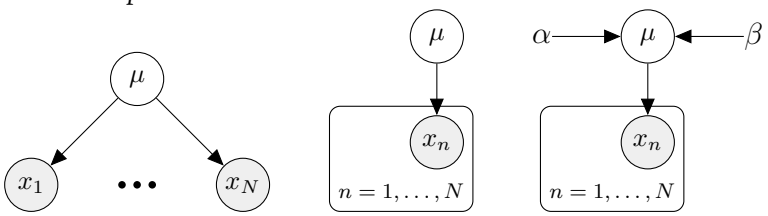
We conclude this subsection with a concrete example of the coin-flip experiment. Consider a Bernoulli experiment (Example 6.8) where the probability that the outcome x of this experiment is "heads" is

$$p(x \mid \mu) = \mathrm{Ber}(\mu)\,. \quad (8.32)$$

We now repeat this experiment N times and observe outcomes $x_1, \ldots, x_N$ so that we obtain the joint distribution

$$p(x_1, \ldots, x_N \mid \mu) = \prod_{n=1}^{N} p(x_n \mid \mu)\,. \quad (8.33)$$

The expression on the right-hand side is a product of Bernoulli distributions on each individual outcome because the experiments are independent. Recall from Section 6.4.5 that statistical independence means that the distribution factorizes. To write the graphical model down for this setting, we make the distinction between unobserved/latent variables and observed variables. Graphically, observed variables are denoted by shaded nodes so that we obtain the graphical model in Figure 8.10(a). We see that the single parameter μ is the same for all x_n, $n = 1, \ldots, N$ as the outcomes x_n are identically distributed. A more compact, but equivalent, graphical model for this setting is given in Figure 8.10(b), where we use



(a) Version with x_n explicit. (b) Version with plate notation. (c) Hyperparameters α and β on the latent μ.

Figure 8.10 Graphical models for a repeated Bernoulli experiment.

the *plate* notation. The plate (box) repeats everything inside (in this case, the observations x_n) N times. Therefore, both graphical models are equivalent, but the plate notation is more compact. Graphical models immediately allow us to place a hyperprior on μ. A *hyperprior* is a second layer of prior distributions on the parameters of the first layer of priors. Figure 8.10(c) places a Beta(α, β) prior on the latent variable μ. If we treat α and β as deterministic parameters, i.e., not random variables, we omit the circle around it.

plate

hyperprior

8.5.2 Conditional Independence and d-Separation

Directed graphical models allow us to find conditional independence (Section 6.4.5) relationship properties of the joint distribution only by looking at the graph. A concept called *d-separation* (Pearl, 1988) is key to this.

d-separation

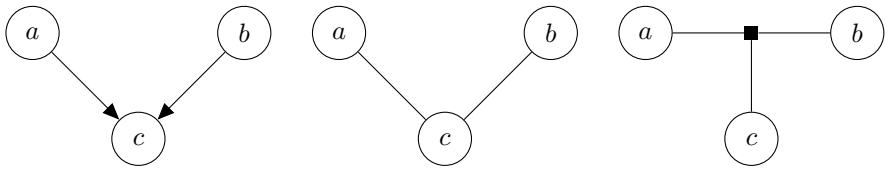
Consider a general directed graph in which $\mathcal{A}, \mathcal{B}, \mathcal{C}$ are arbitrary nonintersecting sets of nodes (whose union may be smaller than the complete set of nodes in the graph). We wish to ascertain whether a particular conditional independence statement, "$\mathcal{A}$ is conditionally independent of $\mathcal{B}$ given $\mathcal{C}$", denoted by

$$\mathcal{A} \perp\!\!\!\perp \mathcal{B} \,|\, \mathcal{C}\,, \tag{8.34}$$

is implied by a given directed acyclic graph. To do so, we consider all possible trails (paths that ignore the direction of the arrows) from any node in $\mathcal{A}$ to any nodes in $\mathcal{B}$. Any such path is said to be blocked if it includes any node such that either of the following are true:

- The arrows on the path meet either head to tail or tail to tail at the node, and the node is in the set $\mathcal{C}$.
- The arrows meet head to head at the node, and neither the node nor any of its descendants is in the set $\mathcal{C}$.

If all paths are blocked, then $\mathcal{A}$ is said to be *d-separated* from $\mathcal{B}$ by $\mathcal{C}$, and the joint distribution over all of the variables in the graph will satisfy $\mathcal{A} \perp\!\!\!\perp \mathcal{B} \,|\, \mathcal{C}$.



(a) Directed graphical model (b) Undirected graphical model (c) Factor graph

Figure 8.12 Three types of graphical models: (a) Directed graphical models (Bayesian networks); (b) Undirected graphical models (Markov random fields); (c) Factor graphs.

Example 8.9 (Conditional Independence)

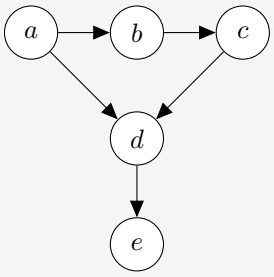


Figure 8.11 D-separation example.

Consider the graphical model in Figure 8.11. Visual inspection gives us

$$b \perp\!\!\!\perp d \,|\, a, c \tag{8.35}$$
$$a \perp\!\!\!\perp c \,|\, b \tag{8.36}$$
$$b \not\perp\!\!\!\perp d \,|\, c \tag{8.37}$$
$$a \not\perp\!\!\!\perp c \,|\, b, e \tag{8.38}$$

Directed graphical models allow a compact representation of probabilistic models, and we will see examples of directed graphical models in Chapters 9, 10, and 11. The representation, along with the concept of conditional independence, allows us to factorize the respective probabilistic models into expressions that are easier to optimize.

The graphical representation of the probabilistic model allows us to visually see the impact of design choices we have made on the structure of the model. We often need to make high-level assumptions about the structure of the model. These modeling assumptions (hyperparameters) affect the prediction performance, but cannot be selected directly using the approaches we have seen so far. We will discuss different ways to choose the structure in Section 8.6.

8.5.3 Further Reading

An introduction to probabilistic graphical models can be found in Bishop (2006, chapter 8), and an extensive description of the different applications and corresponding algorithmic implications can be found in the book by Koller and Friedman (2009). There are three main types of probabilistic graphical models:

directed graphical model
Bayesian network
undirected graphical model
Markov random field
factor graph

- *Directed graphical models* (*Bayesian networks*); see Figure 8.12(a)
- *Undirected graphical models* (*Markov random fields*); see Figure 8.12(b)
- *Factor graphs*; see Figure 8.12(c)

Graphical models allow for graph-based algorithms for inference and learning, e.g., via local message passing. Applications range from ranking in online games (Herbrich et al., 2007) and computer vision (e.g., image segmentation, semantic labeling, image denoising, image restoration (Kittler and Föglein, 1984; Sucar and Gillies, 1994; Shotton et al., 2006; Szeliski et al., 2008)) to coding theory (McEliece et al., 1998), solving linear equation systems (Shental et al., 2008), and iterative Bayesian state estimation in signal processing (Bickson et al., 2007; Deisenroth and Mohamed, 2012).

One topic that is particularly important in real applications that we do not discuss in this book is the idea of structured prediction (Bakir et al., 2007; Nowozin et al., 2014), which allows machine learning models to tackle predictions that are structured, for example sequences, trees, and graphs. The popularity of neural network models has allowed more flexible probabilistic models to be used, resulting in many useful applications of structured models (Goodfellow et al., 2016, chapter 16). In recent years, there has been a renewed interest in graphical models due to their applications to causal inference (Pearl, 2009; Imbens and Rubin, 2015; Peters et al., 2017; Rosenbaum, 2017).

8.6 Model Selection

In machine learning, we often need to make high-level modeling decisions that critically influence the performance of the model. The choices we make (e.g., the functional form of the likelihood) influence the number and type of free parameters in the model and thereby also the flexibility and expressivity of the model. More complex models are more flexible in the sense that they can be used to describe more datasets. For instance, a polynomial of degree 1 (a line $y = a_0 + a_1 x$) can only be used to describe linear relations between inputs x and observations y. A polynomial of degree 2 can additionally describe quadratic relationships between inputs and observations.

A polynomial $y = a_0 + a_1 x + a_2 x^2$ can also describe linear functions by setting $a_2 = 0$, i.e., it is strictly more expressive than a first-order polynomial.

One would now think that very flexible models are generally preferable to simple models because they are more expressive. A general problem

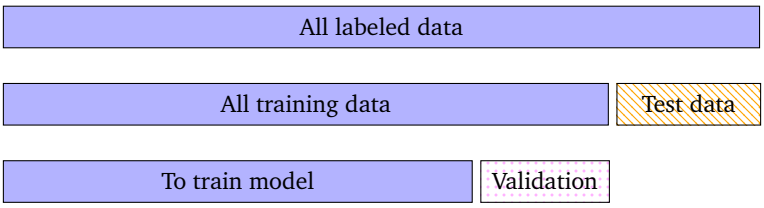


Figure 8.13 Nested cross-validation. We perform two levels of K-fold cross-validation.

is that at training time we can only use the training set to evaluate the performance of the model and learn its parameters. However, the performance on the training set is not really what we are interested in. In Section 8.3, we have seen that maximum likelihood estimation can lead to overfitting, especially when the training dataset is small. Ideally, our model (also) works well on the test set (which is not available at training time). Therefore, we need some mechanisms for assessing how a model *generalizes* to unseen test data. *Model selection* is concerned with exactly this problem.

8.6.1 Nested Cross-Validation

We have already seen an approach (cross-validation in Section 8.2.4) that can be used for model selection. Recall that cross-validation provides an estimate of the generalization error by repeatedly splitting the dataset into training and validation sets. We can apply this idea one more time, i.e., for each split, we can perform another round of cross-validation. This is sometimes referred to as *nested cross-validation*; see Figure 8.13. The inner level is used to estimate the performance of a particular choice of model or hyperparameter on a internal validation set. The outer level is used to estimate generalization performance for the best choice of model chosen by the inner loop. We can test different model and hyperparameter choices in the inner loop. To distinguish the two levels, the set used to estimate the generalization performance is often called the *test set* and the set used for choosing the best model is called the *validation set*. The inner loop estimates the expected value of the generalization error for a given model (8.39), by approximating it using the empirical error on the validation set, i.e.,

nested cross-validation

test set

validation set

$$\mathbb{E}_{\mathcal{V}}[\mathbf{R}(\mathcal{V} \mid M)] \approx \frac{1}{K} \sum_{k=1}^{K} \mathbf{R}(\mathcal{V}^{(k)} \mid M) \,, \tag{8.39}$$

where $\mathbf{R}(\mathcal{V} \mid M)$ is the empirical risk (e.g., root mean square error) on the validation set $\mathcal{V}$ for model M. We repeat this procedure for all models and choose the model that performs best. Note that cross-validation not only gives us the expected generalization error, but we can also obtain high-order statistics, e.g., the standard error, an estimate of how uncertain the

The standard error is defined as $\frac{\sigma}{\sqrt{K}}$, where K is the number of experiments and σ is the standard deviation of the risk of each experiment.

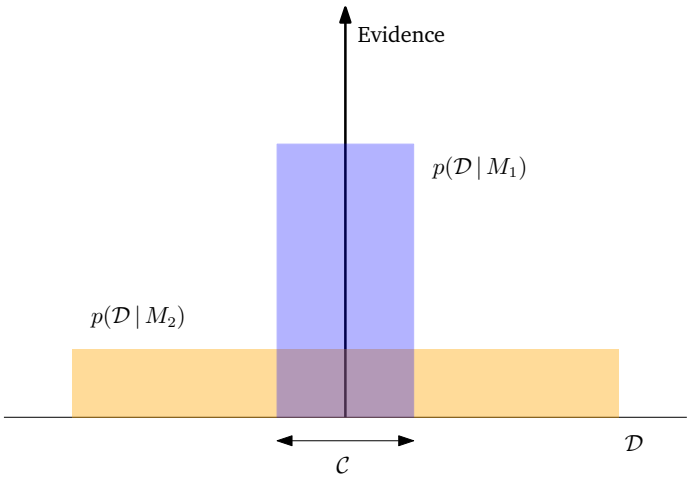


Figure 8.14 Bayesian inference embodies Occam's razor. The horizontal axis describes the space of all possible datasets $\mathcal{D}$. The evidence (vertical axis) evaluates how well a model predicts available data. Since $p(\mathcal{D} \mid M_i)$ needs to integrate to 1, we should choose the model with the greatest evidence. Adapted from MacKay (2003).

mean estimate is. Once the model is chosen, we can evaluate the final performance on the test set.

8.6.2 Bayesian Model Selection

There are many approaches to model selection, some of which are covered in this section. Generally, they all attempt to trade off model complexity and data fit. We assume that simpler models are less prone to overfitting than complex models, and hence the objective of model selection is to find the simplest model that explains the data reasonably well. This concept is also known as *Occam's razor*.

Occam's razor

Remark. If we treat model selection as a hypothesis testing problem, we are looking for the simplest hypothesis that is consistent with the data (Murphy, 2012). ♢

One may consider placing a prior on models that favors simpler models. However, it is not necessary to do this: An "automatic Occam's Razor" is quantitatively embodied in the application of Bayesian probability (Smith and Spiegelhalter, 1980; Jefferys and Berger, 1992; MacKay, 1992). Figure 8.14, adapted from MacKay (2003), gives us the basic intuition why complex and very expressive models may turn out to be a less probable choice for modeling a given dataset $\mathcal{D}$. Let us think of the horizontal axis representing the space of all possible datasets $\mathcal{D}$. If we are interested in the posterior probability $p(M_i \mid \mathcal{D})$ of model M_i given the data $\mathcal{D}$, we can employ Bayes' theorem. Assuming a uniform prior $p(M)$ over all models, Bayes' theorem rewards models in proportion to how much they predicted the data that occurred. This prediction of the data given model M_i, $p(\mathcal{D} \mid M_i)$, is called the *evidence* for M_i. A simple model M_1 can only predict a small number of datasets, which is shown by $p(\mathcal{D} \mid M_1)$; a more powerful model M_2 that has, e.g., more free parameters than M_1, is able

These predictions are quantified by a normalized probability distribution on $\mathcal{D}$, i.e., it needs to integrate/sum to 1.

evidence

to predict a greater variety of datasets. This means, however, that M_2 does not predict the datasets in region C as well as M_1. Suppose that equal prior probabilities have been assigned to the two models. Then, if the dataset falls into region C, the less powerful model M_1 is the more probable model.

Earlier in this chapter, we argued that models need to be able to explain the data, i.e., there should be a way to generate data from a given model. Furthermore, if the model has been appropriately learned from the data, then we expect that the generated data should be similar to the empirical data. For this, it is helpful to phrase model selection as a hierarchical inference problem, which allows us to compute the posterior distribution over models.

Bayesian model selection generative process

Let us consider a finite number of models $M = \{M_1, \ldots, M_K\}$, where each model M_k possesses parameters $\boldsymbol{\theta}_k$. In *Bayesian model selection*, we place a prior $p(M)$ on the set of models. The corresponding *generative process* that allows us to generate data from this model is

$$M_k \sim p(M) \tag{8.40}$$

$$\boldsymbol{\theta}_k \sim p(\boldsymbol{\theta} \mid M_k) \tag{8.41}$$

$$\mathcal{D} \sim p(\mathcal{D} \mid \boldsymbol{\theta}_k) \tag{8.42}$$

and illustrated in Figure 8.15. Given a training set $\mathcal{D}$, we apply Bayes' theorem and compute the posterior distribution over models as

$$p(M_k \mid \mathcal{D}) \propto p(M_k)p(\mathcal{D} \mid M_k)\,. \tag{8.43}$$

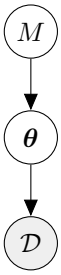
Note that this posterior no longer depends on the model parameters $\boldsymbol{\theta}_k$ because they have been integrated out in the Bayesian setting since

$$p(\mathcal{D} \mid M_k) = \int p(\mathcal{D} \mid \boldsymbol{\theta}_k)p(\boldsymbol{\theta}_k \mid M_k)d\boldsymbol{\theta}_k\,, \tag{8.44}$$

where $p(\boldsymbol{\theta}_k \mid M_k)$ is the prior distribution of the model parameters $\boldsymbol{\theta}_k$ of model M_k. The term (8.44) is referred to as the *model evidence* or *marginal likelihood*. From the posterior in (8.43), we determine the MAP estimate

$$M^* = \arg\max_{M_k} p(M_k \mid \mathcal{D})\,. \tag{8.45}$$

Figure 8.15 Illustration of the hierarchical generative process in Bayesian model selection. We place a prior $p(M)$ on the set of models. For each model, there is a distribution $p(\boldsymbol{\theta} \mid M)$ on the corresponding model parameters, which is used to generate the data $\mathcal{D}$.



model evidence
marginal likelihood

With a uniform prior $p(M_k) = \frac{1}{K}$, which gives every model equal (prior) probability, determining the MAP estimate over models amounts to picking the model that maximizes the model evidence (8.44).

Remark (Likelihood and Marginal Likelihood). There are some important differences between a likelihood and a marginal likelihood (evidence): While the likelihood is prone to overfitting, the marginal likelihood is typically not as the model parameters have been marginalized out (i.e., we no longer have to fit the parameters). Furthermore, the marginal likelihood automatically embodies a trade-off between model complexity and data fit (Occam's razor). ♢

8.6.3 Bayes Factors for Model Comparison

Consider the problem of comparing two probabilistic models M_1, M_2, given a dataset $\mathcal{D}$. If we compute the posteriors $p(M_1 \mid \mathcal{D})$ and $p(M_2 \mid \mathcal{D})$, we can compute the ratio of the posteriors

$$\underbrace{\frac{p(M_1 \mid \mathcal{D})}{p(M_2 \mid \mathcal{D})}}_{\text{posterior odds}} = \frac{\frac{p(\mathcal{D} \mid M_1)p(M_1)}{p(\mathcal{D})}}{\frac{p(\mathcal{D} \mid M_2)p(M_2)}{p(\mathcal{D})}} = \underbrace{\frac{p(M_1)}{p(M_2)}}_{\text{prior odds}} \underbrace{\frac{p(\mathcal{D} \mid M_1)}{p(\mathcal{D} \mid M_2)}}_{\text{Bayes factor}}. \tag{8.46}$$

The ratio of the posteriors is also called the *posterior odds*. The first fraction on the right-hand side of (8.46), the *prior odds*, measures how much our prior (initial) beliefs favor M_1 over M_2. The ratio of the marginal likelihoods (second fraction on the right-hand-side) is called the *Bayes factor* and measures how well the data $\mathcal{D}$ is predicted by M_1 compared to M_2.

posterior odds

prior odds

Bayes factor

Remark. The *Jeffreys-Lindley paradox* states that the "Bayes factor always favors the simpler model since the probability of the data under a complex model with a diffuse prior will be very small" (Murphy, 2012). Here, a diffuse prior refers to a prior that does not favor specific models, i.e., many models are a priori plausible under this prior. ♢

Jeffreys-Lindley paradox

If we choose a uniform prior over models, the prior odds term in (8.46) is 1, i.e., the posterior odds is the ratio of the marginal likelihoods (Bayes factor)

$$\frac{p(\mathcal{D} \mid M_1)}{p(\mathcal{D} \mid M_2)}. \tag{8.47}$$

If the Bayes factor is greater than 1, we choose model M_1, otherwise model M_2. In a similar way to frequentist statistics, there are guidelines on the size of the ratio that one should consider before "significance" of the result (Jeffreys, 1961).

Remark (Computing the Marginal Likelihood). The marginal likelihood plays an important role in model selection: We need to compute Bayes factors (8.46) and posterior distributions over models (8.43).

Unfortunately, computing the marginal likelihood requires us to solve an integral (8.44). This integration is generally analytically intractable, and we will have to resort to approximation techniques, e.g., numerical integration (Stoer and Burlirsch, 2002), stochastic approximations using Monte Carlo (Murphy, 2012), or Bayesian Monte Carlo techniques (O'Hagan, 1991; Rasmussen and Ghahramani, 2003).

However, there are special cases in which we can solve it. In Section 6.6.1, we discussed conjugate models. If we choose a conjugate parameter prior $p(\boldsymbol{\theta})$, we can compute the marginal likelihood in closed form. In Chapter 9, we will do exactly this in the context of linear regression. ♢

We have seen a brief introduction to the basic concepts of machine learning in this chapter. For the rest of this part of the book we will see

how the three different flavors of learning in Sections 8.2, 8.3, and 8.4 are applied to the four pillars of machine learning (regression, dimensionality reduction, density estimation, and classification).

8.6.4 Further Reading

We mentioned at the start of the section that there are high-level modeling choices that influence the performance of the model. Examples include the following:

- The degree of a polynomial in a regression setting
- The number of components in a mixture model
- The network architecture of a (deep) neural network
- The type of kernel in a support vector machine
- The dimensionality of the latent space in PCA
- The learning rate (schedule) in an optimization algorithm

In parametric models, the number of parameters is often related to the complexity of the model class.

Rasmussen and Ghahramani (2001) showed that the automatic Occam's razor does not necessarily penalize the number of parameters in a model, but it is active in terms of the complexity of functions. They also showed that the automatic Occam's razor also holds for Bayesian nonparametric models with many parameters, e.g., Gaussian processes.

If we focus on the maximum likelihood estimate, there exist a number of heuristics for model selection that discourage overfitting. They are called information criteria, and we choose the model with the largest value. The *Akaike information criterion* (AIC) (Akaike, 1974)

Akaike information criterion

$$\log p(\boldsymbol{x} \mid \boldsymbol{\theta}) - M \tag{8.48}$$

corrects for the bias of the maximum likelihood estimator by addition of a penalty term to compensate for the overfitting of more complex models with lots of parameters. Here, M is the number of model parameters. The AIC estimates the relative information lost by a given model.

The *Bayesian information criterion* (BIC) (Schwarz, 1978)

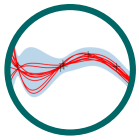
Bayesian information criterion

$$\log p(\boldsymbol{x}) = \log \int p(\boldsymbol{x} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta}) \mathrm{d}\boldsymbol{\theta} \approx \log p(\boldsymbol{x} \mid \boldsymbol{\theta}) - \frac{1}{2} M \log N \tag{8.49}$$

can be used for exponential family distributions. Here, N is the number of data points and M is the number of parameters. BIC penalizes model complexity more heavily than AIC.

9

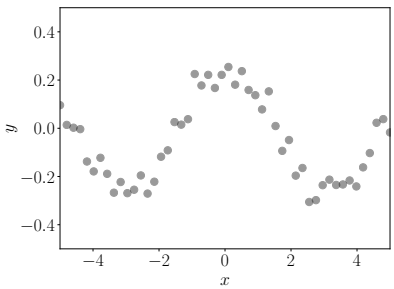
Linear Regression



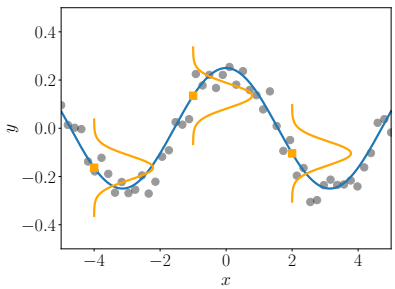
In the following, we will apply the mathematical concepts from Chapters 2, 5, 6, and 7 to solve linear regression (curve fitting) problems. In *regression*, we aim to find a function f that maps inputs $\boldsymbol{x} \in \mathbb{R}^D$ to corresponding function values $f(\boldsymbol{x}) \in \mathbb{R}$. We assume we are given a set of training inputs $\boldsymbol{x}_n$ and corresponding noisy observations $y_n = f(\boldsymbol{x}_n) + \epsilon$, where ϵ is an i.i.d. random variable that describes measurement/observation noise and potentially unmodeled processes (which we will not consider further in this chapter). Throughout this chapter, we assume zero-mean Gaussian noise. Our task is to find a function that not only models the training data, but generalizes well to predicting function values at input locations that are not part of the training data (see Chapter 8). An illustration of such a regression problem is given in Figure 9.1. A typical regression setting is given in Figure 9.1(a): For some input values x_n, we observe (noisy) function values $y_n = f(x_n) + \epsilon$. The task is to infer the function f that generated the data and generalizes well to function values at new input locations. A possible solution is given in Figure 9.1(b), where we also show three distributions centered at the function values $f(x)$ that represent the noise in the data.

regression

Regression is a fundamental problem in machine learning, and regression problems appear in a diverse range of research areas and applica-



(a) Regression problem: observed noisy function values from which we wish to infer the underlying function that generated the data.



(b) Regression solution: possible function that could have generated the data (blue) with indication of the measurement noise of the function value at the corresponding inputs (orange distributions).

Figure 9.1
(a) Dataset;
(b) possible solution to the regression problem.

tions, including time-series analysis (e.g., system identification), control and robotics (e.g., reinforcement learning, forward/inverse model learning), optimization (e.g., line searches, global optimization), and deep-learning applications (e.g., computer games, speech-to-text translation, image recognition, automatic video annotation). Regression is also a key ingredient of classification algorithms. Finding a regression function requires solving a variety of problems, including the following:

Normally, the type of noise could also be a "model choice", but we fix the noise to be Gaussian in this chapter.

- **Choice of the model (type) and the parametrization** of the regression function. Given a dataset, what function classes (e.g., polynomials) are good candidates for modeling the data, and what particular parametrization (e.g., degree of the polynomial) should we choose? Model selection, as discussed in Section 8.6, allows us to compare various models to find the simplest model that explains the training data reasonably well.
- **Finding good parameters.** Having chosen a model of the regression function, how do we find good model parameters? Here, we will need to look at different loss/objective functions (they determine what a "good" fit is) and optimization algorithms that allow us to minimize this loss.
- **Overfitting and model selection.** Overfitting is a problem when the regression function fits the training data "too well" but does not generalize to unseen test data. Overfitting typically occurs if the underlying model (or its parametrization) is overly flexible and expressive; see Section 8.6. We will look at the underlying reasons and discuss ways to mitigate the effect of overfitting in the context of linear regression.
- **Relationship between loss functions and parameter priors.** Loss functions (optimization objectives) are often motivated and induced by probabilistic models. We will look at the connection between loss functions and the underlying prior assumptions that induce these losses.
- **Uncertainty modeling.** In any practical setting, we have access to only a finite, potentially large, amount of (training) data for selecting the model class and the corresponding parameters. Given that this finite amount of training data does not cover all possible scenarios, we may want to describe the remaining parameter uncertainty to obtain a measure of confidence of the model's prediction at test time; the smaller the training set, the more important uncertainty modeling. Consistent modeling of uncertainty equips model predictions with confidence bounds.

In the following, we will be using the mathematical tools from Chapters 3, 5, 6 and 7 to solve linear regression problems. We will discuss maximum likelihood and maximum a posteriori (MAP) estimation to find optimal model parameters. Using these parameter estimates, we will have a brief look at generalization errors and overfitting. Toward the end of this chapter, we will discuss Bayesian linear regression, which allows us to reason about model parameters at a higher level, thereby removing some of the problems encountered in maximum likelihood and MAP estimation.

9.1 Problem Formulation

Because of the presence of observation noise, we will adopt a probabilistic approach and explicitly model the noise using a likelihood function. More specifically, throughout this chapter, we consider a regression problem with the likelihood function

$$p(y \mid \boldsymbol{x}) = \mathcal{N}\big(y \mid f(\boldsymbol{x}),\, \sigma^2\big)\,. \tag{9.1}$$

Here, $\boldsymbol{x} \in \mathbb{R}^D$ are inputs and $y \in \mathbb{R}$ are noisy function values (targets). With (9.1), the functional relationship between $\boldsymbol{x}$ and y is given as

$$y = f(\boldsymbol{x}) + \epsilon\,, \tag{9.2}$$

where $\epsilon \sim \mathcal{N}\big(0,\, \sigma^2\big)$ is independent, identically distributed (i.i.d.) Gaussian measurement noise with mean 0 and variance σ^2. Our objective is to find a function that is close (similar) to the unknown function f that generated the data and that generalizes well.

In this chapter, we focus on parametric models, i.e., we choose a parametrized function and find parameters $\boldsymbol{\theta}$ that "work well" for modeling the data. For the time being, we assume that the noise variance σ^2 is known and focus on learning the model parameters $\boldsymbol{\theta}$. In linear regression, we consider the special case that the parameters $\boldsymbol{\theta}$ appear linearly in our model. An example of linear regression is given by

$$p(y \mid \boldsymbol{x}, \boldsymbol{\theta}) = \mathcal{N}\big(y \mid \boldsymbol{x}^\top \boldsymbol{\theta},\, \sigma^2\big) \tag{9.3}$$

$$\iff y = \boldsymbol{x}^\top \boldsymbol{\theta} + \epsilon\,, \quad \epsilon \sim \mathcal{N}\big(0,\, \sigma^2\big)\,, \tag{9.4}$$

where $\boldsymbol{\theta} \in \mathbb{R}^D$ are the parameters we seek. The class of functions described by (9.4) are straight lines that pass through the origin. In (9.4), we chose a parametrization $f(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{\theta}$.

A Dirac delta (delta function) is zero everywhere except at a single point, and its integral is 1. It can be considered a Gaussian in the limit of $\sigma^2 \to 0$.

The *likelihood* in (9.3) is the probability density function of y evaluated at $\boldsymbol{x}^\top \boldsymbol{\theta}$. Note that the only source of uncertainty originates from the observation noise (as $\boldsymbol{x}$ and $\boldsymbol{\theta}$ are assumed known in (9.3)). Without observation noise, the relationship between $\boldsymbol{x}$ and y would be deterministic and (9.3) would be a Dirac delta.

likelihood

Example 9.1

For $x, \theta \in \mathbb{R}$ the linear regression model in (9.4) describes straight lines (linear functions), and the parameter θ is the slope of the line. Figure 9.2(a) shows some example functions for different values of θ.

Linear regression refers to models that are linear in the parameters.

The linear regression model in (9.3)–(9.4) is not only linear in the parameters, but also linear in the inputs x. Figure 9.2(a) shows examples of such functions. We will see later that $y = \boldsymbol{\phi}^\top(\boldsymbol{x})\boldsymbol{\theta}$ for nonlinear transformations $\boldsymbol{\phi}$ is also a linear regression model because "linear regression"

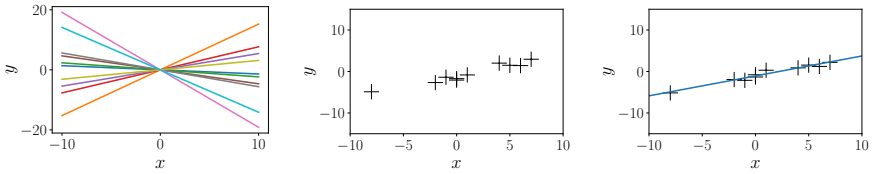


Figure 9.2 Linear regression example. (a) Example functions that fall into this category; (b) training set; (c) maximum likelihood estimate.

(a) Example functions (straight lines) that can be described using the linear model in (9.4).

(b) Training set.

(c) Maximum likelihood estimate.

refers to models that are "linear in the parameters", i.e., models that describe a function by a linear combination of input features. Here, a "feature" is a representation $\boldsymbol{\phi}(\boldsymbol{x})$ of the inputs $\boldsymbol{x}$.

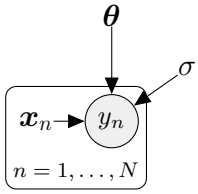
In the following, we will discuss in more detail how to find good parameters $\boldsymbol{\theta}$ and how to evaluate whether a parameter set "works well". For the time being, we assume that the noise variance σ^2 is known.

9.2 Parameter Estimation

Consider the linear regression setting (9.4) and assume we are given a *training set* $\mathcal{D} := \{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$ consisting of N inputs $\boldsymbol{x}_n \in \mathbb{R}^D$ and corresponding observations/targets $y_n \in \mathbb{R}$, $n = 1, \ldots, N$. The corresponding graphical model is given in Figure 9.3. Note that y_i and y_j are conditionally independent given their respective inputs $\boldsymbol{x}_i, \boldsymbol{x}_j$ so that the likelihood factorizes according to

training set

Figure 9.3 Probabilistic graphical model for linear regression. Observed random variables are shaded, deterministic/ known values are without circles.



$$p(\mathcal{Y} \mid \mathcal{X}, \boldsymbol{\theta}) = p(y_1, \ldots, y_N \mid \boldsymbol{x}_1, \ldots, \boldsymbol{x}_N, \boldsymbol{\theta}) \tag{9.5a}$$

$$= \prod_{n=1}^{N} p(y_n \mid \boldsymbol{x}_n, \boldsymbol{\theta}) = \prod_{n=1}^{N} \mathcal{N}\left(y_n \mid \boldsymbol{x}_n^\top \boldsymbol{\theta}, \sigma^2\right), \tag{9.5b}$$

where we defined $\mathcal{X} := \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$ and $\mathcal{Y} := \{y_1, \ldots, y_N\}$ as the sets of training inputs and corresponding targets, respectively. The likelihood and the factors $p(y_n \mid \boldsymbol{x}_n, \boldsymbol{\theta})$ are Gaussian due to the noise distribution; see (9.3).

In the following, we will discuss how to find optimal parameters $\boldsymbol{\theta}^* \in \mathbb{R}^D$ for the linear regression model (9.4). Once the parameters $\boldsymbol{\theta}^*$ are found, we can predict function values by using this parameter estimate in (9.4) so that at an arbitrary test input $\boldsymbol{x}_*$ the distribution of the corresponding target y_* is

$$p(y_* \mid \boldsymbol{x}_*, \boldsymbol{\theta}^*) = \mathcal{N}\left(y_* \mid \boldsymbol{x}_*^\top \boldsymbol{\theta}^*, \sigma^2\right). \tag{9.6}$$

In the following, we will have a look at parameter estimation by maximizing the likelihood, a topic that we already covered to some degree in Section 8.3.

9.2.1 Maximum Likelihood Estimation

maximum likelihood estimation

A widely used approach to finding the desired parameters $\boldsymbol{\theta}_{\text{ML}}$ is *maximum likelihood estimation*, where we find parameters $\boldsymbol{\theta}_{\text{ML}}$ that maximize the likelihood (9.5b). Intuitively, maximizing the likelihood means maximizing the predictive distribution of the training data given the model parameters. We obtain the maximum likelihood parameters as

Maximizing the likelihood means maximizing the predictive distribution of the (training) data given the parameters.

$$\boldsymbol{\theta}_{\text{ML}} = \arg\max_{\boldsymbol{\theta}} p(\mathcal{Y} \mid \mathcal{X}, \boldsymbol{\theta}) \,. \tag{9.7}$$

Remark. The likelihood $p(\boldsymbol{y} \mid \boldsymbol{x}, \boldsymbol{\theta})$ is not a probability distribution in $\boldsymbol{\theta}$: It is simply a function of the parameters $\boldsymbol{\theta}$ but does not integrate to 1 (i.e., it is unnormalized), and may not even be integrable with respect to $\boldsymbol{\theta}$. However, the likelihood in (9.7) is a normalized probability distribution in $\boldsymbol{y}$. ♢

The likelihood is not a probability distribution in the parameters.

To find the desired parameters $\boldsymbol{\theta}_{\text{ML}}$ that maximize the likelihood, we typically perform gradient ascent (or gradient descent on the negative likelihood). In the case of linear regression we consider here, however, a closed-form solution exists, which makes iterative gradient descent unnecessary. In practice, instead of maximizing the likelihood directly, we apply the log-transformation to the likelihood function and minimize the negative log-likelihood.

Since the logarithm is a (strictly) monotonically increasing function, the optimum of a function f is identical to the optimum of $\log f$.

Remark (Log-Transformation). Since the likelihood (9.5b) is a product of N Gaussian distributions, the log-transformation is useful since (a) it does not suffer from numerical underflow, and (b) the differentiation rules will turn out simpler. More specifically, numerical underflow will be a problem when we multiply N probabilities, where N is the number of data points, since we cannot represent very small numbers, such as 10^{-256}. Furthermore, the log-transform will turn the product into a sum of log-probabilities such that the corresponding gradient is a sum of individual gradients, instead of a repeated application of the product rule (5.46) to compute the gradient of a product of N terms. ♢

To find the optimal parameters $\boldsymbol{\theta}_{\text{ML}}$ of our linear regression problem, we minimize the negative log-likelihood

$$-\log p(\mathcal{Y} \mid \mathcal{X}, \boldsymbol{\theta}) = -\log \prod_{n=1}^{N} p(y_n \mid \boldsymbol{x}_n, \boldsymbol{\theta}) = -\sum_{n=1}^{N} \log p(y_n \mid \boldsymbol{x}_n, \boldsymbol{\theta}) \,, \tag{9.8}$$

where we exploited that the likelihood (9.5b) factorizes over the number of data points due to our independence assumption on the training set.

In the linear regression model (9.4), the likelihood is Gaussian (due to the Gaussian additive noise term), such that we arrive at

$$\log p(y_n \mid \boldsymbol{x}_n, \boldsymbol{\theta}) = -\frac{1}{2\sigma^2}(y_n - \boldsymbol{x}_n^\top \boldsymbol{\theta})^2 + \text{const} \,, \tag{9.9}$$

where the constant includes all terms independent of $\boldsymbol{\theta}$. Using (9.9) in the

negative log-likelihood (9.8), we obtain (ignoring the constant terms)

$$\mathcal{L}(\boldsymbol{\theta}) := \frac{1}{2\sigma^2}\sum_{n=1}^{N}(y_n - \boldsymbol{x}_n^\top\boldsymbol{\theta})^2 \tag{9.10a}$$

$$= \frac{1}{2\sigma^2}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})^\top(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}) = \frac{1}{2\sigma^2}\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta}\|^2\,, \tag{9.10b}$$

The negative log-likelihood function is also called *error function*.

design matrix

The squared error is often used as a measure of distance. Recall from Section 3.1 that $\|\boldsymbol{x}\|^2 = \boldsymbol{x}^\top\boldsymbol{x}$ if we choose the dot product as the inner product.

where we define the *design matrix* $\boldsymbol{X} := [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]^\top \in \mathbb{R}^{N\times D}$ as the collection of training inputs and $\boldsymbol{y} := [y_1, \ldots, y_N]^\top \in \mathbb{R}^N$ as a vector that collects all training targets. Note that the nth row in the design matrix $\boldsymbol{X}$ corresponds to the training input $\boldsymbol{x}_n$. In (9.10b), we used the fact that the sum of squared errors between the observations y_n and the corresponding model prediction $\boldsymbol{x}_n^\top\boldsymbol{\theta}$ equals the squared distance between $\boldsymbol{y}$ and $\boldsymbol{X}\boldsymbol{\theta}$.

With (9.10b), we have now a concrete form of the negative log-likelihood function we need to optimize. We immediately see that (9.10b) is quadratic in $\boldsymbol{\theta}$. This means that we can find a unique global solution $\boldsymbol{\theta}_{\text{ML}}$ for minimizing the negative log-likelihood $\mathcal{L}$. We can find the global optimum by computing the gradient of $\mathcal{L}$, setting it to $\mathbf{0}$ and solving for $\boldsymbol{\theta}$.

Using the results from Chapter 5, we compute the gradient of $\mathcal{L}$ with respect to the parameters as

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\boldsymbol{\theta}} = \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}\left(\frac{1}{2\sigma^2}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})^\top(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\theta})\right) \tag{9.11a}$$

$$= \frac{1}{2\sigma^2}\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}\left(\boldsymbol{y}^\top\boldsymbol{y} - 2\boldsymbol{y}^\top\boldsymbol{X}\boldsymbol{\theta} + \boldsymbol{\theta}^\top\boldsymbol{X}^\top\boldsymbol{X}\boldsymbol{\theta}\right) \tag{9.11b}$$

$$= \frac{1}{\sigma^2}(-\boldsymbol{y}^\top\boldsymbol{X} + \boldsymbol{\theta}^\top\boldsymbol{X}^\top\boldsymbol{X}) \in \mathbb{R}^{1\times D}\,. \tag{9.11c}$$

The maximum likelihood estimator $\boldsymbol{\theta}_{\text{ML}}$ solves $\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\boldsymbol{\theta}} = \mathbf{0}^\top$ (necessary optimality condition) and we obtain

Ignoring the possibility of duplicate data points, $\mathrm{rk}(\boldsymbol{X}) = D$ if $N \geqslant D$, i.e., we do not have more parameters than data points.

$$\frac{\mathrm{d}\mathcal{L}}{\mathrm{d}\boldsymbol{\theta}} = \mathbf{0}^\top \overset{(9.11c)}{\iff} \boldsymbol{\theta}_{\text{ML}}^\top\boldsymbol{X}^\top\boldsymbol{X} = \boldsymbol{y}^\top\boldsymbol{X} \tag{9.12a}$$

$$\iff \boldsymbol{\theta}_{\text{ML}}^\top = \boldsymbol{y}^\top\boldsymbol{X}(\boldsymbol{X}^\top\boldsymbol{X})^{-1} \tag{9.12b}$$

$$\iff \boldsymbol{\theta}_{\text{ML}} = (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y}\,. \tag{9.12c}$$

We could right-multiply the first equation by $(\boldsymbol{X}^\top\boldsymbol{X})^{-1}$ because $\boldsymbol{X}^\top\boldsymbol{X}$ is positive definite if $\mathrm{rk}(\boldsymbol{X}) = D$, where $\mathrm{rk}(\boldsymbol{X})$ denotes the rank of $\boldsymbol{X}$.

Remark. Setting the gradient to $\mathbf{0}^\top$ is a necessary and sufficient condition, and we obtain a global minimum since the Hessian $\nabla^2_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}) = \boldsymbol{X}^\top\boldsymbol{X} \in \mathbb{R}^{D\times D}$ is positive definite. ♢

Remark. The maximum likelihood solution in (9.12c) requires us to solve a system of linear equations of the form $\boldsymbol{A}\boldsymbol{\theta} = \boldsymbol{b}$ with $\boldsymbol{A} = (\boldsymbol{X}^\top\boldsymbol{X})$ and $\boldsymbol{b} = \boldsymbol{X}^\top\boldsymbol{y}$. ♢

Example 9.2 (Fitting Lines)
Let us have a look at Figure 9.2, where we aim to fit a straight line $f(x) = \theta x$, where θ is an unknown slope, to a dataset using maximum likelihood estimation. Examples of functions in this model class (straight lines) are shown in Figure 9.2(a). For the dataset shown in Figure 9.2(b), we find the maximum likelihood estimate of the slope parameter θ using (9.12c) and obtain the maximum likelihood linear function in Figure 9.2(c).

Maximum Likelihood Estimation with Features

So far, we considered the linear regression setting described in (9.4), which allowed us to fit straight lines to data using maximum likelihood estimation. However, straight lines are not sufficiently expressive when it comes to fitting more interesting data. Fortunately, linear regression offers us a way to fit nonlinear functions within the linear regression framework: Since "linear regression" only refers to "linear in the parameters", we can perform an arbitrary nonlinear transformation $\boldsymbol{\phi}(\boldsymbol{x})$ of the inputs $\boldsymbol{x}$ and then linearly combine the components of this transformation. The corresponding linear regression model is

Linear regression refers to "linear-in-the-parameters" regression models, but the inputs can undergo any nonlinear transformation.

$$\begin{aligned} p(y \mid \boldsymbol{x}, \boldsymbol{\theta}) &= \mathcal{N}\big(y \mid \boldsymbol{\phi}^\top(\boldsymbol{x})\boldsymbol{\theta},\, \sigma^2\big) \\ \iff y &= \boldsymbol{\phi}^\top(\boldsymbol{x})\boldsymbol{\theta} + \epsilon = \sum_{k=0}^{K-1} \theta_k \phi_k(\boldsymbol{x}) + \epsilon\,, \end{aligned} \tag{9.13}$$

where $\boldsymbol{\phi} : \mathbb{R}^D \to \mathbb{R}^K$ is a (nonlinear) transformation of the inputs $\boldsymbol{x}$ and $\phi_k : \mathbb{R}^D \to \mathbb{R}$ is the kth component of the *feature vector* $\boldsymbol{\phi}$. Note that the model parameters $\boldsymbol{\theta}$ still appear only linearly.

feature vector

Example 9.3 (Polynomial Regression)
We are concerned with a regression problem $y = \boldsymbol{\phi}^\top(x)\boldsymbol{\theta} + \epsilon$, where $x \in \mathbb{R}$ and $\boldsymbol{\theta} \in \mathbb{R}^K$. A transformation that is often used in this context is

$$\boldsymbol{\phi}(x) = \begin{bmatrix} \phi_0(x) \\ \phi_1(x) \\ \vdots \\ \phi_{K-1}(x) \end{bmatrix} = \begin{bmatrix} 1 \\ x \\ x^2 \\ x^3 \\ \vdots \\ x^{K-1} \end{bmatrix} \in \mathbb{R}^K\,. \tag{9.14}$$

This means that we "lift" the original one-dimensional input space into a K-dimensional feature space consisting of all monomials x^k for $k = 0, \ldots, K-1$. With these features, we can model polynomials of degree $\leqslant K-1$ within the framework of linear regression: A polynomial of degree

$K-1$ is

$$f(x) = \sum_{k=0}^{K-1} \theta_k x^k = \boldsymbol{\phi}^\top(x)\boldsymbol{\theta}\,, \tag{9.15}$$

where $\boldsymbol{\phi}$ is defined in (9.14) and $\boldsymbol{\theta} = [\theta_0, \ldots, \theta_{K-1}]^\top \in \mathbb{R}^K$ contains the (linear) parameters θ_k.

Let us now have a look at maximum likelihood estimation of the parameters $\boldsymbol{\theta}$ in the linear regression model (9.13). We consider training inputs $\boldsymbol{x}_n \in \mathbb{R}^D$ and targets $y_n \in \mathbb{R}$, $n = 1, \ldots, N$, and define the *feature matrix* (*design matrix*) as

feature matrix
design matrix

$$\boldsymbol{\Phi} := \begin{bmatrix} \boldsymbol{\phi}^\top(\boldsymbol{x}_1) \\ \vdots \\ \boldsymbol{\phi}^\top(\boldsymbol{x}_N) \end{bmatrix} = \begin{bmatrix} \phi_0(\boldsymbol{x}_1) & \cdots & \phi_{K-1}(\boldsymbol{x}_1) \\ \phi_0(\boldsymbol{x}_2) & \cdots & \phi_{K-1}(\boldsymbol{x}_2) \\ \vdots & & \vdots \\ \phi_0(\boldsymbol{x}_N) & \cdots & \phi_{K-1}(\boldsymbol{x}_N) \end{bmatrix} \in \mathbb{R}^{N \times K}\,, \tag{9.16}$$

where $\Phi_{ij} = \phi_j(\boldsymbol{x}_i)$ and $\phi_j : \mathbb{R}^D \to \mathbb{R}$.

Example 9.4 (Feature Matrix for Second-order Polynomials)
For a second-order polynomial and N training points $x_n \in \mathbb{R}, n = 1, \ldots, N$, the feature matrix is

$$\boldsymbol{\Phi} = \begin{bmatrix} 1 & x_1 & x_1^2 \\ 1 & x_2 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_N & x_N^2 \end{bmatrix}\,. \tag{9.17}$$

With the feature matrix $\boldsymbol{\Phi}$ defined in (9.16), the negative log-likelihood for the linear regression model (9.13) can be written as

$$-\log p(\mathcal{Y} \mid \mathcal{X}, \boldsymbol{\theta}) = \frac{1}{2\sigma^2}(\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\theta})^\top(\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\theta}) + \text{const}\,. \tag{9.18}$$

Comparing (9.18) with the negative log-likelihood in (9.10b) for the "feature-free" model, we immediately see we just need to replace $\boldsymbol{X}$ with $\boldsymbol{\Phi}$. Since both $\boldsymbol{X}$ and $\boldsymbol{\Phi}$ are independent of the parameters $\boldsymbol{\theta}$ that we wish to optimize, we arrive immediately at the *maximum likelihood estimate*

maximum likelihood estimate

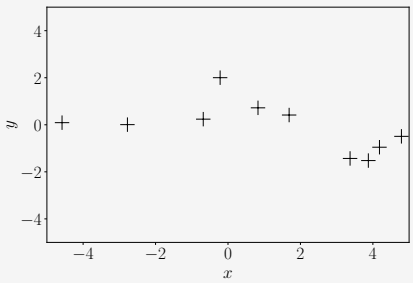
$$\boldsymbol{\theta}_{\text{ML}} = (\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\boldsymbol{y} \tag{9.19}$$

for the linear regression problem with nonlinear features defined in (9.13).

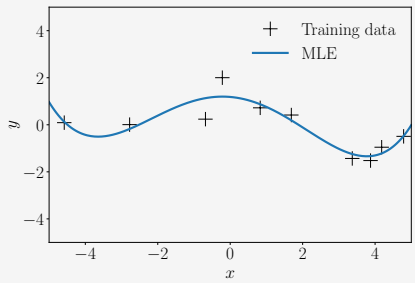
Remark. When we were working without features, we required $\boldsymbol{X}^\top\boldsymbol{X}$ to be invertible, which is the case when $\text{rk}(\boldsymbol{X}) = D$, i.e., the columns of $\boldsymbol{X}$

are linearly independent. In (9.19), we therefore require $\boldsymbol{\Phi}^\top \boldsymbol{\Phi} \in \mathbb{R}^{K \times K}$ to be invertible. This is the case if and only if $\mathrm{rk}(\boldsymbol{\Phi}) = K$. ♢

Example 9.5 (Maximum Likelihood Polynomial Fit)



(a) Regression dataset.



(b) Polynomial of degree 4 determined by maximum likelihood estimation.

Figure 9.4 Polynomial regression: (a) dataset consisting of (x_n, y_n) pairs, $n = 1, \ldots, 10$; (b) maximum likelihood polynomial of degree 4.

Consider the dataset in Figure 9.4(a). The dataset consists of $N = 10$ pairs (x_n, y_n), where $x_n \sim \mathcal{U}[-5, 5]$ and $y_n = -\sin(x_n/5) + \cos(x_n) + \epsilon$, where $\epsilon \sim \mathcal{N}(0,\, 0.2^2)$.

We fit a polynomial of degree 4 using maximum likelihood estimation, i.e., parameters $\boldsymbol{\theta}_{\mathrm{ML}}$ are given in (9.19). The maximum likelihood estimate yields function values $\boldsymbol{\phi}^\top(x_*)\boldsymbol{\theta}_{\mathrm{ML}}$ at any test location x_*. The result is shown in Figure 9.4(b).

Estimating the Noise Variance

Thus far, we assumed that the noise variance σ^2 is known. However, we can also use the principle of maximum likelihood estimation to obtain the maximum likelihood estimator σ^2_{ML} for the noise variance. To do this, we follow the standard procedure: We write down the log-likelihood, compute its derivative with respect to $\sigma^2 > 0$, set it to 0, and solve. The log-likelihood is given by

$$\log p(\mathcal{Y} \mid \mathcal{X}, \boldsymbol{\theta}, \sigma^2) = \sum_{n=1}^{N} \log \mathcal{N}\big(y_n \mid \boldsymbol{\phi}^\top(\boldsymbol{x}_n)\boldsymbol{\theta},\, \sigma^2\big) \tag{9.20a}$$

$$= \sum_{n=1}^{N} \left(-\frac{1}{2}\log(2\pi) - \frac{1}{2}\log\sigma^2 - \frac{1}{2\sigma^2}(y_n - \boldsymbol{\phi}^\top(\boldsymbol{x}_n)\boldsymbol{\theta})^2 \right) \tag{9.20b}$$

$$= -\frac{N}{2}\log\sigma^2 - \frac{1}{2\sigma^2}\underbrace{\sum_{n=1}^{N}(y_n - \boldsymbol{\phi}^\top(\boldsymbol{x}_n)\boldsymbol{\theta})^2}_{=:s} + \text{const}\,. \tag{9.20c}$$

The partial derivative of the log-likelihood with respect to σ^2 is then

$$\frac{\partial \log p(\mathcal{Y} \mid \mathcal{X}, \boldsymbol{\theta}, \sigma^2)}{\partial \sigma^2} = -\frac{N}{2\sigma^2} + \frac{1}{2\sigma^4} s = 0 \tag{9.21a}$$

$$\iff \frac{N}{2\sigma^2} = \frac{s}{2\sigma^4} \tag{9.21b}$$

so that we identify

$$\sigma^2_{\text{ML}} = \frac{s}{N} = \frac{1}{N} \sum_{n=1}^{N} (y_n - \boldsymbol{\phi}^\top(\boldsymbol{x}_n)\boldsymbol{\theta})^2 \,. \tag{9.22}$$

Therefore, the maximum likelihood estimate of the noise variance is the empirical mean of the squared distances between the noise-free function values $\boldsymbol{\phi}^\top(\boldsymbol{x}_n)\boldsymbol{\theta}$ and the corresponding noisy observations y_n at input locations $\boldsymbol{x}_n$.

9.2.2 Overfitting in Linear Regression

We just discussed how to use maximum likelihood estimation to fit linear models (e.g., polynomials) to data. We can evaluate the quality of the model by computing the error/loss incurred. One way of doing this is to compute the negative log-likelihood (9.10b), which we minimized to determine the maximum likelihood estimator. Alternatively, given that the noise parameter σ^2 is not a free model parameter, we can ignore the scaling by $1/\sigma^2$, so that we end up with a squared-error-loss function $\|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\theta}\|^2$. Instead of using this squared loss, we often use the *root mean square error* (*RMSE*)

root mean square error
RMSE

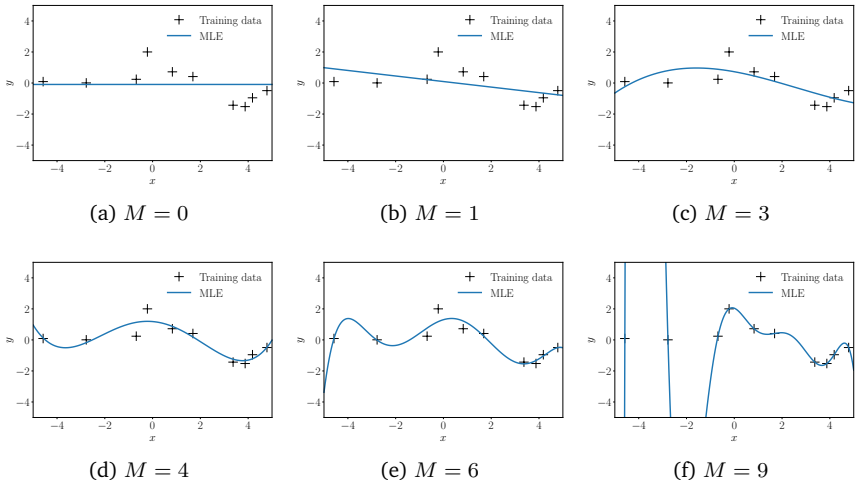
$$\sqrt{\frac{1}{N} \|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\theta}\|^2} = \sqrt{\frac{1}{N} \sum_{n=1}^{N} (y_n - \boldsymbol{\phi}^\top(\boldsymbol{x}_n)\boldsymbol{\theta})^2} \,, \tag{9.23}$$

which (a) allows us to compare errors of datasets with different sizes and (b) has the same scale and the same units as the observed function values y_n. For example, if we fit a model that maps post-codes ($\boldsymbol{x}$ is given in latitude, longitude) to house prices (y-values are EUR) then the RMSE is also measured in EUR, whereas the squared error is given in EUR^2. If we choose to include the factor σ^2 from the original negative log-likelihood (9.10b), then we end up with a unitless objective, i.e., in the preceding example, our objective would no longer be in EUR or EUR^2.

The RMSE is normalized.

The negative log-likelihood is unitless.

For model selection (see Section 8.6), we can use the RMSE (or the negative log-likelihood) to determine the best degree of the polynomial by finding the polynomial degree M that minimizes the objective. Given that the polynomial degree is a natural number, we can perform a brute-force search and enumerate all (reasonable) values of M. For a training set of size N it is sufficient to test $0 \leqslant M \leqslant N-1$. For $M < N$, the maximum likelihood estimator is unique. For $M \geqslant N$, we have more parameters



(a) $M = 0$ (b) $M = 1$ (c) $M = 3$

(d) $M = 4$ (e) $M = 6$ (f) $M = 9$

Figure 9.5 Maximum likelihood fits for different polynomial degrees M.

than data points, and would need to solve an underdetermined system of linear equations ($\boldsymbol{\Phi}^\top\boldsymbol{\Phi}$ in (9.19) would also no longer be invertible) so that there are infinitely many possible maximum likelihood estimators.

Figure 9.5 shows a number of polynomial fits determined by maximum likelihood for the dataset from Figure 9.4(a) with $N = 10$ observations. We notice that polynomials of low degree (e.g., constants ($M = 0$) or linear ($M = 1$)) fit the data poorly and, hence, are poor representations of the true underlying function. For degrees $M = 3, \dots, 6$, the fits look plausible and smoothly interpolate the data. When we go to higher-degree polynomials, we notice that they fit the data better and better. In the extreme case of $M = N - 1 = 9$, the function will pass through every single data point. However, these high-degree polynomials oscillate wildly and are a poor representation of the underlying function that generated the data, such that we suffer from *overfitting*.

The case of $M = N - 1$ is extreme in the sense that otherwise the null space of the corresponding system of linear equations would be non-trivial, and we would have infinitely many optimal solutions to the linear regression problem.

overfitting

Remember that the goal is to achieve good generalization by making accurate predictions for new (unseen) data. We obtain some quantitative insight into the dependence of the generalization performance on the polynomial of degree M by considering a separate test set comprising 200 data points generated using exactly the same procedure used to generate the training set. As test inputs, we chose a linear grid of 200 points in the interval of $[-5, 5]$. For each choice of M, we evaluate the RMSE (9.23) for both the training data and the test data.

Looking now at the test error, which is a qualitive measure of the generalization properties of the corresponding polynomial, we notice that initially the test error decreases; see Figure 9.6 (orange). For fourth-order polynomials, the test error is relatively low and stays relatively constant up to degree 5. However, from degree 6 onward the test error increases significantly, and high-order polynomials have very bad generalization properties. In this particular example, this also is evident from the corresponding

Note that the noise variance $\sigma^2 > 0$.

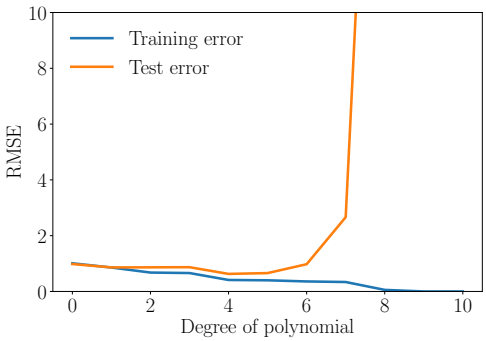


Figure 9.6 Training and test error.

training error

test error

maximum likelihood fits in Figure 9.5. Note that the *training error* (blue curve in Figure 9.6) never increases when the degree of the polynomial increases. In our example, the best generalization (the point of the smallest *test error*) is obtained for a polynomial of degree $M = 4$.

9.2.3 Maximum A Posteriori Estimation

We just saw that maximum likelihood estimation is prone to overfitting. We often observe that the magnitude of the parameter values becomes relatively large if we run into overfitting (Bishop, 2006).

To mitigate the effect of huge parameter values, we can place a prior distribution $p(\boldsymbol{\theta})$ on the parameters. The prior distribution explicitly encodes what parameter values are plausible (before having seen any data). For example, a Gaussian prior $p(\theta) = \mathcal{N}(0, 1)$ on a single parameter θ encodes that parameter values are expected lie in the interval $[-2, 2]$ (two standard deviations around the mean value). Once a dataset $\mathcal{X}, \mathcal{Y}$ is available, instead of maximizing the likelihood we seek parameters that maximize the posterior distribution $p(\boldsymbol{\theta} \mid \mathcal{X}, \mathcal{Y})$. This procedure is called *maximum a posteriori* (*MAP*) estimation.

maximum a posteriori

MAP

The posterior over the parameters $\boldsymbol{\theta}$, given the training data $\mathcal{X}, \mathcal{Y}$, is obtained by applying Bayes' theorem (Section 6.3) as

$$p(\boldsymbol{\theta} \mid \mathcal{X}, \mathcal{Y}) = \frac{p(\mathcal{Y} \mid \mathcal{X}, \boldsymbol{\theta}) p(\boldsymbol{\theta})}{p(\mathcal{Y} \mid \mathcal{X})} . \tag{9.24}$$

Since the posterior explicitly depends on the parameter prior $p(\boldsymbol{\theta})$, the prior will have an effect on the parameter vector we find as the maximizer of the posterior. We will see this more explicitly in the following. The parameter vector $\boldsymbol{\theta}_{\text{MAP}}$ that maximizes the posterior (9.24) is the MAP estimate.

To find the MAP estimate, we follow steps that are similar in flavor to maximum likelihood estimation. We start with the log-transform and compute the log-posterior as

$$\log p(\boldsymbol{\theta} \mid \mathcal{X}, \mathcal{Y}) = \log p(\mathcal{Y} \mid \mathcal{X}, \boldsymbol{\theta}) + \log p(\boldsymbol{\theta}) + \text{const} , \tag{9.25}$$

where the constant comprises the terms that are independent of $\boldsymbol{\theta}$. We see that the log-posterior in (9.25) is the sum of the log-likelihood $p(\mathcal{Y} \mid \mathcal{X}, \boldsymbol{\theta})$ and the log-prior $\log p(\boldsymbol{\theta})$ so that the MAP estimate will be a "compromise" between the prior (our suggestion for plausible parameter values before observing data) and the data-dependent likelihood.

To find the MAP estimate $\boldsymbol{\theta}_{\text{MAP}}$, we minimize the negative log-posterior distribution with respect to $\boldsymbol{\theta}$, i.e., we solve

$$\boldsymbol{\theta}_{\text{MAP}} \in \arg\min_{\boldsymbol{\theta}} \{-\log p(\mathcal{Y} \mid \mathcal{X}, \boldsymbol{\theta}) - \log p(\boldsymbol{\theta})\} . \tag{9.26}$$

The gradient of the negative log-posterior with respect to $\boldsymbol{\theta}$ is

$$-\frac{\mathrm{d} \log p(\boldsymbol{\theta} \mid \mathcal{X}, \mathcal{Y})}{\mathrm{d}\boldsymbol{\theta}} = -\frac{\mathrm{d} \log p(\mathcal{Y} \mid \mathcal{X}, \boldsymbol{\theta})}{\mathrm{d}\boldsymbol{\theta}} - \frac{\mathrm{d} \log p(\boldsymbol{\theta})}{\mathrm{d}\boldsymbol{\theta}} , \tag{9.27}$$

where we identify the first term on the right-hand side as the gradient of the negative log-likelihood from (9.11c).

With a (conjugate) Gaussian prior $p(\boldsymbol{\theta}) = \mathcal{N}(\mathbf{0}, b^2\boldsymbol{I})$ on the parameters $\boldsymbol{\theta}$, the negative log-posterior for the linear regression setting (9.13), we obtain the negative log posterior

$$-\log p(\boldsymbol{\theta} \mid \mathcal{X}, \mathcal{Y}) = \frac{1}{2\sigma^2}(\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\theta})^\top(\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\theta}) + \frac{1}{2b^2}\boldsymbol{\theta}^\top\boldsymbol{\theta} + \text{const} . \tag{9.28}$$

Here, the first term corresponds to the contribution from the log-likelihood, and the second term originates from the log-prior. The gradient of the log-posterior with respect to the parameters $\boldsymbol{\theta}$ is then

$$-\frac{\mathrm{d} \log p(\boldsymbol{\theta} \mid \mathcal{X}, \mathcal{Y})}{\mathrm{d}\boldsymbol{\theta}} = \frac{1}{\sigma^2}(\boldsymbol{\theta}^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi} - \boldsymbol{y}^\top\boldsymbol{\Phi}) + \frac{1}{b^2}\boldsymbol{\theta}^\top . \tag{9.29}$$

We will find the MAP estimate $\boldsymbol{\theta}_{\text{MAP}}$ by setting this gradient to $\mathbf{0}^\top$ and solving for $\boldsymbol{\theta}_{\text{MAP}}$. We obtain

$$\frac{1}{\sigma^2}(\boldsymbol{\theta}^\top\boldsymbol{\Phi}^\top\boldsymbol{\Phi} - \boldsymbol{y}^\top\boldsymbol{\Phi}) + \frac{1}{b^2}\boldsymbol{\theta}^\top = \mathbf{0}^\top \tag{9.30a}$$

$$\iff \boldsymbol{\theta}^\top\left(\frac{1}{\sigma^2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \frac{1}{b^2}\boldsymbol{I}\right) - \frac{1}{\sigma^2}\boldsymbol{y}^\top\boldsymbol{\Phi} = \mathbf{0}^\top \tag{9.30b}$$

$$\iff \boldsymbol{\theta}^\top\left(\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \frac{\sigma^2}{b^2}\boldsymbol{I}\right) = \boldsymbol{y}^\top\boldsymbol{\Phi} \tag{9.30c}$$

$$\iff \boldsymbol{\theta}^\top = \boldsymbol{y}^\top\boldsymbol{\Phi}\left(\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \frac{\sigma^2}{b^2}\boldsymbol{I}\right)^{-1} \tag{9.30d}$$

so that the MAP estimate is (by transposing both sides of the last equality)

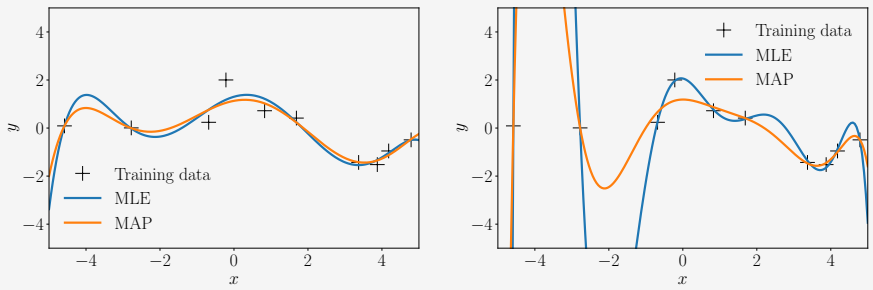
$$\boldsymbol{\theta}_{\text{MAP}} = \left(\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \frac{\sigma^2}{b^2}\boldsymbol{I}\right)^{-1}\boldsymbol{\Phi}^\top\boldsymbol{y} . \tag{9.31}$$

$\boldsymbol{\Phi}^\top\boldsymbol{\Phi}$ is symmetric, positive semi definite. The additional term in (9.31) is strictly positive definite so that the inverse exists.

Comparing the MAP estimate in (9.31) with the maximum likelihood estimate in (9.19), we see that the only difference between both solutions is the additional term $\frac{\sigma^2}{b^2}\boldsymbol{I}$ in the inverse matrix. This term ensures that

$\boldsymbol{\Phi}^\top \boldsymbol{\Phi} + \frac{\sigma^2}{b^2}\boldsymbol{I}$ is symmetric and strictly positive definite (i.e., its inverse exists and the MAP estimate is the unique solution of a system of linear equations). Moreover, it reflects the impact of the regularizer.

Example 9.6 (MAP Estimation for Polynomial Regression)
In the polynomial regression example from Section 9.2.1, we place a Gaussian prior $p(\boldsymbol{\theta}) = \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ on the parameters $\boldsymbol{\theta}$ and determine the MAP estimates according to (9.31). In Figure 9.7, we show both the maximum likelihood and the MAP estimates for polynomials of degree 6 (left) and degree 8 (right). The prior (regularizer) does not play a significant role for the low-degree polynomial, but keeps the function relatively smooth for higher-degree polynomials. Although the MAP estimate can push the boundaries of overfitting, it is not a general solution to this problem, so we need a more principled approach to tackle overfitting.



(a) Polynomials of degree 6.

(b) Polynomials of degree 8.

Figure 9.7 Polynomial regression: maximum likelihood and MAP estimates. (a) Polynomials of degree 6; (b) polynomials of degree 8.

9.2.4 MAP Estimation as Regularization

Instead of placing a prior distribution on the parameters $\boldsymbol{\theta}$, it is also possible to mitigate the effect of overfitting by penalizing the amplitude of the parameter by means of *regularization*. In *regularized least squares*, we consider the loss function

regularization
regularized least squares

$$\|\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\theta}\|^2 + \lambda \|\boldsymbol{\theta}\|_2^2 \,, \tag{9.32}$$

which we minimize with respect to $\boldsymbol{\theta}$ (see Section 8.2.3). Here, the first term is a *data-fit term* (also called *misfit term*), which is proportional to the negative log-likelihood; see (9.10b). The second term is called the *regularizer*, and the *regularization parameter* $\lambda \geqslant 0$ controls the "strictness" of the regularization.

data-fit term
misfit term
regularizer
regularization parameter

Remark. Instead of the Euclidean norm $\|\cdot\|_2$, we can choose any p-norm $\|\cdot\|_p$ in (9.32). In practice, smaller values for p lead to sparser solutions. Here, "sparse" means that many parameter values $\theta_d = 0$, which is also

useful for variable selection. For $p = 1$, the regularizer is called *LASSO* (least absolute shrinkage and selection operator) and was proposed by Tibshirani (1996). ♢

LASSO

The regularizer $\lambda \|\boldsymbol{\theta}\|_2^2$ in (9.32) can be interpreted as a negative log-Gaussian prior, which we use in MAP estimation; see (9.26). More specifically, with a Gaussian prior $p(\boldsymbol{\theta}) = \mathcal{N}(\mathbf{0}, b^2\boldsymbol{I})$, we obtain the negative log-Gaussian prior

$$-\log p(\boldsymbol{\theta}) = \frac{1}{2b^2}\|\boldsymbol{\theta}\|_2^2 + \text{const} \tag{9.33}$$

so that for $\lambda = \frac{1}{2b^2}$ the regularization term and the negative log-Gaussian prior are identical.

Given that the regularized least-squares loss function in (9.32) consists of terms that are closely related to the negative log-likelihood plus a negative log-prior, it is not surprising that, when we minimize this loss, we obtain a solution that closely resembles the MAP estimate in (9.31). More specifically, minimizing the regularized least-squares loss function yields

$$\boldsymbol{\theta}_{\text{RLS}} = (\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \lambda\boldsymbol{I})^{-1}\boldsymbol{\Phi}^\top\boldsymbol{y}\,, \tag{9.34}$$

which is identical to the MAP estimate in (9.31) for $\lambda = \frac{\sigma^2}{b^2}$, where σ^2 is the noise variance and b^2 the variance of the (isotropic) Gaussian prior $p(\boldsymbol{\theta}) = \mathcal{N}(\mathbf{0}, b^2\boldsymbol{I})$.

A point estimate is a single specific parameter value, unlike a distribution over plausible parameter settings.

So far, we have covered parameter estimation using maximum likelihood and MAP estimation where we found point estimates $\boldsymbol{\theta}^*$ that optimize an objective function (likelihood or posterior). We saw that both maximum likelihood and MAP estimation can lead to overfitting. In the next section, we will discuss Bayesian linear regression, where we use Bayesian inference (Section 8.4) to find a posterior distribution over the unknown parameters, which we subsequently use to make predictions. More specifically, for predictions we will average over all plausible sets of parameters instead of focusing on a point estimate.

9.3 Bayesian Linear Regression

Previously, we looked at linear regression models where we estimated the model parameters $\boldsymbol{\theta}$, e.g., by means of maximum likelihood or MAP estimation. We discovered that MLE can lead to severe overfitting, in particular, in the small-data regime. MAP addresses this issue by placing a prior on the parameters that plays the role of a regularizer.

Bayesian linear regression

Bayesian linear regression pushes the idea of the parameter prior a step further and does not even attempt to compute a point estimate of the parameters, but instead the full posterior distribution over the parameters is taken into account when making predictions. This means we do not fit any parameters, but we compute a mean over all plausible parameters settings (according to the posterior).

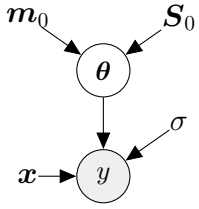
9.3.1 Model

In Bayesian linear regression, we consider the model

$$\begin{aligned} &\text{prior} && p(\boldsymbol{\theta}) = \mathcal{N}\left(\boldsymbol{m}_0,\, \boldsymbol{S}_0\right), \\ &\text{likelihood} && p(y \mid \boldsymbol{x}, \boldsymbol{\theta}) = \mathcal{N}\left(y \mid \boldsymbol{\phi}^\top(\boldsymbol{x})\boldsymbol{\theta},\, \sigma^2\right), \end{aligned} \tag{9.35}$$

where we now explicitly place a Gaussian prior $p(\boldsymbol{\theta}) = \mathcal{N}\left(\boldsymbol{m}_0,\, \boldsymbol{S}_0\right)$ on $\boldsymbol{\theta}$, which turns the parameter vector into a random variable. This allows us to write down the corresponding graphical model in Figure 9.8, where we made the parameters of the Gaussian prior on $\boldsymbol{\theta}$ explicit. The full probabilistic model, i.e., the joint distribution of observed and unobserved random variables, y and $\boldsymbol{\theta}$, respectively, is

Figure 9.8 Graphical model for Bayesian linear regression.



$$p(y, \boldsymbol{\theta} \mid \boldsymbol{x}) = p(y \mid \boldsymbol{x}, \boldsymbol{\theta}) p(\boldsymbol{\theta}) \,. \tag{9.36}$$

9.3.2 Prior Predictions

In practice, we are usually not so much interested in the parameter values $\boldsymbol{\theta}$ themselves. Instead, our focus often lies in the predictions we make with those parameter values. In a Bayesian setting, we take the parameter distribution and average over all plausible parameter settings when we make predictions. More specifically, to make predictions at an input $\boldsymbol{x}_*$, we integrate out $\boldsymbol{\theta}$ and obtain

$$p(y_* \mid \boldsymbol{x}_*) = \int p(y_* \mid \boldsymbol{x}_*, \boldsymbol{\theta}) p(\boldsymbol{\theta}) \mathrm{d}\boldsymbol{\theta} = \mathbb{E}_{\boldsymbol{\theta}}[p(y_* \mid \boldsymbol{x}_*, \boldsymbol{\theta})] \,, \tag{9.37}$$

which we can interpret as the average prediction of $y_* \mid \boldsymbol{x}_*, \boldsymbol{\theta}$ for all plausible parameters $\boldsymbol{\theta}$ according to the prior distribution $p(\boldsymbol{\theta})$. Note that predictions using the prior distribution only require us to specify the input $\boldsymbol{x}_*$, but no training data.

In our model (9.35), we chose a conjugate (Gaussian) prior on $\boldsymbol{\theta}$ so that the predictive distribution is Gaussian as well (and can be computed in closed form): With the prior distribution $p(\boldsymbol{\theta}) = \mathcal{N}\left(\boldsymbol{m}_0,\, \boldsymbol{S}_0\right)$, we obtain the predictive distribution as

$$p(y_* \mid \boldsymbol{x}_*) = \mathcal{N}\left(\boldsymbol{\phi}^\top(\boldsymbol{x}_*)\boldsymbol{m}_0,\, \boldsymbol{\phi}^\top(\boldsymbol{x}_*)\boldsymbol{S}_0\boldsymbol{\phi}(\boldsymbol{x}_*) + \sigma^2\right), \tag{9.38}$$

where we exploited that (i) the prediction is Gaussian due to conjugacy (see Section 6.6) and the marginalization property of Gaussians (see Section 6.5), (ii) the Gaussian noise is independent so that

$$\mathbb{V}[y_*] = \mathbb{V}_{\boldsymbol{\theta}}[\boldsymbol{\phi}^\top(\boldsymbol{x}_*)\boldsymbol{\theta}] + \mathbb{V}_\epsilon[\epsilon] \,, \tag{9.39}$$

and (iii) y_* is a linear transformation of $\boldsymbol{\theta}$ so that we can apply the rules for computing the mean and covariance of the prediction analytically by using (6.50) and (6.51), respectively. In (9.38), the term $\boldsymbol{\phi}^\top(\boldsymbol{x}_*)\boldsymbol{S}_0\boldsymbol{\phi}(\boldsymbol{x}_*)$ in the predictive variance explicitly accounts for the uncertainty associated

with the parameters $\boldsymbol{\theta}$, whereas σ^2 is the uncertainty contribution due to the measurement noise.

If we are interested in predicting noise-free function values $f(\boldsymbol{x}_*) = \boldsymbol{\phi}^\top(\boldsymbol{x}_*)\boldsymbol{\theta}$ instead of the noise-corrupted targets y_* we obtain

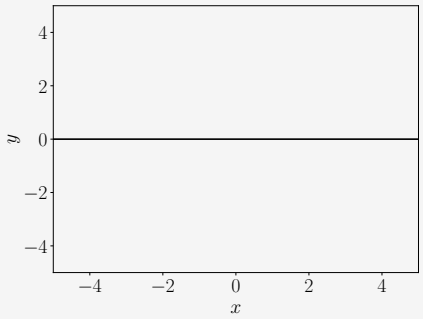
$$p(f(\boldsymbol{x}_*)) = \mathcal{N}\left(\boldsymbol{\phi}^\top(\boldsymbol{x}_*)\boldsymbol{m}_0,\ \boldsymbol{\phi}^\top(\boldsymbol{x}_*)\boldsymbol{S}_0\boldsymbol{\phi}(\boldsymbol{x}_*)\right), \tag{9.40}$$

which only differs from (9.38) in the omission of the noise variance σ^2 in the predictive variance.

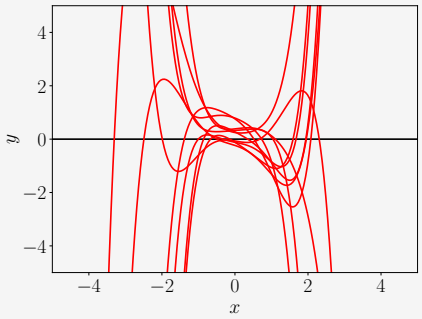
The parameter distribution $p(\boldsymbol{\theta})$ induces a distribution over functions.

Remark (Distribution over Functions). Since we can represent the distribution $p(\boldsymbol{\theta})$ using a set of samples $\boldsymbol{\theta}_i$ and every sample $\boldsymbol{\theta}_i$ gives rise to a function $f_i(\cdot) = \boldsymbol{\theta}_i^\top\boldsymbol{\phi}(\cdot)$, it follows that the parameter distribution $p(\boldsymbol{\theta})$ induces a distribution $p(f(\cdot))$ over functions. Here we use the notation $(\cdot)$ to explicitly denote a functional relationship. ♢

Example 9.7 (Prior over Functions)



(a) Prior distribution over functions.



(b) Samples from the prior distribution over functions.

Figure 9.9 Prior over functions. (a) Distribution over functions represented by the mean function (black line) and the marginal uncertainties (shaded), representing the 67% and 95% confidence bounds, respectively; (b) samples from the prior over functions, which are induced by the samples from the parameter prior.

Let us consider a Bayesian linear regression problem with polynomials of degree 5. We choose a parameter prior $p(\boldsymbol{\theta}) = \mathcal{N}\left(\mathbf{0}, \frac{1}{4}\boldsymbol{I}\right)$. Figure 9.9 visualizes the induced prior distribution over functions (shaded area: dark gray: 67% confidence bound; light gray: 95% confidence bound) induced by this parameter prior, including some function samples from this prior.

A function sample is obtained by first sampling a parameter vector $\boldsymbol{\theta}_i \sim p(\boldsymbol{\theta})$ and then computing $f_i(\cdot) = \boldsymbol{\theta}_i^\top\boldsymbol{\phi}(\cdot)$. We used 200 input locations $x_* \in [-5, 5]$ to which we apply the feature function $\boldsymbol{\phi}(\cdot)$. The uncertainty (represented by the shaded area) in Figure 9.9 is solely due to the parameter uncertainty because we considered the noise-free predictive distribution (9.40).

So far, we looked at computing predictions using the parameter prior $p(\boldsymbol{\theta})$. However, when we have a parameter posterior (given some training data $\mathcal{X}$, $\mathcal{Y}$), the same principles for prediction and inference hold as in (9.37) – we just need to replace the prior $p(\boldsymbol{\theta})$ with the posterior

$p(\boldsymbol{\theta} \mid \mathcal{X}, \mathcal{Y})$. In the following, we will derive the posterior distribution in detail before using it to make predictions.

9.3.3 Posterior Distribution

Given a training set of inputs $\boldsymbol{x}_n \in \mathbb{R}^D$ and corresponding observations $y_n \in \mathbb{R}$, $n = 1, \ldots, N$, we compute the posterior over the parameters using Bayes' theorem as

$$p(\boldsymbol{\theta} \mid \mathcal{X}, \mathcal{Y}) = \frac{p(\mathcal{Y} \mid \mathcal{X}, \boldsymbol{\theta}) p(\boldsymbol{\theta})}{p(\mathcal{Y} \mid \mathcal{X})}, \tag{9.41}$$

where $\mathcal{X}$ is the set of training inputs and $\mathcal{Y}$ the collection of corresponding training targets. Furthermore, $p(\mathcal{Y} \mid \mathcal{X}, \boldsymbol{\theta})$ is the likelihood, $p(\boldsymbol{\theta})$ the parameter prior, and

$$p(\mathcal{Y} \mid \mathcal{X}) = \int p(\mathcal{Y} \mid \mathcal{X}, \boldsymbol{\theta}) p(\boldsymbol{\theta}) \mathrm{d}\boldsymbol{\theta} = \mathbb{E}_{\boldsymbol{\theta}}[p(\mathcal{Y} \mid \mathcal{X}, \boldsymbol{\theta})] \tag{9.42}$$

marginal likelihood
evidence
The marginal likelihood is the expected likelihood under the parameter prior.

the *marginal likelihood/evidence*, which is independent of the parameters $\boldsymbol{\theta}$ and ensures that the posterior is normalized, i.e., it integrates to 1. We can think of the marginal likelihood as the likelihood averaged over all possible parameter settings (with respect to the prior distribution $p(\boldsymbol{\theta})$).

Theorem 9.1 (Parameter Posterior)**.** *In our model* (9.35)*, the parameter posterior* (9.41) *can be computed in closed form as*

$$p(\boldsymbol{\theta} \mid \mathcal{X}, \mathcal{Y}) = \mathcal{N}(\boldsymbol{\theta} \mid \boldsymbol{m}_N, \boldsymbol{S}_N), \tag{9.43a}$$

$$\boldsymbol{S}_N = (\boldsymbol{S}_0^{-1} + \sigma^{-2} \boldsymbol{\Phi}^\top \boldsymbol{\Phi})^{-1}, \tag{9.43b}$$

$$\boldsymbol{m}_N = \boldsymbol{S}_N (\boldsymbol{S}_0^{-1} \boldsymbol{m}_0 + \sigma^{-2} \boldsymbol{\Phi}^\top \boldsymbol{y}), \tag{9.43c}$$

where the subscript N indicates the size of the training set.

Proof Bayes' theorem tells us that the posterior $p(\boldsymbol{\theta} \mid \mathcal{X}, \mathcal{Y})$ is proportional to the product of the likelihood $p(\mathcal{Y} \mid \mathcal{X}, \boldsymbol{\theta})$ and the prior $p(\boldsymbol{\theta})$:

$$\text{Posterior} \quad p(\boldsymbol{\theta} \mid \mathcal{X}, \mathcal{Y}) = \frac{p(\mathcal{Y} \mid \mathcal{X}, \boldsymbol{\theta}) p(\boldsymbol{\theta})}{p(\mathcal{Y} \mid \mathcal{X})} \tag{9.44a}$$

$$\text{Likelihood} \quad p(\mathcal{Y} \mid \mathcal{X}, \boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{y} \mid \boldsymbol{\Phi}\boldsymbol{\theta}, \sigma^2 \boldsymbol{I}) \tag{9.44b}$$

$$\text{Prior} \quad p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta} \mid \boldsymbol{m}_0, \boldsymbol{S}_0). \tag{9.44c}$$

Instead of looking at the product of the prior and the likelihood, we can transform the problem into log-space and solve for the mean and covariance of the posterior by completing the squares.

The sum of the log-prior and the log-likelihood is

$$\log \mathcal{N}(\boldsymbol{y} \mid \boldsymbol{\Phi}\boldsymbol{\theta}, \sigma^2 \boldsymbol{I}) + \log \mathcal{N}(\boldsymbol{\theta} \mid \boldsymbol{m}_0, \boldsymbol{S}_0) \tag{9.45a}$$

$$= -\frac{1}{2}\big(\sigma^{-2}(\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\theta})^\top (\boldsymbol{y} - \boldsymbol{\Phi}\boldsymbol{\theta}) + (\boldsymbol{\theta} - \boldsymbol{m}_0)^\top \boldsymbol{S}_0^{-1} (\boldsymbol{\theta} - \boldsymbol{m}_0)\big) + \text{const} \tag{9.45b}$$

where the constant contains terms independent of $\boldsymbol{\theta}$. We will ignore the constant in the following. We now factorize (9.45b), which yields

$$-\frac{1}{2}(\sigma^{-2}\boldsymbol{y}^\top\boldsymbol{y} - 2\sigma^{-2}\boldsymbol{y}^\top\boldsymbol{\Phi}\boldsymbol{\theta} + \boldsymbol{\theta}^\top\sigma^{-2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{\theta} + \boldsymbol{\theta}^\top\boldsymbol{S}_0^{-1}\boldsymbol{\theta} - 2\boldsymbol{m}_0^\top\boldsymbol{S}_0^{-1}\boldsymbol{\theta} + \boldsymbol{m}_0^\top\boldsymbol{S}_0^{-1}\boldsymbol{m}_0) \tag{9.46a}$$

$$= -\frac{1}{2}(\boldsymbol{\theta}^\top(\sigma^{-2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \boldsymbol{S}_0^{-1})\boldsymbol{\theta} - 2(\sigma^{-2}\boldsymbol{\Phi}^\top\boldsymbol{y} + \boldsymbol{S}_0^{-1}\boldsymbol{m}_0)^\top\boldsymbol{\theta}) + \text{const}\,, \tag{9.46b}$$

where the constant contains the black terms in (9.46a), which are independent of $\boldsymbol{\theta}$. The orange terms are terms that are linear in $\boldsymbol{\theta}$, and the blue terms are the ones that are quadratic in $\boldsymbol{\theta}$. Inspecting (9.46b), we find that this equation is quadratic in $\boldsymbol{\theta}$. The fact that the unnormalized log-posterior distribution is a (negative) quadratic form implies that the posterior is Gaussian, i.e.,

$$p(\boldsymbol{\theta}\,|\,\mathcal{X},\mathcal{Y}) = \exp(\log p(\boldsymbol{\theta}\,|\,\mathcal{X},\mathcal{Y})) \propto \exp(\log p(\mathcal{Y}\,|\,\mathcal{X},\boldsymbol{\theta}) + \log p(\boldsymbol{\theta})) \tag{9.47a}$$

$$\propto \exp\Big(-\frac{1}{2}(\boldsymbol{\theta}^\top(\sigma^{-2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \boldsymbol{S}_0^{-1})\boldsymbol{\theta} - 2(\sigma^{-2}\boldsymbol{\Phi}^\top\boldsymbol{y} + \boldsymbol{S}_0^{-1}\boldsymbol{m}_0)^\top\boldsymbol{\theta})\Big)\,, \tag{9.47b}$$

where we used (9.46b) in the last expression.

The remaining task is it to bring this (unnormalized) Gaussian into the form that is proportional to $\mathcal{N}(\boldsymbol{\theta}\,|\,\boldsymbol{m}_N,\,\boldsymbol{S}_N)$, i.e., we need to identify the mean $\boldsymbol{m}_N$ and the covariance matrix $\boldsymbol{S}_N$. To do this, we use the concept of *completing the squares*. The desired log-posterior is

completing the squares

$$\log\mathcal{N}(\boldsymbol{\theta}\,|\,\boldsymbol{m}_N,\,\boldsymbol{S}_N) = -\frac{1}{2}(\boldsymbol{\theta}-\boldsymbol{m}_N)^\top\boldsymbol{S}_N^{-1}(\boldsymbol{\theta}-\boldsymbol{m}_N) + \text{const} \tag{9.48a}$$

$$= -\frac{1}{2}(\boldsymbol{\theta}^\top\boldsymbol{S}_N^{-1}\boldsymbol{\theta} - 2\boldsymbol{m}_N^\top\boldsymbol{S}_N^{-1}\boldsymbol{\theta} + \boldsymbol{m}_N^\top\boldsymbol{S}_N^{-1}\boldsymbol{m}_N)\,. \tag{9.48b}$$

Here, we factorized the quadratic form $(\boldsymbol{\theta}-\boldsymbol{m}_N)^\top\boldsymbol{S}_N^{-1}(\boldsymbol{\theta}-\boldsymbol{m}_N)$ into a term that is quadratic in $\boldsymbol{\theta}$ alone (blue), a term that is linear in $\boldsymbol{\theta}$ (orange), and a constant term (black). This allows us now to find $\boldsymbol{S}_N$ and $\boldsymbol{m}_N$ by matching the colored expressions in (9.46b) and (9.48b), which yields

Since $p(\boldsymbol{\theta}\,|\,\mathcal{X},\mathcal{Y}) = \mathcal{N}(\boldsymbol{m}_N,\,\boldsymbol{S}_N)$, it holds that $\boldsymbol{\theta}_{\text{MAP}} = \boldsymbol{m}_N$.

$$\boldsymbol{S}_N^{-1} = \boldsymbol{\Phi}^\top\sigma^{-2}\boldsymbol{I}\boldsymbol{\Phi} + \boldsymbol{S}_0^{-1} \tag{9.49a}$$

$$\iff \boldsymbol{S}_N = (\sigma^{-2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \boldsymbol{S}_0^{-1})^{-1} \tag{9.49b}$$

and

$$\boldsymbol{m}_N^\top\boldsymbol{S}_N^{-1} = (\sigma^{-2}\boldsymbol{\Phi}^\top\boldsymbol{y} + \boldsymbol{S}_0^{-1}\boldsymbol{m}_0)^\top \tag{9.50a}$$

$$\iff \boldsymbol{m}_N = \boldsymbol{S}_N(\sigma^{-2}\boldsymbol{\Phi}^\top\boldsymbol{y} + \boldsymbol{S}_0^{-1}\boldsymbol{m}_0)\,. \tag{9.50b}$$

□

Remark (General Approach to Completing the Squares). If we are given an equation

$$\boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x} - 2\boldsymbol{a}^\top \boldsymbol{x} + \text{const}_1\,, \tag{9.51}$$

where $\boldsymbol{A}$ is symmetric and positive definite, which we wish to bring into the form

$$(\boldsymbol{x} - \boldsymbol{\mu})^\top \boldsymbol{\Sigma}(\boldsymbol{x} - \boldsymbol{\mu}) + \text{const}_2\,, \tag{9.52}$$

we can do this by setting

$$\boldsymbol{\Sigma} := \boldsymbol{A}\,, \tag{9.53}$$

$$\boldsymbol{\mu} := \boldsymbol{\Sigma}^{-1}\boldsymbol{a} \tag{9.54}$$

and $\text{const}_2 = \text{const}_1 - \boldsymbol{\mu}^\top \boldsymbol{\Sigma}\boldsymbol{\mu}$. ♢

We can see that the terms inside the exponential in (9.47b) are of the form (9.51) with

$$\boldsymbol{A} := \sigma^{-2}\boldsymbol{\Phi}^\top \boldsymbol{\Phi} + \boldsymbol{S}_0^{-1}\,, \tag{9.55}$$

$$\boldsymbol{a} := \sigma^{-2}\boldsymbol{\Phi}^\top \boldsymbol{y} + \boldsymbol{S}_0^{-1}\boldsymbol{m}_0\,. \tag{9.56}$$

Since $\boldsymbol{A}, \boldsymbol{a}$ can be difficult to identify in equations like (9.46a), it is often helpful to bring these equations into the form (9.51) that decouples quadratic term, linear terms, and constants, which simplifies finding the desired solution.

9.3.4 Posterior Predictions

In (9.37), we computed the predictive distribution of y_* at a test input $\boldsymbol{x}_*$ using the parameter prior $p(\boldsymbol{\theta})$. In principle, predicting with the parameter posterior $p(\boldsymbol{\theta} \mid \mathcal{X}, \mathcal{Y})$ is not fundamentally different given that in our conjugate model the prior and posterior are both Gaussian (with different parameters). Therefore, by following the same reasoning as in Section 9.3.2, we obtain the (posterior) predictive distribution

$$p(y_* \mid \mathcal{X}, \mathcal{Y}, \boldsymbol{x}_*) = \int p(y_* \mid \boldsymbol{x}_*, \boldsymbol{\theta}) p(\boldsymbol{\theta} \mid \mathcal{X}, \mathcal{Y}) \mathrm{d}\boldsymbol{\theta} \tag{9.57a}$$

$$= \int \mathcal{N}\big(y_* \mid \boldsymbol{\phi}^\top(\boldsymbol{x}_*)\boldsymbol{\theta},\, \sigma^2\big)\mathcal{N}\big(\boldsymbol{\theta} \mid \boldsymbol{m}_N,\, \boldsymbol{S}_N\big)\mathrm{d}\boldsymbol{\theta} \tag{9.57b}$$

$$= \mathcal{N}\big(y_* \mid \boldsymbol{\phi}^\top(\boldsymbol{x}_*)\boldsymbol{m}_N,\, \boldsymbol{\phi}^\top(\boldsymbol{x}_*)\boldsymbol{S}_N\boldsymbol{\phi}(\boldsymbol{x}_*) + \sigma^2\big)\,. \tag{9.57c}$$

$\mathbb{E}[y_* \mid \mathcal{X}, \mathcal{Y}, \boldsymbol{x}_*] = \boldsymbol{\phi}^\top(\boldsymbol{x}_*)\boldsymbol{m}_N = \boldsymbol{\phi}^\top(\boldsymbol{x}_*)\boldsymbol{\theta}_{\text{MAP}}$.

The term $\boldsymbol{\phi}^\top(\boldsymbol{x}_*)\boldsymbol{S}_N\boldsymbol{\phi}(\boldsymbol{x}_*)$ reflects the posterior uncertainty associated with the parameters $\boldsymbol{\theta}$. Note that $\boldsymbol{S}_N$ depends on the training inputs through $\boldsymbol{\Phi}$; see (9.43b). The predictive mean $\boldsymbol{\phi}^\top(\boldsymbol{x}_*)\boldsymbol{m}_N$ coincides with the predictions made with the MAP estimate $\boldsymbol{\theta}_{\text{MAP}}$.

Remark (Marginal Likelihood and Posterior Predictive Distribution). By replacing the integral in (9.57a), the predictive distribution can be equivalently written as the expectation $\mathbb{E}_{\boldsymbol{\theta}\,|\,\mathcal{X},\mathcal{Y}}[p(y_*\,|\,\boldsymbol{x}_*,\boldsymbol{\theta})]$, where the expectation is taken with respect to the parameter posterior $p(\boldsymbol{\theta}\,|\,\mathcal{X},\mathcal{Y})$.

Writing the posterior predictive distribution in this way highlights a close resemblance to the marginal likelihood (9.42). The key difference between the marginal likelihood and the posterior predictive distribution are (i) the marginal likelihood can be thought of predicting the training targets $\boldsymbol{y}$ and not the test targets y_*, and (ii) the marginal likelihood averages with respect to the parameter prior and not the parameter posterior. ♢

Remark (Mean and Variance of Noise-Free Function Values). In many cases, we are not interested in the predictive distribution $p(y_*\,|\,\mathcal{X},\mathcal{Y},\boldsymbol{x}_*)$ of a (noisy) observation y_*. Instead, we would like to obtain the distribution of the (noise-free) function values $f(\boldsymbol{x}_*) = \boldsymbol{\phi}^\top(\boldsymbol{x}_*)\boldsymbol{\theta}$. We determine the corresponding moments by exploiting the properties of means and variances, which yields

$$\begin{aligned}\mathbb{E}[f(\boldsymbol{x}_*)\,|\,\mathcal{X},\mathcal{Y}] &= \mathbb{E}_{\boldsymbol{\theta}}[\boldsymbol{\phi}^\top(\boldsymbol{x}_*)\boldsymbol{\theta}\,|\,\mathcal{X},\mathcal{Y}] = \boldsymbol{\phi}^\top(\boldsymbol{x}_*)\mathbb{E}_{\boldsymbol{\theta}}[\boldsymbol{\theta}\,|\,\mathcal{X},\mathcal{Y}] \\ &= \boldsymbol{\phi}^\top(\boldsymbol{x}_*)\boldsymbol{m}_N = \boldsymbol{m}_N^\top\boldsymbol{\phi}(\boldsymbol{x}_*)\,, \end{aligned} \tag{9.58}$$

$$\begin{aligned}\mathbb{V}_{\boldsymbol{\theta}}[f(\boldsymbol{x}_*)\,|\,\mathcal{X},\mathcal{Y}] &= \mathbb{V}_{\boldsymbol{\theta}}[\boldsymbol{\phi}^\top(\boldsymbol{x}_*)\boldsymbol{\theta}\,|\,\mathcal{X},\mathcal{Y}] \\ &= \boldsymbol{\phi}^\top(\boldsymbol{x}_*)\mathbb{V}_{\boldsymbol{\theta}}[\boldsymbol{\theta}\,|\,\mathcal{X},\mathcal{Y}]\boldsymbol{\phi}(\boldsymbol{x}_*) \\ &= \boldsymbol{\phi}^\top(\boldsymbol{x}_*)\boldsymbol{S}_N\boldsymbol{\phi}(\boldsymbol{x}_*)\,. \end{aligned} \tag{9.59}$$

We see that the predictive mean is the same as the predictive mean for noisy observations as the noise has mean 0, and the predictive variance only differs by σ^2, which is the variance of the measurement noise: When we predict noisy function values, we need to include σ^2 as a source of uncertainty, but this term is not needed for noise-free predictions. Here, the only remaining uncertainty stems from the parameter posterior. ♢

Integrating out parameters induces a distribution over functions.

Remark (Distribution over Functions). The fact that we integrate out the parameters $\boldsymbol{\theta}$ induces a distribution over functions: If we sample $\boldsymbol{\theta}_i \sim p(\boldsymbol{\theta}\,|\,\mathcal{X},\mathcal{Y})$ from the parameter posterior, we obtain a single function realization $\boldsymbol{\theta}_i^\top\boldsymbol{\phi}(\cdot)$. The *mean function*, i.e., the set of all expected function values $\mathbb{E}_{\boldsymbol{\theta}}[f(\cdot)\,|\,\boldsymbol{\theta},\mathcal{X},\mathcal{Y}]$, of this distribution over functions is $\boldsymbol{m}_N^\top\boldsymbol{\phi}(\cdot)$. The (marginal) variance, i.e., the variance of the function $f(\cdot)$, is given by $\boldsymbol{\phi}^\top(\cdot)\boldsymbol{S}_N\boldsymbol{\phi}(\cdot)$. ♢

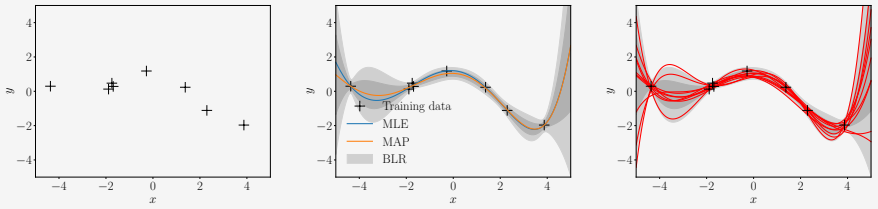
mean function

Example 9.8 (Posterior over Functions)

Let us revisit the Bayesian linear regression problem with polynomials of degree 5. We choose a parameter prior $p(\boldsymbol{\theta}) = \mathcal{N}(\mathbf{0}, \frac{1}{4}\boldsymbol{I})$. Figure 9.9 visualizes the prior over functions induced by the parameter prior and sample functions from this prior.

Figure 9.10 shows the posterior over functions that we obtain via Bayesian linear regression. The training dataset is shown in panel (a); panel (b) shows the posterior distribution over functions, including the functions we would obtain via maximum likelihood and MAP estimation. The function we obtain using the MAP estimate also corresponds to the posterior mean function in the Bayesian linear regression setting. Panel (c) shows some plausible realizations (samples) of functions under that posterior over functions.

Figure 9.10 Bayesian linear regression and posterior over functions. (a) training data; (b) posterior distribution over functions; (c) Samples from the posterior over functions.



(a) Training data.

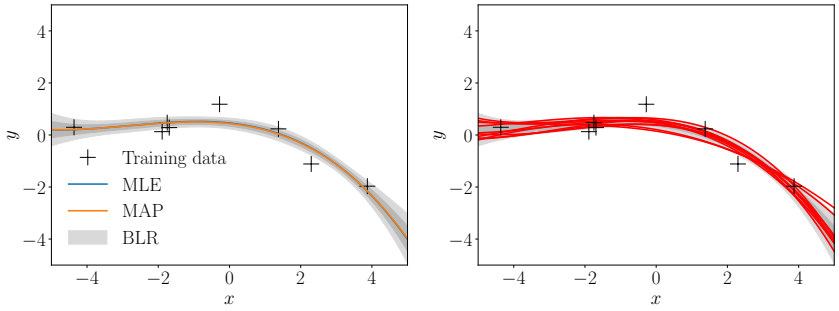
(b) Posterior over functions represented by the marginal uncertainties (shaded) showing the 67% and 95% predictive confidence bounds, the maximum likelihood estimate (MLE) and the MAP estimate (MAP), the latter of which is identical to the posterior mean function.

(c) Samples from the posterior over functions, which are induced by the samples from the parameter posterior.

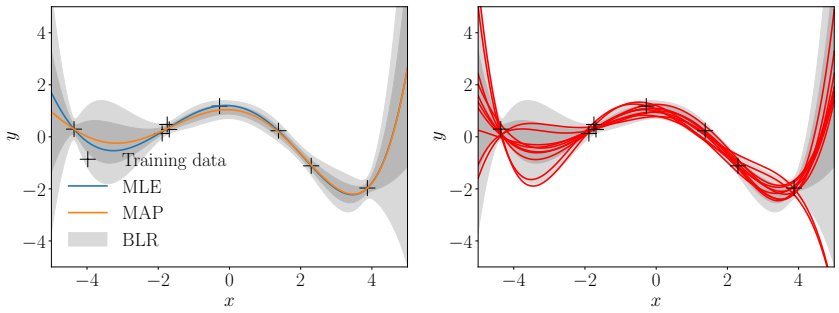
Figure 9.11 shows some posterior distributions over functions induced by the parameter posterior. For different polynomial degrees M, the left panels show the maximum likelihood function $\boldsymbol{\theta}_{\text{ML}}^\top \boldsymbol{\phi}(\cdot)$, the MAP function $\boldsymbol{\theta}_{\text{MAP}}^\top \boldsymbol{\phi}(\cdot)$ (which is identical to the posterior mean function), and the 67% and 95% predictive confidence bounds obtained by Bayesian linear regression, represented by the shaded areas.

The right panels show samples from the posterior over functions: Here, we sampled parameters $\boldsymbol{\theta}_i$ from the parameter posterior and computed the function $\boldsymbol{\phi}^\top(\boldsymbol{x}_*)\boldsymbol{\theta}_i$, which is a single realization of a function under the posterior distribution over functions. For low-order polynomials, the parameter posterior does not allow the parameters to vary much: The sampled functions are nearly identical. When we make the model more flexible by adding more parameters (i.e., we end up with a higher-order polynomial), these parameters are not sufficiently constrained by the posterior, and the sampled functions can be easily visually separated. We also see in the corresponding panels on the left how the uncertainty increases, especially at the boundaries.

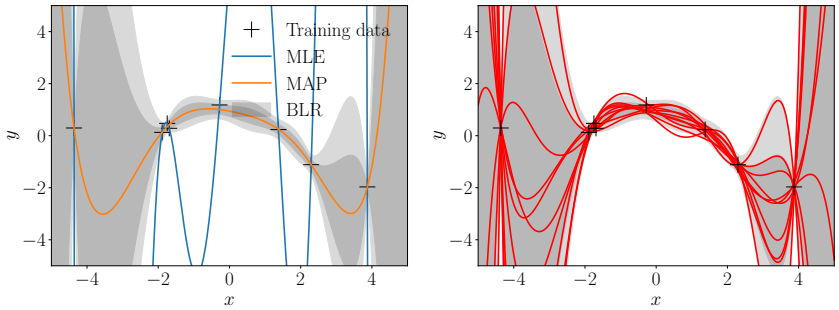
Although for a seventh-order polynomial the MAP estimate yields a reasonable fit, the Bayesian linear regression model additionally tells us that



(a) Posterior distribution for polynomials of degree $M = 3$ (left) and samples from the posterior over functions (right).



(b) Posterior distribution for polynomials of degree $M = 5$ (left) and samples from the posterior over functions (right).



(c) Posterior distribution for polynomials of degree $M = 7$ (left) and samples from the posterior over functions (right).

Figure 9.11 Bayesian linear regression. Left panels: Shaded areas indicate the 67% (dark gray) and 95% (light gray) predictive confidence bounds. The mean of the Bayesian linear regression model coincides with the MAP estimate. The predictive uncertainty is the sum of the noise term and the posterior parameter uncertainty, which depends on the location of the test input. Right panels: sampled functions from the posterior distribution.

the posterior uncertainty is huge. This information can be critical when we use these predictions in a decision-making system, where bad decisions can have significant consequences (e.g., in reinforcement learning or robotics).

9.3.5 Computing the Marginal Likelihood

In Section 8.6.2, we highlighted the importance of the marginal likelihood for Bayesian model selection. In the following, we compute the marginal likelihood for Bayesian linear regression with a conjugate Gaussian prior on the parameters, i.e., exactly the setting we have been discussing in this chapter.

Just to recap, we consider the following generative process:

$$\boldsymbol{\theta} \sim \mathcal{N}(\boldsymbol{m}_0, \boldsymbol{S}_0) \tag{9.60a}$$

$$y_n \,|\, \boldsymbol{x}_n, \boldsymbol{\theta} \sim \mathcal{N}(\boldsymbol{x}_n^\top \boldsymbol{\theta}, \sigma^2)\,, \tag{9.60b}$$

$n = 1, \ldots, N$. The marginal likelihood is given by

The marginal likelihood can be interpreted as the expected likelihood under the prior, i.e., $\mathbb{E}_{\boldsymbol{\theta}}[p(\mathcal{Y} \,|\, \mathcal{X}, \boldsymbol{\theta})]$.

$$p(\mathcal{Y} \,|\, \mathcal{X}) = \int p(\mathcal{Y} \,|\, \mathcal{X}, \boldsymbol{\theta}) p(\boldsymbol{\theta}) \mathrm{d}\boldsymbol{\theta} \tag{9.61a}$$

$$= \int \mathcal{N}(\boldsymbol{y} \,|\, \boldsymbol{X}\boldsymbol{\theta}, \sigma^2 \boldsymbol{I}) \mathcal{N}(\boldsymbol{\theta} \,|\, \boldsymbol{m}_0, \boldsymbol{S}_0) \mathrm{d}\boldsymbol{\theta}\,, \tag{9.61b}$$

where we integrate out the model parameters $\boldsymbol{\theta}$. We compute the marginal likelihood in two steps: First, we show that the marginal likelihood is Gaussian (as a distribution in $\boldsymbol{y}$); second, we compute the mean and covariance of this Gaussian.

1. The marginal likelihood is Gaussian: From Section 6.5.2, we know that (i) the product of two Gaussian random variables is an (unnormalized) Gaussian distribution, and (ii) a linear transformation of a Gaussian random variable is Gaussian distributed. In (9.61b), we require a linear transformation to bring $\mathcal{N}(\boldsymbol{y} \,|\, \boldsymbol{X}\boldsymbol{\theta}, \sigma^2 \boldsymbol{I})$ into the form $\mathcal{N}(\boldsymbol{\theta} \,|\, \boldsymbol{\mu}, \boldsymbol{\Sigma})$ for some $\boldsymbol{\mu}, \boldsymbol{\Sigma}$. Once this is done, the integral can be solved in closed form. The result is the normalizing constant of the product of the two Gaussians. The normalizing constant itself has Gaussian shape; see (6.76).
2. Mean and covariance. We compute the mean and covariance matrix of the marginal likelihood by exploiting the standard results for means and covariances of affine transformations of random variables; see Section 6.4.4. The mean of the marginal likelihood is computed as

$$\mathbb{E}[\mathcal{Y} \,|\, \mathcal{X}] = \mathbb{E}_{\boldsymbol{\theta}, \boldsymbol{\epsilon}}[\boldsymbol{X}\boldsymbol{\theta} + \boldsymbol{\epsilon}] = \boldsymbol{X}\mathbb{E}_{\boldsymbol{\theta}}[\boldsymbol{\theta}] = \boldsymbol{X}\boldsymbol{m}_0\,. \tag{9.62}$$

   Note that $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}, \sigma^2 \boldsymbol{I})$ is a vector of i.i.d. random variables. The covariance matrix is given as

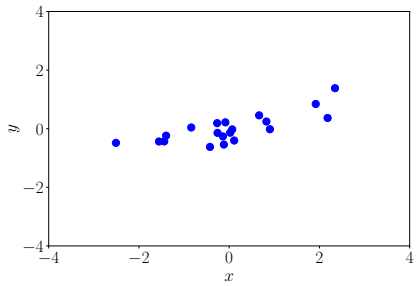
$$\mathrm{Cov}[\mathcal{Y}|\mathcal{X}] = \mathrm{Cov}_{\boldsymbol{\theta}, \boldsymbol{\epsilon}}[\boldsymbol{X}\boldsymbol{\theta} + \boldsymbol{\epsilon}] = \mathrm{Cov}_{\boldsymbol{\theta}}[\boldsymbol{X}\boldsymbol{\theta}] + \sigma^2 \boldsymbol{I} \tag{9.63a}$$

$$= \boldsymbol{X}\,\mathrm{Cov}_{\boldsymbol{\theta}}[\boldsymbol{\theta}]\boldsymbol{X}^\top + \sigma^2 \boldsymbol{I} = \boldsymbol{X}\boldsymbol{S}_0\boldsymbol{X}^\top + \sigma^2 \boldsymbol{I}\,. \tag{9.63b}$$

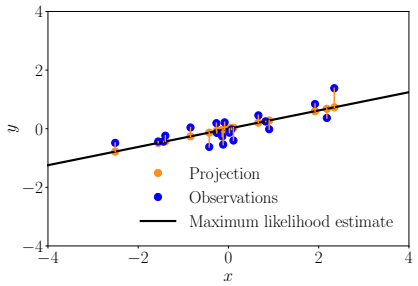
Hence, the marginal likelihood is

$$p(\mathcal{Y} \,|\, \mathcal{X}) = (2\pi)^{-\frac{N}{2}} \det(\boldsymbol{X}\boldsymbol{S}_0\boldsymbol{X}^\top + \sigma^2 \boldsymbol{I})^{-\frac{1}{2}} \tag{9.64a}$$

$$\cdot \exp\left(-\tfrac{1}{2}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{m}_0)^\top (\boldsymbol{X}\boldsymbol{S}_0\boldsymbol{X}^\top + \sigma^2 \boldsymbol{I})^{-1}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{m}_0)\right)$$



(a) Regression dataset consisting of noisy observations y_n (blue) of function values $f(x_n)$ at input locations x_n.



(b) The orange dots are the projections of the noisy observations (blue dots) onto the line $\theta_{\text{ML}}x$. The maximum likelihood solution to a linear regression problem finds a subspace (line) onto which the overall projection error (orange lines) of the observations is minimized.

Figure 9.12 Geometric interpretation of least squares. (a) Dataset; (b) maximum likelihood solution interpreted as a projection.

$$= \mathcal{N}\left(\boldsymbol{y} \,|\, \boldsymbol{X}\boldsymbol{m}_0,\, \boldsymbol{X}\boldsymbol{S}_0\boldsymbol{X}^\top + \sigma^2\boldsymbol{I}\right). \tag{9.64b}$$

Given the close connection with the posterior predictive distribution (see Remark on Marginal Likelihood and Posterior Predictive Distribution earlier in this section), the functional form of the marginal likelihood should not be too surprising.

9.4 Maximum Likelihood as Orthogonal Projection

Having crunched through much algebra to derive maximum likelihood and MAP estimates, we will now provide a geometric interpretation of maximum likelihood estimation. Let us consider a simple linear regression setting

$$y = x\theta + \epsilon, \quad \epsilon \sim \mathcal{N}\left(0,\, \sigma^2\right), \tag{9.65}$$

in which we consider linear functions $f : \mathbb{R} \to \mathbb{R}$ that go through the origin (we omit features here for clarity). The parameter θ determines the slope of the line. Figure 9.12(a) shows a one-dimensional dataset.

With a training data set $\{(x_1, y_1), \ldots, (x_N, y_N)\}$ we recall the results from Section 9.2.1 and obtain the maximum likelihood estimator for the slope parameter as

$$\theta_{\text{ML}} = (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y} = \frac{\boldsymbol{X}^\top\boldsymbol{y}}{\boldsymbol{X}^\top\boldsymbol{X}} \in \mathbb{R}, \tag{9.66}$$

where $\boldsymbol{X} = [x_1, \ldots, x_N]^\top \in \mathbb{R}^N$, $\boldsymbol{y} = [y_1, \ldots, y_N]^\top \in \mathbb{R}^N$.

This means for the training inputs $\boldsymbol{X}$ we obtain the optimal (maximum likelihood) reconstruction of the training targets as

$$\boldsymbol{X}\theta_{\text{ML}} = \boldsymbol{X}\frac{\boldsymbol{X}^\top\boldsymbol{y}}{\boldsymbol{X}^\top\boldsymbol{X}} = \frac{\boldsymbol{X}\boldsymbol{X}^\top}{\boldsymbol{X}^\top\boldsymbol{X}}\boldsymbol{y}, \tag{9.67}$$

i.e., we obtain the approximation with the minimum least-squares error between $\boldsymbol{y}$ and $\boldsymbol{X}\theta$.

Linear regression can be thought of as a method for solving systems of linear equations.

As we are looking for a solution of $\boldsymbol{y} = \boldsymbol{X}\theta$, we can think of linear regression as a problem for solving systems of linear equations. Therefore, we can relate to concepts from linear algebra and analytic geometry that we discussed in Chapters 2 and 3. In particular, looking carefully at (9.67) we see that the maximum likelihood estimator θ_{ML} in our example from (9.65) effectively does an orthogonal projection of $\boldsymbol{y}$ onto the one-dimensional subspace spanned by $\boldsymbol{X}$. Recalling the results on orthogonal projections from Section 3.8, we identify $\frac{\boldsymbol{X}\boldsymbol{X}^\top}{\boldsymbol{X}^\top\boldsymbol{X}}$ as the projection matrix, θ_{ML} as the coordinates of the projection onto the one-dimensional subspace of $\mathbb{R}^N$ spanned by $\boldsymbol{X}$ and $\boldsymbol{X}\theta_{\text{ML}}$ as the orthogonal projection of $\boldsymbol{y}$ onto this subspace.

Maximum likelihood linear regression performs an orthogonal projection.

Therefore, the maximum likelihood solution provides also a geometrically optimal solution by finding the vectors in the subspace spanned by $\boldsymbol{X}$ that are "closest" to the corresponding observations $\boldsymbol{y}$, where "closest" means the smallest (squared) distance of the function values y_n to $x_n\theta$. This is achieved by orthogonal projections. Figure 9.12(b) shows the projection of the noisy observations onto the subspace that minimizes the squared distance between the original dataset and its projection (note that the x-coordinate is fixed), which corresponds to the maximum likelihood solution.

In the general linear regression case where

$$y = \boldsymbol{\phi}^\top(\boldsymbol{x})\boldsymbol{\theta} + \epsilon, \quad \epsilon \sim \mathcal{N}\left(0,\, \sigma^2\right) \tag{9.68}$$

with vector-valued features $\boldsymbol{\phi}(\boldsymbol{x}) \in \mathbb{R}^K$, we again can interpret the maximum likelihood result

$$\boldsymbol{y} \approx \boldsymbol{\Phi}\boldsymbol{\theta}_{\text{ML}}\,, \tag{9.69}$$

$$\boldsymbol{\theta}_{\text{ML}} = (\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\boldsymbol{y} \tag{9.70}$$

as a projection onto a K-dimensional subspace of $\mathbb{R}^N$, which is spanned by the columns of the feature matrix $\boldsymbol{\Phi}$; see Section 3.8.2.

If the feature functions ϕ_k that we use to construct the feature matrix $\boldsymbol{\Phi}$ are orthonormal (see Section 3.7), we obtain a special case where the columns of $\boldsymbol{\Phi}$ form an orthonormal basis (see Section 3.5), such that $\boldsymbol{\Phi}^\top\boldsymbol{\Phi} = \boldsymbol{I}$. This will then lead to the projection

$$\boldsymbol{\Phi}(\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\boldsymbol{y} = \boldsymbol{\Phi}\boldsymbol{\Phi}^\top\boldsymbol{y} = \left(\sum_{k=1}^{K}\boldsymbol{\phi}_k\boldsymbol{\phi}_k^\top\right)\boldsymbol{y} \tag{9.71}$$

so that the maximum likelihood projection is simply the sum of projections of $\boldsymbol{y}$ onto the individual basis vectors $\boldsymbol{\phi}_k$, i.e., the columns of $\boldsymbol{\Phi}$. Furthermore, the coupling between different features has disappeared due to the orthogonality of the basis. Many popular basis functions in signal processing, such as wavelets and Fourier bases, are orthogonal basis functions.

When the basis is not orthogonal, one can convert a set of linearly independent basis functions to an orthogonal basis by using the Gram-Schmidt process; see Section 3.8.3 and (Strang, 2003).

9.5 Further Reading

In this chapter, we discussed linear regression for Gaussian likelihoods and conjugate Gaussian priors on the parameters of the model. This allowed for closed-form Bayesian inference. However, in some applications we may want to choose a different likelihood function. For example, in a binary *classification* setting, we observe only two possible (categorical) outcomes, and a Gaussian likelihood is inappropriate in this setting. Instead, we can choose a Bernoulli likelihood that will return a probability of the predicted label to be 1 (or 0). We refer to the books by Barber (2012), Bishop (2006), and Murphy (2012) for an in-depth introduction to classification problems. A different example where non-Gaussian likelihoods are important is count data. Counts are non-negative integers, and in this case a Binomial or Poisson likelihood would be a better choice than a Gaussian. All these examples fall into the category of *generalized linear models*, a flexible generalization of linear regression that allows for response variables that have error distributions other than a Gaussian distribution. The GLM generalizes linear regression by allowing the linear model to be related to the observed values via a smooth and invertible function $\sigma(\cdot)$ that may be nonlinear so that $y = \sigma(f(\boldsymbol{x}))$, where $f(\boldsymbol{x}) = \boldsymbol{\theta}^\top \boldsymbol{\phi}(\boldsymbol{x})$ is the linear regression model from (9.13). We can therefore think of a generalized linear model in terms of function composition $y = \sigma \circ f$, where f is a linear regression model and σ the activation function. Note that although we are talking about "generalized linear models", the outputs y are no longer linear in the parameters $\boldsymbol{\theta}$. In *logistic regression*, we choose the *logistic sigmoid* $\sigma(f) = \frac{1}{1+\exp(-f)} \in [0, 1]$, which can be interpreted as the probability of observing $y = 1$ of a Bernoulli random variable $y \in \{0, 1\}$. The function $\sigma(\cdot)$ is called *transfer function* or *activation function*, and its inverse is called the *canonical link function*. From this perspective, it is also clear that generalized linear models are the building blocks of (deep) feedforward neural networks: If we consider a generalized linear model $\boldsymbol{y} = \sigma(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b})$, where $\boldsymbol{A}$ is a weight matrix and $\boldsymbol{b}$ a bias vector, we identify this generalized linear model as a single-layer neural network with activation function $\sigma(\cdot)$. We can now recursively compose these functions via

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \boldsymbol{f}_k(\boldsymbol{x}_k) \\ \boldsymbol{f}_k(\boldsymbol{x}_k) &= \sigma_k(\boldsymbol{A}_k \boldsymbol{x}_k + \boldsymbol{b}_k) \end{aligned} \tag{9.72}$$

for $k = 0, \ldots, K-1$, where $\boldsymbol{x}_0$ are the input features and $\boldsymbol{x}_K = \boldsymbol{y}$ are the observed outputs, such that $\boldsymbol{f}_{K-1} \circ \cdots \circ \boldsymbol{f}_0$ is a K-layer deep neural network. Therefore, the building blocks of this deep neural network are

classification

generalized linear model

Generalized linear models are the building blocks of deep neural networks.

logistic regression

logistic sigmoid

transfer function

activation function

canonical link function

For ordinary linear regression the activation function would simply be the identity.

A great post on the relation between GLMs and deep networks is available at `https://tinyurl.com/glm-dnn`.

the generalized linear models defined in (9.72). Neural networks (Bishop, 1995; Goodfellow et al., 2016) are significantly more expressive and flexible than linear regression models. However, maximum likelihood parameter estimation is a non-convex optimization problem, and marginalization of the parameters in a fully Bayesian setting is analytically intractable.

Gaussian process

We briefly hinted at the fact that a distribution over parameters induces a distribution over regression functions. *Gaussian processes* (Rasmussen and Williams, 2006) are regression models where the concept of a distribution over function is central. Instead of placing a distribution over parameters, a Gaussian process places a distribution directly on the space of functions without the "detour" via the parameters. To do so, the Gaussian process exploits the *kernel trick* (Schölkopf and Smola, 2002), which allows us to compute inner products between two function values $f(\boldsymbol{x}_i), f(\boldsymbol{x}_j)$ only by looking at the corresponding input $\boldsymbol{x}_i, \boldsymbol{x}_j$. A Gaussian process is closely related to both Bayesian linear regression and support vector regression but can also be interpreted as a Bayesian neural network with a single hidden layer where the number of units tends to infinity (Neal, 1996; Williams, 1997). Excellent introductions to Gaussian processes can be found in MacKay (1998) and Rasmussen and Williams (2006).

kernel trick

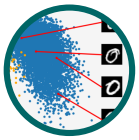
We focused on Gaussian parameter priors in the discussions in this chapter, because they allow for closed-form inference in linear regression models. However, even in a regression setting with Gaussian likelihoods, we may choose a non-Gaussian prior. Consider a setting, where the inputs are $\boldsymbol{x} \in \mathbb{R}^D$ and our training set is small and of size $N \ll D$. This means that the regression problem is underdetermined. In this case, we can choose a parameter prior that enforces sparsity, i.e., a prior that tries to set as many parameters to 0 as possible (*variable selection*). This prior provides a stronger regularizer than the Gaussian prior, which often leads to an increased prediction accuracy and interpretability of the model. The Laplace prior is one example that is frequently used for this purpose. A linear regression model with the Laplace prior on the parameters is equivalent to linear regression with L1 regularization (*LASSO*) (Tibshirani, 1996). The Laplace distribution is sharply peaked at zero (its first derivative is discontinuous) and it concentrates its probability mass closer to zero than the Gaussian distribution, which encourages parameters to be 0. Therefore, the nonzero parameters are relevant for the regression problem, which is the reason why we also speak of "variable selection".

variable selection

LASSO

10

Dimensionality Reduction with Principal Component Analysis

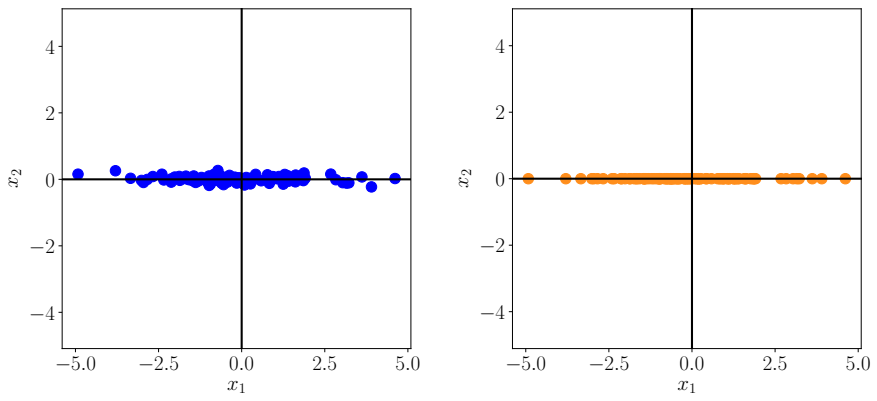


A 640×480 pixel color image is a data point in a million-dimensional space, where every pixel responds to three dimensions, one for each color channel (red, green, blue).

Working directly with high-dimensional data, such as images, comes with some difficulties: It is hard to analyze, interpretation is difficult, visualization is nearly impossible, and (from a practical point of view) storage of the data vectors can be expensive. However, high-dimensional data often has properties that we can exploit. For example, high-dimensional data is often overcomplete, i.e., many dimensions are redundant and can be explained by a combination of other dimensions. Furthermore, dimensions in high-dimensional data are often correlated so that the data possesses an intrinsic lower-dimensional structure. Dimensionality reduction exploits structure and correlation and allows us to work with a more compact representation of the data, ideally without losing information. We can think of dimensionality reduction as a compression technique, similar to jpeg or mp3, which are compression algorithms for images and music.

In this chapter, we will discuss *principal component analysis* (*PCA*), an algorithm for linear *dimensionality reduction*. PCA, proposed by Pearson (1901) and Hotelling (1933), has been around for more than 100 years and is still one of the most commonly used techniques for data compression and data visualization. It is also used for the identification of simple patterns, latent factors, and structures of high-dimensional data. In the

principal component analysis
PCA
dimensionality reduction



(a) Dataset with x_1 and x_2 coordinates.

(b) Compressed dataset where only the x_1 coordinate is relevant.

Figure 10.1 Illustration: dimensionality reduction. (a) The original dataset does not vary much along the x_2 direction. (b) The data from (a) can be represented using the x_1-coordinate alone with nearly no loss.

Karhunen-Loève transform

signal processing community, PCA is also known as the *Karhunen-Loève transform*. In this chapter, we derive PCA from first principles, drawing on our understanding of basis and basis change (Sections 2.6.1 and 2.7.2), projections (Section 3.8), eigenvalues (Section 4.2), Gaussian distributions (Section 6.5), and constrained optimization (Section 7.2).

Dimensionality reduction generally exploits a property of high-dimensional data (e.g., images) that it often lies on a low-dimensional subspace. Figure 10.1 gives an illustrative example in two dimensions. Although the data in Figure 10.1(a) does not quite lie on a line, the data does not vary much in the x_2-direction, so that we can express it as if it were on a line – with nearly no loss; see Figure 10.1(b). To describe the data in Figure 10.1(b), only the x_1-coordinate is required, and the data lies in a one-dimensional subspace of $\mathbb{R}^2$.

10.1 Problem Setting

In PCA, we are interested in finding projections $\tilde{\boldsymbol{x}}_n$ of data points $\boldsymbol{x}_n$ that are as similar to the original data points as possible, but which have a significantly lower intrinsic dimensionality. Figure 10.1 gives an illustration of what this could look like.

More concretely, we consider an i.i.d. dataset $\mathcal{X} = \{\boldsymbol{x}_1, \dots, \boldsymbol{x}_N\}$, $\boldsymbol{x}_n \in \mathbb{R}^D$, with mean $\mathbf{0}$ that possesses the *data covariance matrix* (6.42)

data covariance matrix

$$\boldsymbol{S} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n \boldsymbol{x}_n^\top . \tag{10.1}$$

Furthermore, we assume there exists a low-dimensional compressed representation (code)

$$\boldsymbol{z}_n = \boldsymbol{B}^\top \boldsymbol{x}_n \in \mathbb{R}^M \tag{10.2}$$

of $\boldsymbol{x}_n$, where we define the projection matrix

$$\boldsymbol{B} := [\boldsymbol{b}_1, \dots, \boldsymbol{b}_M] \in \mathbb{R}^{D \times M} . \tag{10.3}$$

The columns $\boldsymbol{b}_1, \dots, \boldsymbol{b}_M$ of $\boldsymbol{B}$ form a basis of the M-dimensional subspace in which the projected data $\tilde{\boldsymbol{x}} = \boldsymbol{B}\boldsymbol{B}^\top \boldsymbol{x} \in \mathbb{R}^D$ live.

We assume that the columns of $\boldsymbol{B}$ are orthonormal (Definition 3.7) so that $\boldsymbol{b}_i^\top \boldsymbol{b}_j = 0$ if and only if $i \neq j$ and $\boldsymbol{b}_i^\top \boldsymbol{b}_i = 1$. We seek an M-dimensional subspace $U \subseteq \mathbb{R}^D$, $\dim(U) = M < D$ onto which we project the data. We denote the projected data by $\tilde{\boldsymbol{x}}_n \in U$, and their coordinates (with respect to the basis vectors $\boldsymbol{b}_1, \dots, \boldsymbol{b}_M$ of U) by $\boldsymbol{z}_n$. Our aim is to find projections $\tilde{\boldsymbol{x}}_n \in \mathbb{R}^D$ (or equivalently the codes $\boldsymbol{z}_n$ and the basis vectors $\boldsymbol{b}_1, \dots, \boldsymbol{b}_M$) so that they are as similar to the original data $\boldsymbol{x}_n$ and minimize the loss due to compression.

Example 10.1 (Coordinate Representation/Code)
Consider $\mathbb{R}^2$ with the canonical basis $\boldsymbol{e}_1 = [1, 0]^\top$, $\boldsymbol{e}_2 = [0, 1]^\top$. From

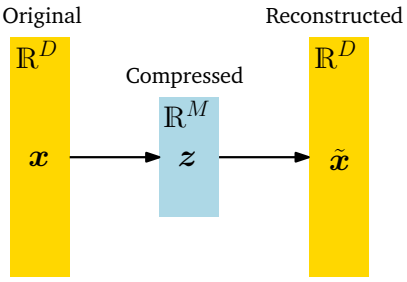


Figure 10.2 Graphical illustration of PCA. In PCA, we find a compressed version $\boldsymbol{z}$ of original data $\boldsymbol{x}$. The compressed data can be reconstructed into $\tilde{\boldsymbol{x}}$, which lives in the original data space, but has an intrinsic lower-dimensional representation than $\boldsymbol{x}$.

Chapter 2, we know that $\boldsymbol{x} \in \mathbb{R}^2$ can be represented as a linear combination of these basis vectors, e.g.,

$$\begin{bmatrix}5\\3\end{bmatrix} = 5\boldsymbol{e}_1 + 3\boldsymbol{e}_2\,. \tag{10.4}$$

However, when we consider vectors of the form

$$\tilde{\boldsymbol{x}} = \begin{bmatrix}0\\z\end{bmatrix} \in \mathbb{R}^2\,, \quad z \in \mathbb{R}\,, \tag{10.5}$$

they can always be written as $0\boldsymbol{e}_1 + z\boldsymbol{e}_2$. To represent these vectors it is sufficient to remember/store the *coordinate/code* z of $\tilde{\boldsymbol{x}}$ with respect to the $\boldsymbol{e}_2$ vector.

More precisely, the set of $\tilde{\boldsymbol{x}}$ vectors (with the standard vector addition and scalar multiplication) forms a vector subspace U (see Section 2.4) with $\dim(U) = 1$ because $U = \text{span}[\boldsymbol{e}_2]$.

The dimension of a vector space corresponds to the number of its basis vectors (see Section 2.6.1).

In Section 10.2, we will find low-dimensional representations that retain as much information as possible and minimize the compression loss. An alternative derivation of PCA is given in Section 10.3, where we will be looking at minimizing the squared reconstruction error $\|\boldsymbol{x}_n - \tilde{\boldsymbol{x}}_n\|^2$ between the original data $\boldsymbol{x}_n$ and its projection $\tilde{\boldsymbol{x}}_n$.

Figure 10.2 illustrates the setting we consider in PCA, where $\boldsymbol{z}$ represents the lower-dimensional representation of the compressed data $\tilde{\boldsymbol{x}}$ and plays the role of a bottleneck, which controls how much information can flow between $\boldsymbol{x}$ and $\tilde{\boldsymbol{x}}$. In PCA, we consider a linear relationship between the original data $\boldsymbol{x}$ and its low-dimensional code $\boldsymbol{z}$ so that $\boldsymbol{z} = \boldsymbol{B}^\top\boldsymbol{x}$ and $\tilde{\boldsymbol{x}} = \boldsymbol{B}\boldsymbol{z}$ for a suitable matrix $\boldsymbol{B}$. Based on the motivation of thinking of PCA as a data compression technique, we can interpret the arrows in Figure 10.2 as a pair of operations representing encoders and decoders. The linear mapping represented by $\boldsymbol{B}$ can be thought of as a decoder, which maps the low-dimensional code $\boldsymbol{z} \in \mathbb{R}^M$ back into the original data space $\mathbb{R}^D$. Similarly, $\boldsymbol{B}^\top$ can be thought of an encoder, which encodes the original data $\boldsymbol{x}$ as a low-dimensional (compressed) code $\boldsymbol{z}$.

Throughout this chapter, we will use the MNIST digits dataset as a re-

Figure 10.3 Examples of handwritten digits from the MNIST dataset. http://yann.lecun.com/exdb/mnist/.

occurring example, which contains $60{,}000$ examples of handwritten digits 0 through 9. Each digit is a grayscale image of size 28×28, i.e., it contains 784 pixels so that we can interpret every image in this dataset as a vector $\boldsymbol{x} \in \mathbb{R}^{784}$. Examples of these digits are shown in Figure 10.3.

10.2 Maximum Variance Perspective

Figure 10.1 gave an example of how a two-dimensional dataset can be represented using a single coordinate. In Figure 10.1(b), we chose to ignore the x_2-coordinate of the data because it did not add too much information so that the compressed data is similar to the original data in Figure 10.1(a). We could have chosen to ignore the x_1-coordinate, but then the compressed data had been very dissimilar from the original data, and much information in the data would have been lost.

If we interpret information content in the data as how "space filling" the dataset is, then we can describe the information contained in the data by looking at the spread of the data. From Section 6.4.1, we know that the variance is an indicator of the spread of the data, and we can derive PCA as a dimensionality reduction algorithm that maximizes the variance in the low-dimensional representation of the data to retain as much information as possible. Figure 10.4 illustrates this.

Considering the setting discussed in Section 10.1, our aim is to find a matrix $\boldsymbol{B}$ (see (10.3)) that retains as much information as possible when compressing data by projecting it onto the subspace spanned by the columns $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_M$ of $\boldsymbol{B}$. Retaining most information after data compression is equivalent to capturing the largest amount of variance in the low-dimensional code (Hotelling, 1933).

Remark. (Centered Data) For the data covariance matrix in (10.1), we assumed centered data. We can make this assumption without loss of generality: Let us assume that $\boldsymbol{\mu}$ is the mean of the data. Using the properties of the variance, which we discussed in Section 6.4.4, we obtain

$$\mathbb{V}_{\boldsymbol{z}}[\boldsymbol{z}] = \mathbb{V}_{\boldsymbol{x}}[\boldsymbol{B}^\top(\boldsymbol{x} - \boldsymbol{\mu})] = \mathbb{V}_{\boldsymbol{x}}[\boldsymbol{B}^\top\boldsymbol{x} - \boldsymbol{B}^\top\boldsymbol{\mu}] = \mathbb{V}_{\boldsymbol{x}}[\boldsymbol{B}^\top\boldsymbol{x}]\,, \tag{10.6}$$

i.e., the variance of the low-dimensional code does not depend on the mean of the data. Therefore, we assume without loss of generality that the data has mean $\mathbf{0}$ for the remainder of this section. With this assumption the mean of the low-dimensional code is also $\mathbf{0}$ since $\mathbb{E}_{\boldsymbol{z}}[\boldsymbol{z}] = \mathbb{E}_{\boldsymbol{x}}[\boldsymbol{B}^\top\boldsymbol{x}] = \boldsymbol{B}^\top\mathbb{E}_{\boldsymbol{x}}[\boldsymbol{x}] = \mathbf{0}$. ♢

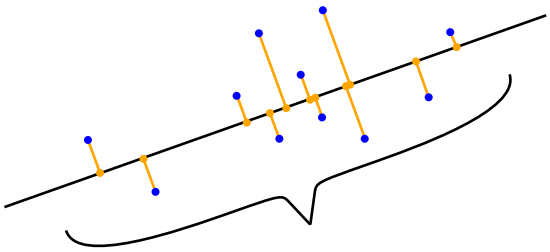


Figure 10.4 PCA finds a lower-dimensional subspace (line) that maintains as much variance (spread of the data) as possible when the data (blue) is projected onto this subspace (orange).

10.2.1 Direction with Maximal Variance

We maximize the variance of the low-dimensional code using a sequential approach. We start by seeking a single vector $\boldsymbol{b}_1 \in \mathbb{R}^D$ that maximizes the variance of the projected data, i.e., we aim to maximize the variance of the first coordinate z_1 of $\boldsymbol{z} \in \mathbb{R}^M$ so that

The vector $\boldsymbol{b}_1$ will be the first column of the matrix $\boldsymbol{B}$ and therefore the first of M orthonormal basis vectors that span the lower-dimensional subspace.

$$V_1 := \mathbb{V}[z_1] = \frac{1}{N}\sum_{n=1}^{N} z_{1n}^2 \tag{10.7}$$

is maximized, where we exploited the i.i.d. assumption of the data and defined z_{1n} as the first coordinate of the low-dimensional representation $\boldsymbol{z}_n \in \mathbb{R}^M$ of $\boldsymbol{x}_n \in \mathbb{R}^D$. Note that first component of $\boldsymbol{z}_n$ is given by

$$z_{1n} = \boldsymbol{b}_1^\top \boldsymbol{x}_n \,, \tag{10.8}$$

i.e., it is the coordinate of the orthogonal projection of $\boldsymbol{x}_n$ onto the one-dimensional subspace spanned by $\boldsymbol{b}_1$ (Section 3.8). We substitute (10.8) into (10.7), which yields

$$V_1 = \frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{b}_1^\top \boldsymbol{x}_n)^2 = \frac{1}{N}\sum_{n=1}^{N}\boldsymbol{b}_1^\top \boldsymbol{x}_n \boldsymbol{x}_n^\top \boldsymbol{b}_1 \tag{10.9a}$$

$$= \boldsymbol{b}_1^\top \left(\frac{1}{N}\sum_{n=1}^{N}\boldsymbol{x}_n \boldsymbol{x}_n^\top\right)\boldsymbol{b}_1 = \boldsymbol{b}_1^\top \boldsymbol{S}\boldsymbol{b}_1 \,, \tag{10.9b}$$

where $\boldsymbol{S}$ is the data covariance matrix defined in (10.1). In (10.9a), we have used the fact that the dot product of two vectors is symmetric with respect to its arguments, that is, $\boldsymbol{b}_1^\top \boldsymbol{x}_n = \boldsymbol{x}_n^\top \boldsymbol{b}_1$.

Notice that arbitrarily increasing the magnitude of the vector $\boldsymbol{b}_1$ increases V_1, that is, a vector $\boldsymbol{b}_1$ that is two times longer can result in V_1 that is potentially four times larger. Therefore, we restrict all solutions to $\|\boldsymbol{b}_1\|^2 = 1$, which results in a constrained optimization problem in which we seek the direction along which the data varies most.

$\|\boldsymbol{b}_1\|^2 = 1 \iff \|\boldsymbol{b}_1\| = 1.$

With the restriction of the solution space to unit vectors the vector $\boldsymbol{b}_1$ that points in the direction of maximum variance can be found by the

constrained optimization problem

$$\begin{aligned}&\max_{\boldsymbol{b}_1} \boldsymbol{b}_1^\top \boldsymbol{S} \boldsymbol{b}_1 \\ &\text{subject to } \|\boldsymbol{b}_1\|^2 = 1\,.\end{aligned} \tag{10.10}$$

Following Section 7.2, we obtain the Lagrangian

$$\mathfrak{L}(\boldsymbol{b}_1, \lambda) = \boldsymbol{b}_1^\top \boldsymbol{S} \boldsymbol{b}_1 + \lambda_1(1 - \boldsymbol{b}_1^\top \boldsymbol{b}_1) \tag{10.11}$$

to solve this constrained optimization problem. The partial derivatives of $\mathfrak{L}$ with respect to $\boldsymbol{b}_1$ and λ_1 are

$$\frac{\partial \mathfrak{L}}{\partial \boldsymbol{b}_1} = 2\boldsymbol{b}_1^\top \boldsymbol{S} - 2\lambda_1 \boldsymbol{b}_1^\top\,, \qquad \frac{\partial \mathfrak{L}}{\partial \lambda_1} = 1 - \boldsymbol{b}_1^\top \boldsymbol{b}_1\,, \tag{10.12}$$

respectively. Setting these partial derivatives to $\mathbf{0}$ gives us the relations

$$\boldsymbol{S}\boldsymbol{b}_1 = \lambda_1 \boldsymbol{b}_1\,, \tag{10.13}$$

$$\boldsymbol{b}_1^\top \boldsymbol{b}_1 = 1\,. \tag{10.14}$$

By comparing this with the definition of an eigenvalue decomposition (Section 4.4), we see that $\boldsymbol{b}_1$ is an eigenvector of the data covariance matrix $\boldsymbol{S}$, and the Lagrange multiplier λ_1 plays the role of the corresponding eigenvalue. This eigenvector property (10.13) allows us to rewrite our variance objective (10.10) as

$$V_1 = \boldsymbol{b}_1^\top \boldsymbol{S} \boldsymbol{b}_1 = \lambda_1 \boldsymbol{b}_1^\top \boldsymbol{b}_1 = \lambda_1\,, \tag{10.15}$$

i.e., the variance of the data projected onto a one-dimensional subspace equals the eigenvalue that is associated with the basis vector $\boldsymbol{b}_1$ that spans this subspace. Therefore, to maximize the variance of the low-dimensional code, we choose the basis vector associated with the largest eigenvalue of the data covariance matrix. This eigenvector is called the first *principal component*. We can determine the effect/contribution of the principal component $\boldsymbol{b}_1$ in the original data space by mapping the coordinate z_{1n} back into data space, which gives us the projected data point

$$\tilde{\boldsymbol{x}}_n = \boldsymbol{b}_1 z_{1n} = \boldsymbol{b}_1 \boldsymbol{b}_1^\top \boldsymbol{x}_n \in \mathbb{R}^D \tag{10.16}$$

in the original data space.

The quantity $\sqrt{\lambda_1}$ is also called the *loading* of the unit vector $\boldsymbol{b}_1$ and represents the standard deviation of the data accounted for by the principal subspace $\operatorname{span}[\boldsymbol{b}_1]$.

principal component

Remark. Although $\tilde{\boldsymbol{x}}_n$ is a D-dimensional vector, it only requires a single coordinate z_{1n} to represent it with respect to the basis vector $\boldsymbol{b}_1 \in \mathbb{R}^D$. $\diamondsuit$

10.2.2 M-dimensional Subspace with Maximal Variance

Assume we have found the first $m-1$ principal components as the $m-1$ eigenvectors of $\boldsymbol{S}$ that are associated with the largest $m-1$ eigenvalues. Since $\boldsymbol{S}$ is symmetric, the spectral theorem (Theorem 4.15) states that we can use these eigenvectors to construct an orthonormal eigenbasis of an

$(m-1)$-dimensional subspace of $\mathbb{R}^D$. Generally, the mth principal component can be found by subtracting the effect of the first $m-1$ principal components $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{m-1}$ from the data, thereby trying to find principal components that compress the remaining information. We then arrive at the new data matrix

$$\hat{\boldsymbol{X}} := \boldsymbol{X} - \sum_{i=1}^{m-1} \boldsymbol{b}_i \boldsymbol{b}_i^\top \boldsymbol{X} = \boldsymbol{X} - \boldsymbol{B}_{m-1}\boldsymbol{X}\,, \tag{10.17}$$

where $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N] \in \mathbb{R}^{D\times N}$ contains the data points as column vectors and $\boldsymbol{B}_{m-1} := \sum_{i=1}^{m-1} \boldsymbol{b}_i \boldsymbol{b}_i^\top$ is a projection matrix that projects onto the subspace spanned by $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{m-1}$.

The matrix $\hat{\boldsymbol{X}} := [\hat{\boldsymbol{x}}_1, \ldots, \hat{\boldsymbol{x}}_N] \in \mathbb{R}^{D\times N}$ in (10.17) contains the information in the data that has not yet been compressed.

Remark (Notation). Throughout this chapter, we do not follow the convention of collecting data $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$ as the rows of the data matrix, but we define them to be the columns of $\boldsymbol{X}$. This means that our data matrix $\boldsymbol{X}$ is a $D \times N$ matrix instead of the conventional $N \times D$ matrix. The reason for our choice is that the algebra operations work out smoothly without the need to either transpose the matrix or to redefine vectors as row vectors that are left-multiplied onto matrices. ♢

To find the mth principal component, we maximize the variance

$$V_m = \mathbb{V}[z_m] = \frac{1}{N}\sum_{n=1}^{N} z_{mn}^2 = \frac{1}{N}\sum_{n=1}^{N} (\boldsymbol{b}_m^\top \hat{\boldsymbol{x}_n})^2 = \boldsymbol{b}_m^\top \hat{\boldsymbol{S}} \boldsymbol{b}_m\,, \tag{10.18}$$

subject to $\|\boldsymbol{b}_m\|^2 = 1$, where we followed the same steps as in (10.9b) and defined $\hat{\boldsymbol{S}}$ as the data covariance matrix of the transformed dataset $\hat{\mathcal{X}} := \{\hat{\boldsymbol{x}}_1, \ldots, \hat{\boldsymbol{x}}_N\}$. As previously, when we looked at the first principal component alone, we solve a constrained optimization problem and discover that the optimal solution $\boldsymbol{b}_m$ is the eigenvector of $\hat{\boldsymbol{S}}$ that is associated with the largest eigenvalue of $\hat{\boldsymbol{S}}$.

It turns out that $\boldsymbol{b}_m$ is also an eigenvector of $\boldsymbol{S}$. More generally, the sets of eigenvectors of $\boldsymbol{S}$ and $\hat{\boldsymbol{S}}$ are identical. Since both $\boldsymbol{S}$ and $\hat{\boldsymbol{S}}$ are symmetric, we can find an ONB of eigenvectors (spectral theorem 4.15), i.e., there exist D distinct eigenvectors for both $\boldsymbol{S}$ and $\hat{\boldsymbol{S}}$. Next, we show that every eigenvector of $\boldsymbol{S}$ is an eigenvector of $\hat{\boldsymbol{S}}$. Assume we have already found eigenvectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{m-1}$ of $\hat{\boldsymbol{S}}$. Consider an eigenvector $\boldsymbol{b}_i$ of $\boldsymbol{S}$, i.e., $\boldsymbol{S}\boldsymbol{b}_i = \lambda_i \boldsymbol{b}_i$. In general,

$$\hat{\boldsymbol{S}}\boldsymbol{b}_i = \frac{1}{N}\hat{\boldsymbol{X}}\hat{\boldsymbol{X}}^\top \boldsymbol{b}_i = \frac{1}{N}(\boldsymbol{X} - \boldsymbol{B}_{m-1}\boldsymbol{X})(\boldsymbol{X} - \boldsymbol{B}_{m-1}\boldsymbol{X})^\top \boldsymbol{b}_i \tag{10.19a}$$

$$= (\boldsymbol{S} - \boldsymbol{S}\boldsymbol{B}_{m-1} - \boldsymbol{B}_{m-1}\boldsymbol{S} + \boldsymbol{B}_{m-1}\boldsymbol{S}\boldsymbol{B}_{m-1})\boldsymbol{b}_i\,. \tag{10.19b}$$

We distinguish between two cases. If $i \geqslant m$, i.e., $\boldsymbol{b}_i$ is an eigenvector that is not among the first $m-1$ principal components, then $\boldsymbol{b}_i$ is orthogonal to the first $m-1$ principal components and $\boldsymbol{B}_{m-1}\boldsymbol{b}_i = \boldsymbol{0}$. If $i < m$, i.e., $\boldsymbol{b}_i$ is among the first $m-1$ principal components, then $\boldsymbol{b}_i$ is a basis vector

of the principal subspace onto which $\boldsymbol{B}_{m-1}$ projects. Since $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{m-1}$ are an ONB of this principal subspace, we obtain $\boldsymbol{B}_{m-1}\boldsymbol{b}_i = \boldsymbol{b}_i$. The two cases can be summarized as follows:

$$\boldsymbol{B}_{m-1}\boldsymbol{b}_i = \boldsymbol{b}_i \quad \text{if } i < m\,, \qquad \boldsymbol{B}_{m-1}\boldsymbol{b}_i = \boldsymbol{0} \quad \text{if } i \geqslant m\,. \tag{10.20}$$

In the case $i \geqslant m$, by using (10.20) in (10.19b), we obtain $\hat{\boldsymbol{S}}\boldsymbol{b}_i = (\boldsymbol{S} - \boldsymbol{B}_{m-1}\boldsymbol{S})\boldsymbol{b}_i = \boldsymbol{S}\boldsymbol{b}_i = \lambda_i \boldsymbol{b}_i$, i.e., $\boldsymbol{b}_i$ is also an eigenvector of $\hat{\boldsymbol{S}}$ with eigenvalue λ_i. Specifically,

$$\hat{\boldsymbol{S}}\boldsymbol{b}_m = \boldsymbol{S}\boldsymbol{b}_m = \lambda_m \boldsymbol{b}_m\,. \tag{10.21}$$

Equation (10.21) reveals that $\boldsymbol{b}_m$ is not only an eigenvector of $\boldsymbol{S}$ but also of $\hat{\boldsymbol{S}}$. Specifically, λ_m is the largest eigenvalue of $\hat{\boldsymbol{S}}$ and λ_m is the mth largest eigenvalue of $\boldsymbol{S}$, and both have the associated eigenvector $\boldsymbol{b}_m$.

In the case $i < m$, by using (10.20) in (10.19b), we obtain

$$\hat{\boldsymbol{S}}\boldsymbol{b}_i = (\boldsymbol{S} - \boldsymbol{S}\boldsymbol{B}_{m-1} - \boldsymbol{B}_{m-1}\boldsymbol{S} + \boldsymbol{B}_{m-1}\boldsymbol{S}\boldsymbol{B}_{m-1})\boldsymbol{b}_i = \boldsymbol{0} = 0\boldsymbol{b}_i \tag{10.22}$$

This means that $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{m-1}$ are also eigenvectors of $\hat{\boldsymbol{S}}$, but they are associated with eigenvalue 0 so that $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{m-1}$ span the null space of $\hat{\boldsymbol{S}}$.

Overall, every eigenvector of $\boldsymbol{S}$ is also an eigenvector of $\hat{\boldsymbol{S}}$. However, if the eigenvectors of $\boldsymbol{S}$ are part of the $(m-1)$ dimensional principal subspace, then the associated eigenvalue of $\hat{\boldsymbol{S}}$ is 0.

This derivation shows that there is an intimate connection between the M-dimensional subspace with maximal variance and the eigenvalue decomposition. We will revisit this connection in Section 10.4.

With the relation (10.21) and $\boldsymbol{b}_m^\top \boldsymbol{b}_m = 1$, the variance of the data projected onto the mth principal component is

$$V_m = \boldsymbol{b}_m^\top \boldsymbol{S}\boldsymbol{b}_m \overset{(10.21)}{=} \lambda_m \boldsymbol{b}_m^\top \boldsymbol{b}_m = \lambda_m\,. \tag{10.23}$$

This means that the variance of the data, when projected onto an M-dimensional subspace, equals the sum of the eigenvalues that are associated with the corresponding eigenvectors of the data covariance matrix.

Example 10.2 (Eigenvalues of MNIST "8")

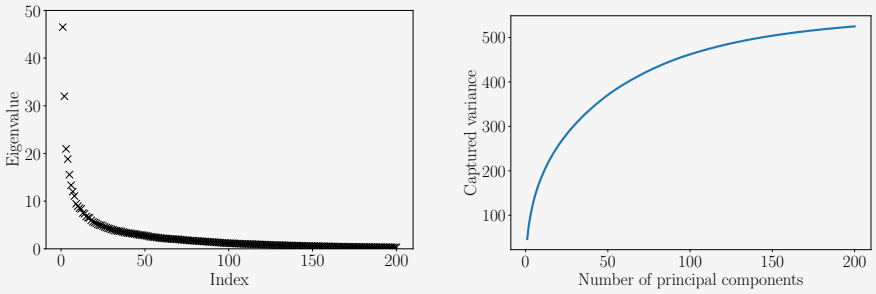


Figure 10.5 Properties of the training data of MNIST "8". (a) Eigenvalues sorted in descending order; (b) Variance captured by the principal components associated with the largest eigenvalues.

(a) Eigenvalues (sorted in descending order) of the data covariance matrix of all digits "8" in the MNIST training set.

(b) Variance captured by the principal components.

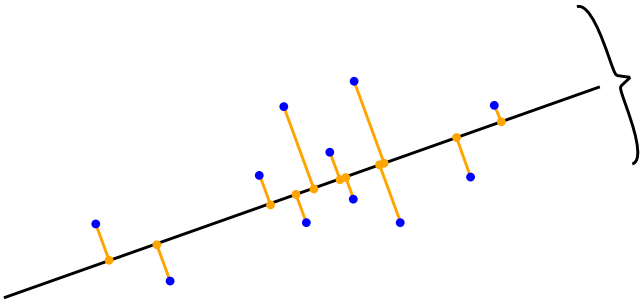


Figure 10.6 Illustration of the projection approach: Find a subspace (line) that minimizes the length of the difference vector between projected (orange) and original (blue) data.

Taking all digits "8" in the MNIST training data, we compute the eigenvalues of the data covariance matrix. Figure 10.5(a) shows the 200 largest eigenvalues of the data covariance matrix. We see that only a few of them have a value that differs significantly from 0. Therefore, most of the variance, when projecting data onto the subspace spanned by the corresponding eigenvectors, is captured by only a few principal components, as shown in Figure 10.5(b).

Overall, to find an M-dimensional subspace of $\mathbb{R}^D$ that retains as much information as possible, PCA tells us to choose the columns of the matrix $\boldsymbol{B}$ in (10.3) as the M eigenvectors of the data covariance matrix $\boldsymbol{S}$ that are associated with the M largest eigenvalues. The maximum amount of variance PCA can capture with the first M principal components is

$$V_M = \sum_{m=1}^{M} \lambda_m \,, \tag{10.24}$$

where the λ_m are the M largest eigenvalues of the data covariance matrix $\boldsymbol{S}$. Consequently, the variance lost by data compression via PCA is

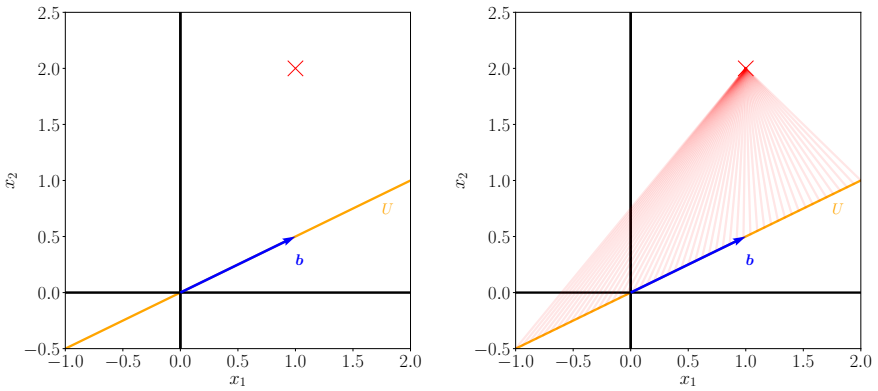
$$J_M := \sum_{j=M+1}^{D} \lambda_j = V_D - V_M \,. \tag{10.25}$$

Instead of these absolute quantities, we can define the relative variance captured as $\frac{V_M}{V_D}$, and the relative variance lost by compression as $1 - \frac{V_M}{V_D}$.

10.3 Projection Perspective

In the following, we will derive PCA as an algorithm that directly minimizes the average reconstruction error. This perspective allows us to interpret PCA as implementing an optimal linear auto-encoder. We will draw heavily from Chapters 2 and 3.

In the previous section, we derived PCA by maximizing the variance in the projected space to retain as much information as possible. In the



(a) Setting.

(b) Differences $\boldsymbol{x} - \tilde{\boldsymbol{x}}_i$ for 50 different $\tilde{\boldsymbol{x}}_i$ are shown by the red lines.

Figure 10.7 Simplified projection setting. (a) A vector $\boldsymbol{x} \in \mathbb{R}^2$ (red cross) shall be projected onto a one-dimensional subspace $U \subseteq \mathbb{R}^2$ spanned by $\boldsymbol{b}$. (b) shows the difference vectors between $\boldsymbol{x}$ and some candidates $\tilde{\boldsymbol{x}}$.

following, we will look at the difference vectors between the original data $\boldsymbol{x}_n$ and their reconstruction $\tilde{\boldsymbol{x}}_n$ and minimize this distance so that $\boldsymbol{x}_n$ and $\tilde{\boldsymbol{x}}_n$ are as close as possible. Figure 10.6 illustrates this setting.

10.3.1 Setting and Objective

Assume an (ordered) orthonormal basis (ONB) $B = (\boldsymbol{b}_1, \ldots, \boldsymbol{b}_D)$ of $\mathbb{R}^D$, i.e., $\boldsymbol{b}_i^\top \boldsymbol{b}_j = 1$ if and only if $i = j$ and 0 otherwise.

From Section 2.5 we know that for a basis $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_D)$ of $\mathbb{R}^D$ any $\boldsymbol{x} \in \mathbb{R}^D$ can be written as a linear combination of the basis vectors of $\mathbb{R}^D$, i.e.,

Vectors $\tilde{\boldsymbol{x}} \in U$ could be vectors on a plane in $\mathbb{R}^3$. The dimensionality of the plane is 2, but the vectors still have three coordinates with respect to the standard basis of $\mathbb{R}^3$.

$$\boldsymbol{x} = \sum_{d=1}^{D} \zeta_d \boldsymbol{b}_d = \sum_{m=1}^{M} \zeta_m \boldsymbol{b}_m + \sum_{j=M+1}^{D} \zeta_j \boldsymbol{b}_j \tag{10.26}$$

for suitable coordinates $\zeta_d \in \mathbb{R}$.

We are interested in finding vectors $\tilde{\boldsymbol{x}} \in \mathbb{R}^D$, which live in lower-dimensional subspace $U \subseteq \mathbb{R}^D$, $\dim(U) = M$, so that

$$\tilde{\boldsymbol{x}} = \sum_{m=1}^{M} z_m \boldsymbol{b}_m \in U \subseteq \mathbb{R}^D \tag{10.27}$$

is as similar to $\boldsymbol{x}$ as possible. Note that at this point we need to assume that the coordinates z_m of $\tilde{\boldsymbol{x}}$ and ζ_m of $\boldsymbol{x}$ are not identical.

In the following, we use exactly this kind of representation of $\tilde{\boldsymbol{x}}$ to find optimal coordinates $\boldsymbol{z}$ and basis vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_M$ such that $\tilde{\boldsymbol{x}}$ is as similar to the original data point $\boldsymbol{x}$ as possible, i.e., we aim to minimize the (Euclidean) distance $\|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|$. Figure 10.7 illustrates this setting.

Without loss of generality, we assume that the dataset $\mathcal{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, $\boldsymbol{x}_n \in \mathbb{R}^D$, is centered at $\boldsymbol{0}$, i.e., $\mathbb{E}[\mathcal{X}] = \boldsymbol{0}$. Without the zero-mean assump-

tion, we would arrive at exactly the same solution, but the notation would be substantially more cluttered.

We are interested in finding the best linear projection of $\mathcal{X}$ onto a lower-dimensional subspace U of $\mathbb{R}^D$ with $\dim(U) = M$ and orthonormal basis vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_M$. We will call this subspace U the *principal subspace*. The projections of the data points are denoted by

principal subspace

$$\tilde{\boldsymbol{x}}_n := \sum_{m=1}^{M} z_{mn}\boldsymbol{b}_m = \boldsymbol{B}\boldsymbol{z}_n \in \mathbb{R}^D\,, \tag{10.28}$$

where $\boldsymbol{z}_n := [z_{1n}, \ldots, z_{Mn}]^\top \in \mathbb{R}^M$ is the coordinate vector of $\tilde{\boldsymbol{x}}_n$ with respect to the basis $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_M)$. More specifically, we are interested in having the $\tilde{\boldsymbol{x}}_n$ as similar to $\boldsymbol{x}_n$ as possible.

The similarity measure we use in the following is the squared distance (Euclidean norm) $\|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|^2$ between $\boldsymbol{x}$ and $\tilde{\boldsymbol{x}}$. We therefore define our objective as minimizing the average squared Euclidean distance (*reconstruction error*) (Pearson, 1901)

reconstruction error

$$J_M := \frac{1}{N}\sum_{n=1}^{N} \|\boldsymbol{x}_n - \tilde{\boldsymbol{x}}_n\|^2\,, \tag{10.29}$$

where we make it explicit that the dimension of the subspace onto which we project the data is M. In order to find this optimal linear projection, we need to find the orthonormal basis of the principal subspace and the coordinates $\boldsymbol{z}_n \in \mathbb{R}^M$ of the projections with respect to this basis.

To find the coordinates $\boldsymbol{z}_n$ and the ONB of the principal subspace, we follow a two-step approach. First, we optimize the coordinates $\boldsymbol{z}_n$ for a given ONB $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_M)$; second, we find the optimal ONB.

10.3.2 Finding Optimal Coordinates

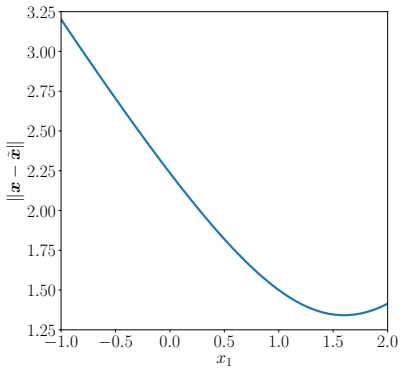
Let us start by finding the optimal coordinates $z_{1n}, \ldots, z_{Mn}$ of the projections $\tilde{\boldsymbol{x}}_n$ for $n = 1, \ldots, N$. Consider Figure 10.7(b), where the principal subspace is spanned by a single vector $\boldsymbol{b}$. Geometrically speaking, finding the optimal coordinates z corresponds to finding the representation of the linear projection $\tilde{\boldsymbol{x}}$ with respect to $\boldsymbol{b}$ that minimizes the distance between $\tilde{\boldsymbol{x}} - \boldsymbol{x}$. From Figure 10.7(b), it is clear that this will be the orthogonal projection, and in the following we will show exactly this.

We assume an ONB $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_M)$ of $U \subseteq \mathbb{R}^D$. To find the optimal coordinates $\boldsymbol{z}_m$ with respect to this basis, we require the partial derivatives

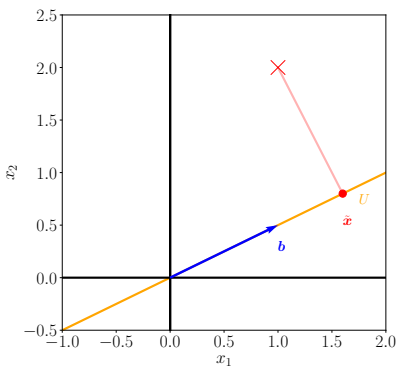
$$\frac{\partial J_M}{\partial z_{in}} = \frac{\partial J_M}{\partial \tilde{\boldsymbol{x}}_n}\frac{\partial \tilde{\boldsymbol{x}}_n}{\partial z_{in}}\,, \tag{10.30a}$$

$$\frac{\partial J_M}{\partial \tilde{\boldsymbol{x}}_n} = -\frac{2}{N}(\boldsymbol{x}_n - \tilde{\boldsymbol{x}}_n)^\top \in \mathbb{R}^{1\times D}\,, \tag{10.30b}$$

Figure 10.8 Optimal projection of a vector $\boldsymbol{x} \in \mathbb{R}^2$ onto a one-dimensional subspace (continuation from Figure 10.7). (a) Distances $\|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|$ for some $\tilde{\boldsymbol{x}} \in U$. (b) Orthogonal projection and optimal coordinates.



(a) Distances $\|\boldsymbol{x} - \tilde{\boldsymbol{x}}\|$ for some $\tilde{\boldsymbol{x}} = z_1\boldsymbol{b} \in U = \text{span}[\boldsymbol{b}]$; see panel (b) for the setting.



(b) The vector $\tilde{\boldsymbol{x}}$ that minimizes the distance in panel (a) is its orthogonal projection onto U. The coordinate of the projection $\tilde{\boldsymbol{x}}$ with respect to the basis vector $\boldsymbol{b}$ that spans U is the factor we need to scale $\boldsymbol{b}$ in order to "reach" $\tilde{\boldsymbol{x}}$.

$$\frac{\partial \tilde{\boldsymbol{x}}_n}{\partial z_{in}} \overset{(10.28)}{=} \frac{\partial}{\partial z_{in}} \left(\sum_{m=1}^{M} z_{mn} \boldsymbol{b}_m \right) = \boldsymbol{b}_i \tag{10.30c}$$

for $i = 1, \ldots, M$, such that we obtain

$$\frac{\partial J_M}{\partial z_{in}} \overset{\substack{(10.30b)\\(10.30c)}}{=} -\frac{2}{N}(\boldsymbol{x}_n - \tilde{\boldsymbol{x}}_n)^\top \boldsymbol{b}_i \overset{(10.28)}{=} -\frac{2}{N}\left(\boldsymbol{x}_n - \sum_{m=1}^{M} z_{mn}\boldsymbol{b}_m \right)^\top \boldsymbol{b}_i \tag{10.31a}$$

$$\overset{\text{ONB}}{=} -\frac{2}{N}(\boldsymbol{x}_n^\top \boldsymbol{b}_i - z_{in}\boldsymbol{b}_i^\top \boldsymbol{b}_i) = -\frac{2}{N}(\boldsymbol{x}_n^\top \boldsymbol{b}_i - z_{in})\,. \tag{10.31b}$$

The coordinates of the optimal projection of $\boldsymbol{x}_n$ with respect to the basis vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_M$ are the coordinates of the orthogonal projection of $\boldsymbol{x}_n$ onto the principal subspace.

since $\boldsymbol{b}_i^\top \boldsymbol{b}_i = 1$. Setting this partial derivative to 0 yields immediately the optimal coordinates

$$z_{in} = \boldsymbol{x}_n^\top \boldsymbol{b}_i = \boldsymbol{b}_i^\top \boldsymbol{x}_n \tag{10.32}$$

for $i = 1, \ldots, M$ and $n = 1, \ldots, N$. This means that the optimal coordinates z_{in} of the projection $\tilde{\boldsymbol{x}}_n$ are the coordinates of the orthogonal projection (see Section 3.8) of the original data point $\boldsymbol{x}_n$ onto the one-dimensional subspace that is spanned by $\boldsymbol{b}_i$. Consequently:

- The optimal linear projection $\tilde{\boldsymbol{x}}_n$ of $\boldsymbol{x}_n$ is an orthogonal projection.
- The coordinates of $\tilde{\boldsymbol{x}}_n$ with respect to the basis $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_M)$ are the coordinates of the orthogonal projection of $\boldsymbol{x}_n$ onto the principal subspace.
- An orthogonal projection is the best linear mapping given the objective (10.29).
- The coordinates ζ_m of $\boldsymbol{x}$ in (10.26) and the coordinates z_m of $\tilde{\boldsymbol{x}}$ in (10.27)

must be identical for $m = 1, \ldots, M$ since $U^\perp = \text{span}[\boldsymbol{b}_{M+1}, \ldots, \boldsymbol{b}_D]$ is the orthogonal complement (see Section 3.6) of $U = \text{span}[\boldsymbol{b}_1, \ldots, \boldsymbol{b}_M]$.

Remark (Orthogonal Projections with Orthonormal Basis Vectors). Let us briefly recap orthogonal projections from Section 3.8. If $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_D)$ is an orthonormal basis of $\mathbb{R}^D$ then

$\boldsymbol{b}_j^\top \boldsymbol{x}$ is the coordinate of the orthogonal projection of $\boldsymbol{x}$ onto the subspace spanned by $\boldsymbol{b}_j$.

$$\tilde{\boldsymbol{x}} = \boldsymbol{b}_j(\boldsymbol{b}_j^\top \boldsymbol{b}_j)^{-1}\boldsymbol{b}_j^\top \boldsymbol{x} = \boldsymbol{b}_j \boldsymbol{b}_j^\top \boldsymbol{x} \in \mathbb{R}^D \tag{10.33}$$

is the orthogonal projection of $\boldsymbol{x}$ onto the subspace spanned by the jth basis vector, and $z_j = \boldsymbol{b}_j^\top \boldsymbol{x}$ is the coordinate of this projection with respect to the basis vector $\boldsymbol{b}_j$ that spans that subspace since $z_j \boldsymbol{b}_j = \tilde{\boldsymbol{x}}$. Figure 10.8(b) illustrates this setting.

More generally, if we aim to project onto an M-dimensional subspace of $\mathbb{R}^D$, we obtain the orthogonal projection of $\boldsymbol{x}$ onto the M-dimensional subspace with orthonormal basis vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_M$ as

$$\tilde{\boldsymbol{x}} = \boldsymbol{B}(\underbrace{\boldsymbol{B}^\top \boldsymbol{B}}_{=\boldsymbol{I}})^{-1}\boldsymbol{B}^\top \boldsymbol{x} = \boldsymbol{B}\boldsymbol{B}^\top \boldsymbol{x}\,, \tag{10.34}$$

where we defined $\boldsymbol{B} := [\boldsymbol{b}_1, \ldots, \boldsymbol{b}_M] \in \mathbb{R}^{D \times M}$. The coordinates of this projection with respect to the ordered basis $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_M)$ are $\boldsymbol{z} := \boldsymbol{B}^\top \boldsymbol{x}$ as discussed in Section 3.8.

We can think of the coordinates as a representation of the projected vector in a new coordinate system defined by $(\boldsymbol{b}_1, \ldots, \boldsymbol{b}_M)$. Note that although $\tilde{\boldsymbol{x}} \in \mathbb{R}^D$, we only need M coordinates $z_1, \ldots, z_M$ to represent this vector; the other $D - M$ coordinates with respect to the basis vectors $(\boldsymbol{b}_{M+1}, \ldots, \boldsymbol{b}_D)$ are always 0. ♢

So far we have shown that for a given ONB we can find the optimal coordinates of $\tilde{\boldsymbol{x}}$ by an orthogonal projection onto the principal subspace. In the following, we will determine what the best basis is.

10.3.3 Finding the Basis of the Principal Subspace

To determine the basis vectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_M$ of the principal subspace, we rephrase the loss function (10.29) using the results we have so far. This will make it easier to find the basis vectors. To reformulate the loss function, we exploit our results from before and obtain

$$\tilde{\boldsymbol{x}}_n = \sum_{m=1}^{M} z_{mn}\boldsymbol{b}_m \overset{(10.32)}{=} \sum_{m=1}^{M} (\boldsymbol{x}_n^\top \boldsymbol{b}_m)\boldsymbol{b}_m\,. \tag{10.35}$$

We now exploit the symmetry of the dot product, which yields

$$\tilde{\boldsymbol{x}}_n = \left(\sum_{m=1}^{M} \boldsymbol{b}_m \boldsymbol{b}_m^\top\right)\boldsymbol{x}_n\,. \tag{10.36}$$

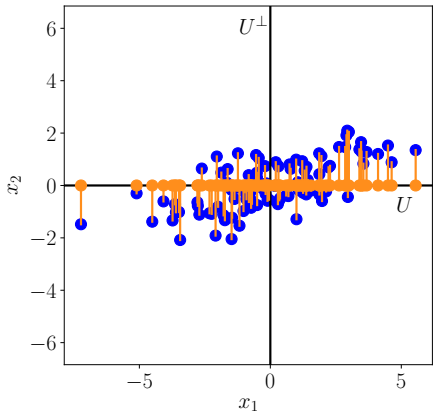


Figure 10.9 Orthogonal projection and displacement vectors. When projecting data points $\boldsymbol{x}_n$ (blue) onto subspace U_1, we obtain $\tilde{\boldsymbol{x}}_n$ (orange). The displacement vector $\tilde{\boldsymbol{x}}_n - \boldsymbol{x}_n$ lies completely in the orthogonal complement U_2 of U_1.

Since we can generally write the original data point $\boldsymbol{x}_n$ as a linear combination of all basis vectors, it holds that

$$\boldsymbol{x}_n = \sum_{d=1}^{D} z_{dn}\boldsymbol{b}_d \overset{(10.32)}{=} \sum_{d=1}^{D}(\boldsymbol{x}_n^\top \boldsymbol{b}_d)\boldsymbol{b}_d = \left(\sum_{d=1}^{D}\boldsymbol{b}_d\boldsymbol{b}_d^\top\right)\boldsymbol{x}_n \tag{10.37a}$$

$$= \left(\sum_{m=1}^{M}\boldsymbol{b}_m\boldsymbol{b}_m^\top\right)\boldsymbol{x}_n + \left(\sum_{j=M+1}^{D}\boldsymbol{b}_j\boldsymbol{b}_j^\top\right)\boldsymbol{x}_n\,, \tag{10.37b}$$

where we split the sum with D terms into a sum over M and a sum over $D-M$ terms. With this result, we find that the displacement vector $\boldsymbol{x}_n - \tilde{\boldsymbol{x}}_n$, i.e., the difference vector between the original data point and its projection, is

$$\boldsymbol{x}_n - \tilde{\boldsymbol{x}}_n = \left(\sum_{j=M+1}^{D}\boldsymbol{b}_j\boldsymbol{b}_j^\top\right)\boldsymbol{x}_n \tag{10.38a}$$

$$= \sum_{j=M+1}^{D}(\boldsymbol{x}_n^\top \boldsymbol{b}_j)\boldsymbol{b}_j\,. \tag{10.38b}$$

This means the difference is exactly the projection of the data point onto the orthogonal complement of the principal subspace: We identify the matrix $\sum_{j=M+1}^{D}\boldsymbol{b}_j\boldsymbol{b}_j^\top$ in (10.38a) as the projection matrix that performs this projection. Hence the displacement vector $\boldsymbol{x}_n - \tilde{\boldsymbol{x}}_n$ lies in the subspace that is orthogonal to the principal subspace as illustrated in Figure 10.9.

Remark (Low-Rank Approximation). In (10.38a), we saw that the projection matrix, which projects $\boldsymbol{x}$ onto $\tilde{\boldsymbol{x}}$, is given by

$$\sum_{m=1}^{M}\boldsymbol{b}_m\boldsymbol{b}_m^\top = \boldsymbol{B}\boldsymbol{B}^\top\,. \tag{10.39}$$

By construction as a sum of rank-one matrices $\boldsymbol{b}_m\boldsymbol{b}_m^\top$ we see that $\boldsymbol{B}\boldsymbol{B}^\top$ is

symmetric and has rank M. Therefore, the average squared reconstruction error can also be written as

$$\frac{1}{N}\sum_{n=1}^{N}\|\boldsymbol{x}_n-\tilde{\boldsymbol{x}}_n\|^2=\frac{1}{N}\sum_{n=1}^{N}\left\|\boldsymbol{x}_n-\boldsymbol{B}\boldsymbol{B}^\top\boldsymbol{x}_n\right\|^2 \tag{10.40a}$$

$$=\frac{1}{N}\sum_{n=1}^{N}\left\|(\boldsymbol{I}-\boldsymbol{B}\boldsymbol{B}^\top)\boldsymbol{x}_n\right\|^2 . \tag{10.40b}$$

Finding orthonormal basis vectors $\boldsymbol{b}_1,\dots,\boldsymbol{b}_M$, which minimize the difference between the original data $\boldsymbol{x}_n$ and their projections $\tilde{\boldsymbol{x}}_n$, is equivalent to finding the best rank-M approximation $\boldsymbol{B}\boldsymbol{B}^\top$ of the identity matrix $\boldsymbol{I}$ (see Section 4.6). ♢

PCA finds the best rank-M approximation of the identity matrix.

Now we have all the tools to reformulate the loss function (10.29).

$$J_M=\frac{1}{N}\sum_{n=1}^{N}\|\boldsymbol{x}_n-\tilde{\boldsymbol{x}}_n\|^2\overset{(10.38b)}{=}\frac{1}{N}\sum_{n=1}^{N}\left\|\sum_{j=M+1}^{D}(\boldsymbol{b}_j^\top\boldsymbol{x}_n)\boldsymbol{b}_j\right\|^2 . \tag{10.41}$$

We now explicitly compute the squared norm and exploit the fact that the $\boldsymbol{b}_j$ form an ONB, which yields

$$J_M=\frac{1}{N}\sum_{n=1}^{N}\sum_{j=M+1}^{D}(\boldsymbol{b}_j^\top\boldsymbol{x}_n)^2=\frac{1}{N}\sum_{n=1}^{N}\sum_{j=M+1}^{D}\boldsymbol{b}_j^\top\boldsymbol{x}_n\boldsymbol{b}_j^\top\boldsymbol{x}_n \tag{10.42a}$$

$$=\frac{1}{N}\sum_{n=1}^{N}\sum_{j=M+1}^{D}\boldsymbol{b}_j^\top\boldsymbol{x}_n\boldsymbol{x}_n^\top\boldsymbol{b}_j , \tag{10.42b}$$

where we exploited the symmetry of the dot product in the last step to write $\boldsymbol{b}_j^\top\boldsymbol{x}_n=\boldsymbol{x}_n^\top\boldsymbol{b}_j$. We now swap the sums and obtain

$$J_M=\sum_{j=M+1}^{D}\boldsymbol{b}_j^\top\underbrace{\left(\frac{1}{N}\sum_{n=1}^{N}\boldsymbol{x}_n\boldsymbol{x}_n^\top\right)}_{=:\boldsymbol{S}}\boldsymbol{b}_j=\sum_{j=M+1}^{D}\boldsymbol{b}_j^\top\boldsymbol{S}\boldsymbol{b}_j \tag{10.43a}$$

$$=\sum_{j=M+1}^{D}\operatorname{tr}(\boldsymbol{b}_j^\top\boldsymbol{S}\boldsymbol{b}_j)=\sum_{j=M+1}^{D}\operatorname{tr}(\boldsymbol{S}\boldsymbol{b}_j\boldsymbol{b}_j^\top)=\operatorname{tr}\Big(\underbrace{\Big(\sum_{j=M+1}^{D}\boldsymbol{b}_j\boldsymbol{b}_j^\top\Big)}_{\text{projection matrix}}\boldsymbol{S}\Big), \tag{10.43b}$$

where we exploited the property that the trace operator $\operatorname{tr}(\cdot)$ (see (4.18)) is linear and invariant to cyclic permutations of its arguments. Since we assumed that our dataset is centered, i.e., $\mathbb{E}[\mathcal{X}]=\boldsymbol{0}$, we identify $\boldsymbol{S}$ as the data covariance matrix. Since the projection matrix in (10.43b) is constructed as a sum of rank-one matrices $\boldsymbol{b}_j\boldsymbol{b}_j^\top$ it itself is of rank $D-M$.

Equation (10.43a) implies that we can formulate the average squared reconstruction error equivalently as the covariance matrix of the data,

projected onto the orthogonal complement of the principal subspace. Minimizing the average squared reconstruction error is therefore equivalent to minimizing the variance of the data when projected onto the subspace we ignore, i.e., the orthogonal complement of the principal subspace. Equivalently, we maximize the variance of the projection that we retain in the principal subspace, which links the projection loss immediately to the maximum-variance formulation of PCA discussed in Section 10.2. But this then also means that we will obtain the same solution that we obtained for the maximum-variance perspective. Therefore, we omit a derivation that is identical to the one presented in Section 10.2 and summarize the results from earlier in the light of the projection perspective.

Minimizing the average squared reconstruction error is equivalent to minimizing the projection of the data covariance matrix onto the orthogonal complement of the principal subspace.

Minimizing the average squared reconstruction error is equivalent to maximizing the variance of the projected data.

The average squared reconstruction error, when projecting onto the M-dimensional principal subspace, is

$$J_M = \sum_{j=M+1}^{D} \lambda_j \,, \tag{10.44}$$

where λ_j are the eigenvalues of the data covariance matrix. Therefore, to minimize (10.44) we need to select the smallest $D - M$ eigenvalues, which then implies that their corresponding eigenvectors are the basis of the orthogonal complement of the principal subspace. Consequently, this means that the basis of the principal subspace comprises the eigenvectors $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_M$ that are associated with the largest M eigenvalues of the data covariance matrix.

Example 10.3 (MNIST Digits Embedding)

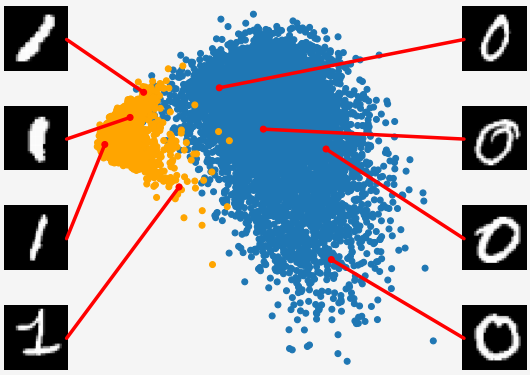


Figure 10.10 Embedding of MNIST digits 0 (blue) and 1 (orange) in a two-dimensional principal subspace using PCA. Four embeddings of the digits "0" and "1" in the principal subspace are highlighted in red with their corresponding original digit.

Figure 10.10 visualizes the training data of the MMIST digits "0" and "1" embedded in the vector subspace spanned by the first two principal components. We observe a relatively clear separation between "0"s (blue dots) and "1"s (orange dots), and we see the variation within each individual

cluster. Four embeddings of the digits "0" and "1" in the principal subspace are highlighted in red with their corresponding original digit. The figure reveals that the variation within the set of "0" is significantly greater than the variation within the set of "1".

10.4 Eigenvector Computation and Low-Rank Approximations

In the previous sections, we obtained the basis of the principal subspace as the eigenvectors that are associated with the largest eigenvalues of the data covariance matrix

$$\boldsymbol{S} = \frac{1}{N}\sum_{n=1}^{N} \boldsymbol{x}_n \boldsymbol{x}_n^\top = \frac{1}{N}\boldsymbol{X}\boldsymbol{X}^\top, \tag{10.45}$$

$$\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N] \in \mathbb{R}^{D\times N}. \tag{10.46}$$

Note that $\boldsymbol{X}$ is a $D \times N$ matrix, i.e., it is the transpose of the "typical" data matrix (Bishop, 2006; Murphy, 2012). To get the eigenvalues (and the corresponding eigenvectors) of $\boldsymbol{S}$, we can follow two approaches:

Use eigendecomposition or SVD to compute eigenvectors.

- We perform an eigendecomposition (see Section 4.2) and compute the eigenvalues and eigenvectors of $\boldsymbol{S}$ directly.
- We use a singular value decomposition (see Section 4.5). Since $\boldsymbol{S}$ is symmetric and factorizes into $\boldsymbol{X}\boldsymbol{X}^\top$ (ignoring the factor $\frac{1}{N}$), the eigenvalues of $\boldsymbol{S}$ are the squared singular values of $\boldsymbol{X}$.

  More specifically, the SVD of $\boldsymbol{X}$ is given by

$$\underbrace{\boldsymbol{X}}_{D\times N} = \underbrace{\boldsymbol{U}}_{D\times D}\underbrace{\boldsymbol{\Sigma}}_{D\times N}\underbrace{\boldsymbol{V}^\top}_{N\times N}, \tag{10.47}$$

where $\boldsymbol{U} \in \mathbb{R}^{D\times D}$ and $\boldsymbol{V}^\top \in \mathbb{R}^{N\times N}$ are orthogonal matrices and $\boldsymbol{\Sigma} \in \mathbb{R}^{D\times N}$ is a matrix whose only nonzero entries are the singular values $\sigma_{ii} \geqslant 0$. It then follows that

$$\boldsymbol{S} = \frac{1}{N}\boldsymbol{X}\boldsymbol{X}^\top = \frac{1}{N}\boldsymbol{U}\boldsymbol{\Sigma}\underbrace{\boldsymbol{V}^\top\boldsymbol{V}}_{=\boldsymbol{I}_N}\boldsymbol{\Sigma}^\top\boldsymbol{U}^\top = \frac{1}{N}\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{\Sigma}^\top\boldsymbol{U}^\top. \tag{10.48}$$

The columns of $\boldsymbol{U}$ are the eigenvectors of $\boldsymbol{S}$.

With the results from Section 4.5, we get that the columns of $\boldsymbol{U}$ are the eigenvectors of $\boldsymbol{X}\boldsymbol{X}^\top$ (and therefore $\boldsymbol{S}$). Furthermore, the eigenvalues λ_d of $\boldsymbol{S}$ are related to the singular values of $\boldsymbol{X}$ via

$$\lambda_d = \frac{\sigma_d^2}{N}. \tag{10.49}$$

This relationship between the eigenvalues of $\boldsymbol{S}$ and the singular values of $\boldsymbol{X}$ provides the connection between the maximum variance view (Section 10.2) and the singular value decomposition.

10.4.1 PCA Using Low-Rank Matrix Approximations

To maximize the variance of the projected data (or minimize the average squared reconstruction error), PCA chooses the columns of $\boldsymbol{U}$ in (10.48) to be the eigenvectors that are associated with the M largest eigenvalues of the data covariance matrix $\boldsymbol{S}$ so that we identify $\boldsymbol{U}$ as the projection matrix $\boldsymbol{B}$ in (10.3), which projects the original data onto a lower-dimensional subspace of dimension M. The *Eckart-Young theorem* (Theorem 4.25 in Section 4.6) offers a direct way to estimate the low-dimensional representation. Consider the best rank-M approximation

Eckart-Young theorem

$$\tilde{\boldsymbol{X}}_M := \operatorname{argmin}_{\operatorname{rk}(\boldsymbol{A}) \leqslant M} \|\boldsymbol{X} - \boldsymbol{A}\|_2 \in \mathbb{R}^{D \times N} \tag{10.50}$$

of $\boldsymbol{X}$, where $\|\cdot\|_2$ is the spectral norm defined in (4.93). The Eckart-Young theorem states that $\tilde{\boldsymbol{X}}_M$ is given by truncating the SVD at the top-M singular value. In other words, we obtain

$$\tilde{\boldsymbol{X}}_M = \underbrace{\boldsymbol{U}_M}_{D \times M} \underbrace{\boldsymbol{\Sigma}_M}_{M \times M} \underbrace{\boldsymbol{V}_M^\top}_{M \times N} \in \mathbb{R}^{D \times N} \tag{10.51}$$

with orthogonal matrices $\boldsymbol{U}_M := [\boldsymbol{u}_1, \ldots, \boldsymbol{u}_M] \in \mathbb{R}^{D \times M}$ and $\boldsymbol{V}_M := [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_M] \in \mathbb{R}^{N \times M}$ and a diagonal matrix $\boldsymbol{\Sigma}_M \in \mathbb{R}^{M \times M}$ whose diagonal entries are the M largest singular values of $\boldsymbol{X}$.

10.4.2 Practical Aspects

Finding eigenvalues and eigenvectors is also important in other fundamental machine learning methods that require matrix decompositions. In theory, as we discussed in Section 4.2, we can solve for the eigenvalues as roots of the characteristic polynomial. However, for matrices larger than 4×4 this is not possible because we would need to find the roots of a polynomial of degree 5 or higher. However, the *Abel-Ruffini theorem* (Ruffini, 1799; Abel, 1826) states that there exists no algebraic solution to this problem for polynomials of degree 5 or more. Therefore, in practice, we solve for eigenvalues or singular values using iterative methods, which are implemented in all modern packages for linear algebra.

Abel-Ruffini theorem

`np.linalg.eigh` or `np.linalg.svd`

In many applications (such as PCA presented in this chapter), we only require a few eigenvectors. It would be wasteful to compute the full decomposition, and then discard all eigenvectors with eigenvalues that are beyond the first few. It turns out that if we are interested in only the first few eigenvectors (with the largest eigenvalues), then iterative processes, which directly optimize these eigenvectors, are computationally more efficient than a full eigendecomposition (or SVD). In the extreme case of only needing the first eigenvector, a simple method called the *power iteration* is very efficient. Power iteration chooses a random vector $\boldsymbol{x}_0$ that is not in

power iteration

the null space of $\boldsymbol{S}$ and follows the iteration

$$\boldsymbol{x}_{k+1} = \frac{\boldsymbol{S}\boldsymbol{x}_k}{\|\boldsymbol{S}\boldsymbol{x}_k\|}, \quad k = 0, 1, \ldots . \tag{10.52}$$

This means the vector $\boldsymbol{x}_k$ is multiplied by $\boldsymbol{S}$ in every iteration and then normalized, i.e., we always have $\|\boldsymbol{x}_k\| = 1$. This sequence of vectors converges to the eigenvector associated with the largest eigenvalue of $\boldsymbol{S}$. The original Google PageRank algorithm (Page et al., 1999) uses such an algorithm for ranking web pages based on their hyperlinks.

If $\boldsymbol{S}$ is invertible, it is sufficient to ensure that $\boldsymbol{x}_0 \neq \boldsymbol{0}$.

10.5 PCA in High Dimensions

In order to do PCA, we need to compute the data covariance matrix. In D dimensions, the data covariance matrix is a $D \times D$ matrix. Computing the eigenvalues and eigenvectors of this matrix is computationally expensive as it scales cubically in D. Therefore, PCA, as we discussed earlier, will be infeasible in very high dimensions. For example, if our $\boldsymbol{x}_n$ are images with $10{,}000$ pixels (e.g., 100×100 pixel images), we would need to compute the eigendecomposition of a $10{,}000 \times 10{,}000$ covariance matrix. In the following, we provide a solution to this problem for the case that we have substantially fewer data points than dimensions, i.e., $N \ll D$.

Assume we have a centered dataset $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, $\boldsymbol{x}_n \in \mathbb{R}^D$. Then the data covariance matrix is given as

$$\boldsymbol{S} = \frac{1}{N}\boldsymbol{X}\boldsymbol{X}^\top \in \mathbb{R}^{D \times D}, \tag{10.53}$$

where $\boldsymbol{X} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]$ is a $D \times N$ matrix whose columns are the data points.

We now assume that $N \ll D$, i.e., the number of data points is smaller than the dimensionality of the data. If there are no duplicate data points, the rank of the covariance matrix $\boldsymbol{S}$ is N, so it has $D - N + 1$ many eigenvalues that are 0. Intuitively, this means that there are some redundancies. In the following, we will exploit this and turn the $D \times D$ covariance matrix into an $N \times N$ covariance matrix whose eigenvalues are all positive.

In PCA, we ended up with the eigenvector equation

$$\boldsymbol{S}\boldsymbol{b}_m = \lambda_m \boldsymbol{b}_m, \quad m = 1, \ldots, M, \tag{10.54}$$

where $\boldsymbol{b}_m$ is a basis vector of the principal subspace. Let us rewrite this equation a bit: With $\boldsymbol{S}$ defined in (10.53), we obtain

$$\boldsymbol{S}\boldsymbol{b}_m = \frac{1}{N}\boldsymbol{X}\boldsymbol{X}^\top \boldsymbol{b}_m = \lambda_m \boldsymbol{b}_m . \tag{10.55}$$

We now multiply $\boldsymbol{X}^\top \in \mathbb{R}^{N \times D}$ from the left-hand side, which yields

$$\frac{1}{N}\underbrace{\boldsymbol{X}^\top \boldsymbol{X}}_{N \times N}\underbrace{\boldsymbol{X}^\top \boldsymbol{b}_m}_{=:\boldsymbol{c}_m} = \lambda_m \boldsymbol{X}^\top \boldsymbol{b}_m \iff \frac{1}{N}\boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{c}_m = \lambda_m \boldsymbol{c}_m , \tag{10.56}$$

and we get a new eigenvector/eigenvalue equation: λ_m remains eigenvalue, which confirms our results from Section 4.5.3 that the nonzero eigenvalues of $\boldsymbol{X}\boldsymbol{X}^\top$ equal the nonzero eigenvalues of $\boldsymbol{X}^\top\boldsymbol{X}$. We obtain the eigenvector of the matrix $\frac{1}{N}\boldsymbol{X}^\top\boldsymbol{X} \in \mathbb{R}^{N\times N}$ associated with λ_m as $\boldsymbol{c}_m := \boldsymbol{X}^\top\boldsymbol{b}_m$. Assuming we have no duplicate data points, this matrix has rank N and is invertible. This also implies that $\frac{1}{N}\boldsymbol{X}^\top\boldsymbol{X}$ has the same (nonzero) eigenvalues as the data covariance matrix $\boldsymbol{S}$. But this is now an $N \times N$ matrix, so that we can compute the eigenvalues and eigenvectors much more efficiently than for the original $D \times D$ data covariance matrix.

Now that we have the eigenvectors of $\frac{1}{N}\boldsymbol{X}^\top\boldsymbol{X}$, we are going to recover the original eigenvectors, which we still need for PCA. Currently, we know the eigenvectors of $\frac{1}{N}\boldsymbol{X}^\top\boldsymbol{X}$. If we left-multiply our eigenvalue/eigenvector equation with $\boldsymbol{X}$, we get

$$\underbrace{\frac{1}{N}\boldsymbol{X}\boldsymbol{X}^\top}_{\boldsymbol{S}}\boldsymbol{X}\boldsymbol{c}_m = \lambda_m\boldsymbol{X}\boldsymbol{c}_m \tag{10.57}$$

and we recover the data covariance matrix again. This now also means that we recover $\boldsymbol{X}\boldsymbol{c}_m$ as an eigenvector of $\boldsymbol{S}$.

Remark. If we want to apply the PCA algorithm that we discussed in Section 10.6, we need to normalize the eigenvectors $\boldsymbol{X}\boldsymbol{c}_m$ of $\boldsymbol{S}$ so that they have norm 1. ♢

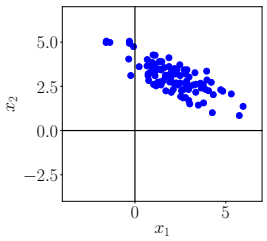
10.6 Key Steps of PCA in Practice

In the following, we will go through the individual steps of PCA using a running example, which is summarized in Figure 10.11. We are given a two-dimensional dataset (Figure 10.11(a)), and we want to use PCA to project it onto a one-dimensional subspace.

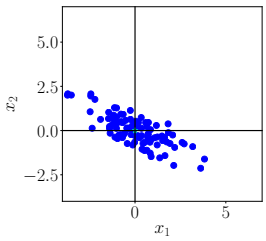
1. **Mean subtraction** We start by centering the data by computing the mean $\boldsymbol{\mu}$ of the dataset and subtracting it from every single data point. This ensures that the dataset has mean $\mathbf{0}$ (Figure 10.11(b)). Mean subtraction is not strictly necessary but reduces the risk of numerical problems.
2. **Standardization** Divide the data points by the standard deviation σ_d of the dataset for every dimension $d = 1, \ldots, D$. Now the data is unit free, and it has variance 1 along each axis, which is indicated by the two arrows in Figure 10.11(c). This step completes the *standardization* of the data.

standardization

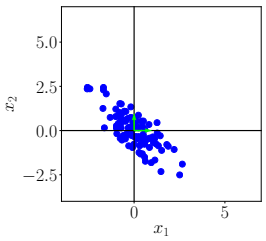
3. **Eigendecomposition of the covariance matrix** Compute the data covariance matrix and its eigenvalues and corresponding eigenvectors. Since the covariance matrix is symmetric, the spectral theorem (Theorem 4.15) states that we can find an ONB of eigenvectors. In Figure 10.11(d), the eigenvectors are scaled by the magnitude of the cor-



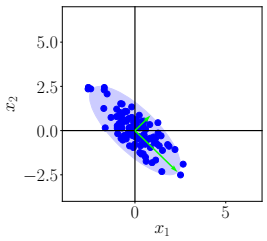
(a) Original dataset.



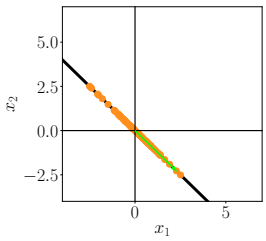
(b) Step 1: Centering by subtracting the mean from each data point.



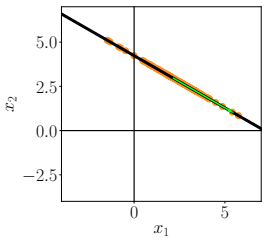
(c) Step 2: Dividing by the standard deviation to make the data unit free. Data has variance 1 along each axis.



(d) Step 3: Compute eigenvalues and eigenvectors (arrows) of the data covariance matrix (ellipse).



(e) Step 4: Project data onto the principal subspace.



(f) Undo the standardization and move projected data back into the original data space from (a).

Figure 10.11 Steps of PCA. (a) Original dataset; (b) centering; (c) divide by standard deviation; (d) eigendecomposition; (e) projection; (f) mapping back to original data space.

responding eigenvalue. The longer vector spans the principal subspace, which we denote by U. The data covariance matrix is represented by the ellipse.

4. **Projection** We can project any data point $\boldsymbol{x}_* \in \mathbb{R}^D$ onto the principal subspace: To get this right, we need to standardize $\boldsymbol{x}_*$ using the mean μ_d and standard deviation σ_d of the training data in the dth dimension, respectively, so that

$$x_*^{(d)} \leftarrow \frac{x_*^{(d)} - \mu_d}{\sigma_d}\,, \quad d = 1, \ldots, D\,, \tag{10.58}$$

where $x_*^{(d)}$ is the dth component of $\boldsymbol{x}_*$. We obtain the projection as

$$\tilde{\boldsymbol{x}}_* = \boldsymbol{B}\boldsymbol{B}^\top \boldsymbol{x}_* \tag{10.59}$$

with coordinates

$$\boldsymbol{z}_* = \boldsymbol{B}^\top \boldsymbol{x}_* \tag{10.60}$$

with respect to the basis of the principal subspace. Here, $\boldsymbol{B}$ is the matrix that contains the eigenvectors that are associated with the largest eigenvalues of the data covariance matrix as columns. PCA returns the coordinates (10.60), not the projections $\boldsymbol{x}_*$.

Having standardized our dataset, (10.59) only yields the projections in the context of the standardized dataset. To obtain our projection in the original data space (i.e., before standardization), we need to undo the standardization (10.58) and multiply by the standard deviation before adding the mean so that we obtain

$$\tilde{x}_*^{(d)} \leftarrow \tilde{x}_*^{(d)}\sigma_d + \mu_d\,, \quad d = 1, \ldots, D\,. \tag{10.61}$$

Figure 10.11(f) illustrates the projection in the original data space.

Example 10.4 (MNIST Digits: Reconstruction)
In the following, we will apply PCA to the MNIST digits dataset, which contains $60{,}000$ examples of handwritten digits 0 through 9. Each digit is an image of size 28×28, i.e., it contains 784 pixels so that we can interpret every image in this dataset as a vector $\boldsymbol{x} \in \mathbb{R}^{784}$. Examples of these digits are shown in Figure 10.3.

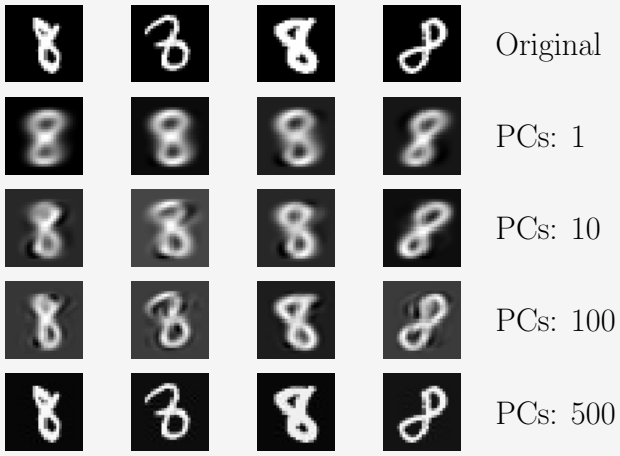


Figure 10.12 Effect of increasing the number of principal components on reconstruction.

For illustration purposes, we apply PCA to a subset of the MNIST digits, and we focus on the digit "8". We used 5,389 training images of the digit "8" and determined the principal subspace as detailed in this chapter. We then used the learned projection matrix to reconstruct a set of test images, which is illustrated in Figure 10.12. The first row of Figure 10.12 shows a set of four original digits from the test set. The following rows show reconstructions of exactly these digits when using a principal subspace of dimensions 1, 10, 100, and 500, respectively. We see that even with a single-dimensional principal subspace we get a halfway decent reconstruction of the original digits, which, however, is blurry and generic. With an increasing number of principal components (PCs), the reconstructions become sharper and more details are accounted for. With 500 prin-

cipal components, we effectively obtain a near-perfect reconstruction. If we were to choose 784 PCs, we would recover the exact digit without any compression loss.

Figure 10.13 shows the average squared reconstruction error, which is

$$\frac{1}{N}\sum_{n=1}^{N}\|\boldsymbol{x}_n - \tilde{\boldsymbol{x}}_n\|^2 = \sum_{i=M+1}^{D}\lambda_i\,, \tag{10.62}$$

as a function of the number M of principal components. We can see that the importance of the principal components drops off rapidly, and only marginal gains can be achieved by adding more PCs. This matches exactly our observation in Figure 10.5, where we discovered that the most of the variance of the projected data is captured by only a few principal components. With about 550 PCs, we can essentially fully reconstruct the training data that contains the digit "8" (some pixels around the boundaries show no variation across the dataset as they are always black).

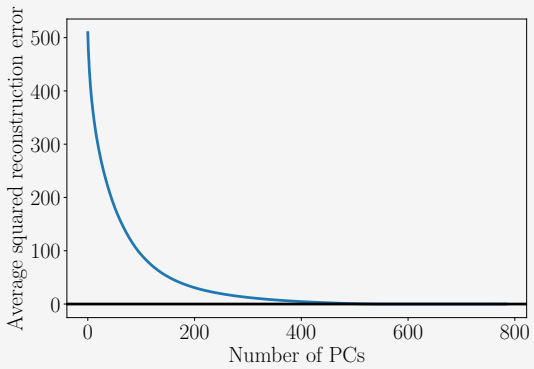


Figure 10.13 Average squared reconstruction error as a function of the number of principal components. The average squared reconstruction error is the sum of the eigenvalues in the orthogonal complement of the principal subspace.

10.7 Latent Variable Perspective

In the previous sections, we derived PCA without any notion of a probabilistic model using the maximum-variance and the projection perspectives. On the one hand, this approach may be appealing as it allows us to sidestep all the mathematical difficulties that come with probability theory, but on the other hand, a probabilistic model would offer us more flexibility and useful insights. More specifically, a probabilistic model would

- Come with a likelihood function, and we can explicitly deal with noisy observations (which we did not even discuss earlier)
- Allow us to do Bayesian model comparison via the marginal likelihood as discussed in Section 8.6
- View PCA as a generative model, which allows us to simulate new data

- Allow us to make straightforward connections to related algorithms
- Deal with data dimensions that are missing at random by applying Bayes' theorem
- Give us a notion of the novelty of a new data point
- Give us a principled way to extend the model, e.g., to a mixture of PCA models
- Have the PCA we derived in earlier sections as a special case
- Allow for a fully Bayesian treatment by marginalizing out the model parameters

probabilistic PCA
PPCA

By introducing a continuous-valued latent variable $\boldsymbol{z} \in \mathbb{R}^M$ it is possible to phrase PCA as a probabilistic latent-variable model. Tipping and Bishop (1999) proposed this latent-variable model as *probabilistic PCA* (*PPCA*). PPCA addresses most of the aforementioned issues, and the PCA solution that we obtained by maximizing the variance in the projected space or by minimizing the reconstruction error is obtained as the special case of maximum likelihood estimation in a noise-free setting.

10.7.1 Generative Process and Probabilistic Model

In PPCA, we explicitly write down the probabilistic model for linear dimensionality reduction. For this we assume a continuous latent variable $\boldsymbol{z} \in \mathbb{R}^M$ with a standard-normal prior $p(\boldsymbol{z}) = \mathcal{N}\left(\boldsymbol{0}, \boldsymbol{I}\right)$ and a linear relationship between the latent variables and the observed $\boldsymbol{x}$ data where

$$\boldsymbol{x} = \boldsymbol{B}\boldsymbol{z} + \boldsymbol{\mu} + \boldsymbol{\epsilon} \in \mathbb{R}^D\,, \tag{10.63}$$

where $\boldsymbol{\epsilon} \sim \mathcal{N}\left(\boldsymbol{0}, \sigma^2\boldsymbol{I}\right)$ is Gaussian observation noise and $\boldsymbol{B} \in \mathbb{R}^{D\times M}$ and $\boldsymbol{\mu} \in \mathbb{R}^D$ describe the linear/affine mapping from latent to observed variables. Therefore, PPCA links latent and observed variables via

$$p(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{B}, \boldsymbol{\mu}, \sigma^2) = \mathcal{N}\left(\boldsymbol{x} \,|\, \boldsymbol{B}\boldsymbol{z} + \boldsymbol{\mu},\, \sigma^2\boldsymbol{I}\right). \tag{10.64}$$

Overall, PPCA induces the following generative process:

$$\boldsymbol{z}_n \sim \mathcal{N}\left(\boldsymbol{z} \,|\, \boldsymbol{0},\, \boldsymbol{I}\right) \tag{10.65}$$

$$\boldsymbol{x}_n \,|\, \boldsymbol{z}_n \sim \mathcal{N}\left(\boldsymbol{x} \,|\, \boldsymbol{B}\boldsymbol{z}_n + \boldsymbol{\mu},\, \sigma^2\boldsymbol{I}\right) \tag{10.66}$$

ancestral sampling

To generate a data point that is typical given the model parameters, we follow an *ancestral sampling* scheme: We first sample a latent variable $\boldsymbol{z}_n$ from $p(\boldsymbol{z})$. Then we use $\boldsymbol{z}_n$ in (10.64) to sample a data point conditioned on the sampled $\boldsymbol{z}_n$, i.e., $\boldsymbol{x}_n \sim p(\boldsymbol{x} \,|\, \boldsymbol{z}_n, \boldsymbol{B}, \boldsymbol{\mu}, \sigma^2)$.

This generative process allows us to write down the probabilistic model (i.e., the joint distribution of all random variables; see Section 8.4) as

$$p(\boldsymbol{x}, \boldsymbol{z}|\boldsymbol{B}, \boldsymbol{\mu}, \sigma^2) = p(\boldsymbol{x}|\boldsymbol{z}, \boldsymbol{B}, \boldsymbol{\mu}, \sigma^2)p(\boldsymbol{z})\,, \tag{10.67}$$

which immediately gives rise to the graphical model in Figure 10.14 using the results from Section 8.5.

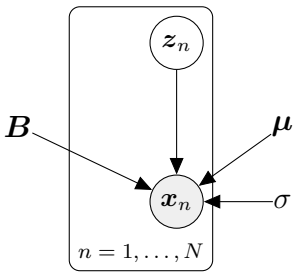


Figure 10.14 Graphical model for probabilistic PCA. The observations $\boldsymbol{x}_n$ explicitly depend on corresponding latent variables $\boldsymbol{z}_n \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$. The model parameters $\boldsymbol{B}, \boldsymbol{\mu}$ and the likelihood parameter σ are shared across the dataset.

Remark. Note the direction of the arrow that connects the latent variables $\boldsymbol{z}$ and the observed data $\boldsymbol{x}$: The arrow points from $\boldsymbol{z}$ to $\boldsymbol{x}$, which means that the PPCA model assumes a lower-dimensional latent cause $\boldsymbol{z}$ for high-dimensional observations $\boldsymbol{x}$. In the end, we are obviously interested in finding something out about $\boldsymbol{z}$ given some observations. To get there we will apply Bayesian inference to "invert" the arrow implicitly and go from observations to latent variables. ♢

Example 10.5 (Generating New Data Using Latent Variables)

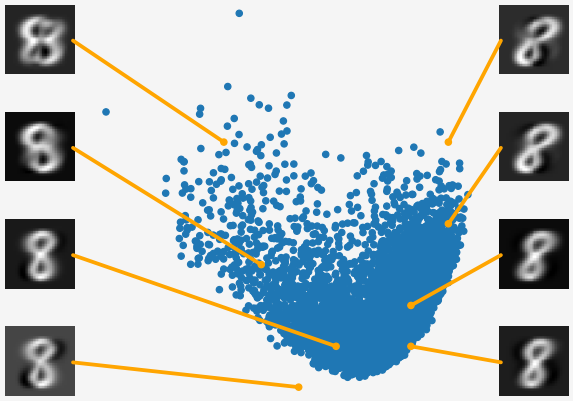


Figure 10.15 Generating new MNIST digits. The latent variables $\boldsymbol{z}$ can be used to generate new data $\tilde{\boldsymbol{x}} = \boldsymbol{B}\boldsymbol{z}$. The closer we stay to the training data, the more realistic the generated data.

Figure 10.15 shows the latent coordinates of the MNIST digits "8" found by PCA when using a two-dimensional principal subspace (blue dots). We can query any vector $\boldsymbol{z}_*$ in this latent space and generate an image $\tilde{\boldsymbol{x}}_* = \boldsymbol{B}\boldsymbol{z}_*$ that resembles the digit "8". We show eight of such generated images with their corresponding latent space representation. Depending on where we query the latent space, the generated images look different (shape, rotation, size, etc.). If we query away from the training data, we see more and more artifacts, e.g., the top-left and top-right digits. Note that the intrinsic dimensionality of these generated images is only two.

10.7.2 Likelihood and Joint Distribution

The likelihood does not depend on the latent variables $\boldsymbol{z}$.

Using the results from Chapter 6, we obtain the likelihood of this probabilistic model by integrating out the latent variable $\boldsymbol{z}$ (see Section 8.4.3) so that

$$p(\boldsymbol{x} \mid \boldsymbol{B}, \boldsymbol{\mu}, \sigma^2) = \int p(\boldsymbol{x} \mid \boldsymbol{z}, \boldsymbol{B}, \boldsymbol{\mu}, \sigma^2) p(\boldsymbol{z}) \mathrm{d}\boldsymbol{z} \tag{10.68a}$$

$$= \int \mathcal{N}\big(\boldsymbol{x} \mid \boldsymbol{B}\boldsymbol{z} + \boldsymbol{\mu},\, \sigma^2 \boldsymbol{I}\big) \mathcal{N}\big(\boldsymbol{z} \mid \boldsymbol{0},\, \boldsymbol{I}\big) \mathrm{d}\boldsymbol{z}\,. \tag{10.68b}$$

From Section 6.5, we know that the solution to this integral is a Gaussian distribution with mean

$$\mathbb{E}_{\boldsymbol{x}}[\boldsymbol{x}] = \mathbb{E}_{\boldsymbol{z}}[\boldsymbol{B}\boldsymbol{z} + \boldsymbol{\mu}] + \mathbb{E}_{\boldsymbol{\epsilon}}[\boldsymbol{\epsilon}] = \boldsymbol{\mu} \tag{10.69}$$

and with covariance matrix

$$\mathbb{V}[\boldsymbol{x}] = \mathbb{V}_{\boldsymbol{z}}[\boldsymbol{B}\boldsymbol{z} + \boldsymbol{\mu}] + \mathbb{V}_{\boldsymbol{\epsilon}}[\boldsymbol{\epsilon}] = \mathbb{V}_{\boldsymbol{z}}[\boldsymbol{B}\boldsymbol{z}] + \sigma^2 \boldsymbol{I} \tag{10.70a}$$

$$= \boldsymbol{B}\mathbb{V}_{\boldsymbol{z}}[\boldsymbol{z}]\boldsymbol{B}^\top + \sigma^2 \boldsymbol{I} = \boldsymbol{B}\boldsymbol{B}^\top + \sigma^2 \boldsymbol{I}\,. \tag{10.70b}$$

The likelihood in (10.68b) can be used for maximum likelihood or MAP estimation of the model parameters.

Remark. We cannot use the conditional distribution in (10.64) for maximum likelihood estimation as it still depends on the latent variables. The likelihood function we require for maximum likelihood (or MAP) estimation should only be a function of the data $\boldsymbol{x}$ and the model parameters, but must not depend on the latent variables. ♢

From Section 6.5, we know that a Gaussian random variable $\boldsymbol{z}$ and a linear/affine transformation $\boldsymbol{x} = \boldsymbol{B}\boldsymbol{z}$ of it are jointly Gaussian distributed. We already know the marginals $p(\boldsymbol{z}) = \mathcal{N}\big(\boldsymbol{z} \mid \boldsymbol{0},\, \boldsymbol{I}\big)$ and $p(\boldsymbol{x}) = \mathcal{N}\big(\boldsymbol{x} \mid \boldsymbol{\mu},\, \boldsymbol{B}\boldsymbol{B}^\top + \sigma^2 \boldsymbol{I}\big)$. The missing cross-covariance is given as

$$\mathrm{Cov}[\boldsymbol{x}, \boldsymbol{z}] = \mathrm{Cov}_{\boldsymbol{z}}[\boldsymbol{B}\boldsymbol{z} + \boldsymbol{\mu}] = \boldsymbol{B}\,\mathrm{Cov}_{\boldsymbol{z}}[\boldsymbol{z}, \boldsymbol{z}] = \boldsymbol{B}\,. \tag{10.71}$$

Therefore, the probabilistic model of PPCA, i.e., the joint distribution of latent and observed random variables is explicitly given by

$$p(\boldsymbol{x}, \boldsymbol{z} \mid \boldsymbol{B}, \boldsymbol{\mu}, \sigma^2) = \mathcal{N}\left(\begin{bmatrix}\boldsymbol{x}\\ \boldsymbol{z}\end{bmatrix} \,\middle|\, \begin{bmatrix}\boldsymbol{\mu}\\ \boldsymbol{0}\end{bmatrix}, \begin{bmatrix}\boldsymbol{B}\boldsymbol{B}^\top + \sigma^2 \boldsymbol{I} & \boldsymbol{B}\\ \boldsymbol{B}^\top & \boldsymbol{I}\end{bmatrix}\right), \tag{10.72}$$

with a mean vector of length $D + M$ and a covariance matrix of size $(D + M) \times (D + M)$.

10.7.3 Posterior Distribution

The joint Gaussian distribution $p(\boldsymbol{x}, \boldsymbol{z} \mid \boldsymbol{B}, \boldsymbol{\mu}, \sigma^2)$ in (10.72) allows us to determine the posterior distribution $p(\boldsymbol{z} \mid \boldsymbol{x})$ immediately by applying the

rules of Gaussian conditioning from Section 6.5.1. The posterior distribution of the latent variable given an observation $\boldsymbol{x}$ is then

$$p(\boldsymbol{z} \mid \boldsymbol{x}) = \mathcal{N}\big(\boldsymbol{z} \mid \boldsymbol{m}, \boldsymbol{C}\big)\,, \tag{10.73}$$

$$\boldsymbol{m} = \boldsymbol{B}^\top(\boldsymbol{B}\boldsymbol{B}^\top + \sigma^2\boldsymbol{I})^{-1}(\boldsymbol{x} - \boldsymbol{\mu})\,, \tag{10.74}$$

$$\boldsymbol{C} = \boldsymbol{I} - \boldsymbol{B}^\top(\boldsymbol{B}\boldsymbol{B}^\top + \sigma^2\boldsymbol{I})^{-1}\boldsymbol{B}\,. \tag{10.75}$$

Note that the posterior covariance does not depend on the observed data $\boldsymbol{x}$. For a new observation $\boldsymbol{x}_*$ in data space, we use (10.73) to determine the posterior distribution of the corresponding latent variable $\boldsymbol{z}_*$. The covariance matrix $\boldsymbol{C}$ allows us to assess how confident the embedding is. A covariance matrix $\boldsymbol{C}$ with a small determinant (which measures volumes) tells us that the latent embedding $\boldsymbol{z}_*$ is fairly certain. If we obtain a posterior distribution $p(\boldsymbol{z}_* \mid \boldsymbol{x}_*)$ with much variance, we may be faced with an outlier. However, we can explore this posterior distribution to understand what other data points $\boldsymbol{x}$ are plausible under this posterior. To do this, we exploit the generative process underlying PPCA, which allows us to explore the posterior distribution on the latent variables by generating new data that is plausible under this posterior:

1. Sample a latent variable $\boldsymbol{z}_* \sim p(\boldsymbol{z} \mid \boldsymbol{x}_*)$ from the posterior distribution over the latent variables (10.73).
2. Sample a reconstructed vector $\tilde{\boldsymbol{x}}_* \sim p(\boldsymbol{x} \mid \boldsymbol{z}_*, \boldsymbol{B}, \boldsymbol{\mu}, \sigma^2)$ from (10.64).

If we repeat this process many times, we can explore the posterior distribution (10.73) on the latent variables $\boldsymbol{z}_*$ and its implications on the observed data. The sampling process effectively hypothesizes data, which is plausible under the posterior distribution.

10.8 Further Reading

We derived PCA from two perspectives: (a) maximizing the variance in the projected space; (b) minimizing the average reconstruction error. However, PCA can also be interpreted from different perspectives. Let us recap what we have done: We took high-dimensional data $\boldsymbol{x} \in \mathbb{R}^D$ and used a matrix $\boldsymbol{B}^\top$ to find a lower-dimensional representation $\boldsymbol{z} \in \mathbb{R}^M$. The columns of $\boldsymbol{B}$ are the eigenvectors of the data covariance matrix $\boldsymbol{S}$ that are associated with the largest eigenvalues. Once we have a low-dimensional representation $\boldsymbol{z}$, we can get a high-dimensional version of it (in the original data space) as $\boldsymbol{x} \approx \tilde{\boldsymbol{x}} = \boldsymbol{B}\boldsymbol{z} = \boldsymbol{B}\boldsymbol{B}^\top\boldsymbol{x} \in \mathbb{R}^D$, where $\boldsymbol{B}\boldsymbol{B}^\top$ is a projection matrix.

We can also think of PCA as a linear *auto-encoder* as illustrated in Figure 10.16. An auto-encoder encodes the data $\boldsymbol{x}_n \in \mathbb{R}^D$ to a *code* $\boldsymbol{z}_n \in \mathbb{R}^M$ and decodes it to a $\tilde{\boldsymbol{x}}_n$ similar to $\boldsymbol{x}_n$. The mapping from the data to the code is called the *encoder*, and the mapping from the code back to the original data space is called the *decoder*. If we consider linear mappings where

auto-encoder
code
encoder
decoder

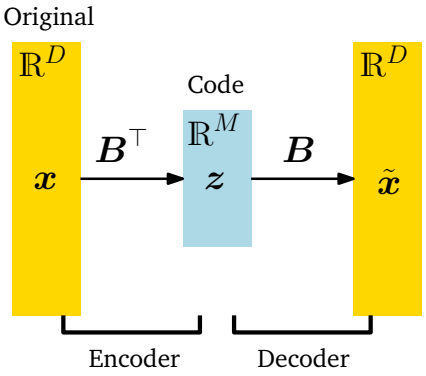


Figure 10.16 PCA can be viewed as a linear auto-encoder. It encodes the high-dimensional data $\boldsymbol{x}$ into a lower-dimensional representation (code) $\boldsymbol{z} \in \mathbb{R}^M$ and decodes $\boldsymbol{z}$ using a decoder. The decoded vector $\tilde{\boldsymbol{x}}$ is the orthogonal projection of the original data $\boldsymbol{x}$ onto the M-dimensional principal subspace.

the code is given by $\boldsymbol{z}_n = \boldsymbol{B}^\top \boldsymbol{x}_n \in \mathbb{R}^M$ and we are interested in minimizing the average squared error between the data $\boldsymbol{x}_n$ and its reconstruction $\tilde{\boldsymbol{x}}_n = \boldsymbol{B}\boldsymbol{z}_n$, $n = 1, \ldots, N$, we obtain

$$\frac{1}{N}\sum_{n=1}^{N} \|\boldsymbol{x}_n - \tilde{\boldsymbol{x}}_n\|^2 = \frac{1}{N}\sum_{n=1}^{N} \left\|\boldsymbol{x}_n - \boldsymbol{B}\boldsymbol{B}^\top \boldsymbol{x}_n\right\|^2 . \tag{10.76}$$

This means we end up with the same objective function as in (10.29) that we discussed in Section 10.3 so that we obtain the PCA solution when we minimize the squared auto-encoding loss. If we replace the linear mapping of PCA with a nonlinear mapping, we get a nonlinear auto-encoder. A prominent example of this is a deep auto-encoder where the linear functions are replaced with deep neural networks. In this context, the encoder is also known as a *recognition network* or *inference network*, whereas the decoder is also called a *generator*.

recognition network
inference network
generator

Another interpretation of PCA is related to information theory. We can think of the code as a smaller or compressed version of the original data point. When we reconstruct our original data using the code, we do not get the exact data point back, but a slightly distorted or noisy version of it. This means that our compression is "lossy". Intuitively, we want to maximize the correlation between the original data and the lower-dimensional code. More formally, this is related to the mutual information. We would then get the same solution to PCA we discussed in Section 10.3 by maximizing the mutual information, a core concept in information theory (MacKay, 2003).

The code is a compressed version of the original data.

In our discussion on PPCA, we assumed that the parameters of the model, i.e., $\boldsymbol{B}, \boldsymbol{\mu}$, and the likelihood parameter σ^2, are known. Tipping and Bishop (1999) describe how to derive maximum likelihood estimates for these parameters in the PPCA setting (note that we use a different notation in this chapter). The maximum likelihood parameters, when pro-

jecting D-dimensional data onto an M-dimensional subspace, are

$$\boldsymbol{\mu}_{\mathrm{ML}} = \frac{1}{N}\sum_{n=1}^{N}\boldsymbol{x}_n\,, \tag{10.77}$$

$$\boldsymbol{B}_{\mathrm{ML}} = \boldsymbol{T}(\boldsymbol{\Lambda} - \sigma^2\boldsymbol{I})^{\frac{1}{2}}\boldsymbol{R}\,, \tag{10.78}$$

$$\sigma^2_{\mathrm{ML}} = \frac{1}{D-M}\sum_{j=M+1}^{D}\lambda_j\,, \tag{10.79}$$

where $\boldsymbol{T} \in \mathbb{R}^{D\times M}$ contains M eigenvectors of the data covariance matrix, $\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1,\ldots,\lambda_M) \in \mathbb{R}^{M\times M}$ is a diagonal matrix with the eigenvalues associated with the principal axes on its diagonal, and $\boldsymbol{R} \in \mathbb{R}^{M\times M}$ is an arbitrary orthogonal matrix. The maximum likelihood solution $\boldsymbol{B}_{\mathrm{ML}}$ is unique up to an arbitrary orthogonal transformation, e.g., we can right-multiply $\boldsymbol{B}_{\mathrm{ML}}$ with any rotation matrix $\boldsymbol{R}$ so that (10.78) essentially is a singular value decomposition (see Section 4.5). An outline of the proof is given by Tipping and Bishop (1999).

The matrix $\boldsymbol{\Lambda} - \sigma^2\boldsymbol{I}$ in (10.78) is guaranteed to be positive semidefinite as the smallest eigenvalue of the data covariance matrix is bounded from below by the noise variance σ^2.

The maximum likelihood estimate for $\boldsymbol{\mu}$ given in (10.77) is the sample mean of the data. The maximum likelihood estimator for the observation noise variance σ^2 given in (10.79) is the average variance in the orthogonal complement of the principal subspace, i.e., the average leftover variance that we cannot capture with the first M principal components is treated as observation noise.

In the noise-free limit where $\sigma \to 0$, PPCA and PCA provide identical solutions: Since the data covariance matrix $\boldsymbol{S}$ is symmetric, it can be diagonalized (see Section 4.4), i.e., there exists a matrix $\boldsymbol{T}$ of eigenvectors of $\boldsymbol{S}$ so that

$$\boldsymbol{S} = \boldsymbol{T}\boldsymbol{\Lambda}\boldsymbol{T}^{-1}\,. \tag{10.80}$$

In the PPCA model, the data covariance matrix is the covariance matrix of the Gaussian likelihood $p(\boldsymbol{x}\,|\,\boldsymbol{B},\boldsymbol{\mu},\sigma^2)$, which is $\boldsymbol{B}\boldsymbol{B}^\top + \sigma^2\boldsymbol{I}$, see (10.70b). For $\sigma \to 0$, we obtain $\boldsymbol{B}\boldsymbol{B}^\top$ so that this data covariance must equal the PCA data covariance (and its factorization given in (10.80)) so that

$$\mathrm{Cov}[\mathcal{X}] = \boldsymbol{T}\boldsymbol{\Lambda}\boldsymbol{T}^{-1} = \boldsymbol{B}\boldsymbol{B}^\top \iff \boldsymbol{B} = \boldsymbol{T}\boldsymbol{\Lambda}^{\frac{1}{2}}\boldsymbol{R}\,, \tag{10.81}$$

i.e., we obtain the maximum likelihood estimate in (10.78) for $\sigma = 0$. From (10.78) and (10.80), it becomes clear that (P)PCA performs a decomposition of the data covariance matrix.

In a streaming setting, where data arrives sequentially, it is recommended to use the iterative expectation maximization (EM) algorithm for maximum likelihood estimation (Roweis, 1998).

To determine the dimensionality of the latent variables (the length of the code, the dimensionality of the lower-dimensional subspace onto which we project the data), Gavish and Donoho (2014) suggest the heuristic that, if we can estimate the noise variance σ^2 of the data, we should

discard all singular values smaller than $\frac{4\sigma\sqrt{D}}{\sqrt{3}}$. Alternatively, we can use (nested) cross-validation (Section 8.6.1) or Bayesian model selection criteria (discussed in Section 8.6.2) to determine a good estimate of the intrinsic dimensionality of the data (Minka, 2001b).

Similar to our discussion on linear regression in Chapter 9, we can place a prior distribution on the parameters of the model and integrate them out. By doing so, we (a) avoid point estimates of the parameters and the issues that come with these point estimates (see Section 8.6) and (b) allow for an automatic selection of the appropriate dimensionality M of the latent space. In this *Bayesian PCA*, which was proposed by Bishop (1999), a prior $p(\boldsymbol{\mu}, \boldsymbol{B}, \sigma^2)$ is placed on the model parameters. The generative process allows us to integrate the model parameters out instead of conditioning on them, which addresses overfitting issues. Since this integration is analytically intractable, Bishop (1999) proposes to use approximate inference methods, such as MCMC or variational inference. We refer to the work by Gilks et al. (1996) and Blei et al. (2017) for more details on these approximate inference techniques.

Bayesian PCA

In PPCA, we considered the linear model $p(\boldsymbol{x}_n \mid \boldsymbol{z}_n) = \mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{B}\boldsymbol{z}_n + \boldsymbol{\mu}, \sigma^2\boldsymbol{I})$ with prior $p(\boldsymbol{z}_n) = \mathcal{N}(\boldsymbol{0}, \boldsymbol{I})$, where all observation dimensions are affected by the same amount of noise. If we allow each observation dimension d to have a different variance σ_d^2, we obtain *factor analysis* (FA) (Spearman, 1904; Bartholomew et al., 2011). This means that FA gives the likelihood some more flexibility than PPCA, but still forces the data to be explained by the model parameters $\boldsymbol{B}$, $\boldsymbol{\mu}$.However, FA no longer allows for a closed-form maximum likelihood solution so that we need to use an iterative scheme, such as the expectation maximization algorithm, to estimate the model parameters. While in PPCA all stationary points are global optima, this no longer holds for FA. Compared to PPCA, FA does not change if we scale the data, but it does return different solutions if we rotate the data.

factor analysis

An overly flexible likelihood would be able to explain more than just the noise.

An algorithm that is also closely related to PCA is *independent component analysis* (*ICA* (Hyvarinen et al., 2001)). Starting again with the latent-variable perspective $p(\boldsymbol{x}_n \mid \boldsymbol{z}_n) = \mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{B}\boldsymbol{z}_n + \boldsymbol{\mu}, \sigma^2\boldsymbol{I})$ we now change the prior on $\boldsymbol{z}_n$ to non-Gaussian distributions. ICA can be used for *blind-source separation*. Imagine you are in a busy train station with many people talking. Your ears play the role of microphones, and they linearly mix different speech signals in the train station. The goal of blind-source separation is to identify the constituent parts of the mixed signals. As discussed previously in the context of maximum likelihood estimation for PPCA, the original PCA solution is invariant to any rotation. Therefore, PCA can identify the best lower-dimensional subspace in which the signals live, but not the signals themselves (Murphy, 2012). ICA addresses this issue by modifying the prior distribution $p(\boldsymbol{z})$ on the latent sources

independent component analysis
ICA

blind-source separation

to require non-Gaussian priors $p(\boldsymbol{z})$. We refer to the books by Hyvarinen et al. (2001) and Murphy (2012) for more details on ICA.

PCA, factor analysis, and ICA are three examples for dimensionality reduction with linear models. Cunningham and Ghahramani (2015) provide a broader survey of linear dimensionality reduction.

The (P)PCA model we discussed here allows for several important extensions. In Section 10.5, we explained how to do PCA when the input dimensionality D is significantly greater than the number N of data points. By exploiting the insight that PCA can be performed by computing (many) inner products, this idea can be pushed to the extreme by considering infinite-dimensional features. The *kernel trick* is the basis of *kernel PCA* and allows us to implicitly compute inner products between infinite-dimensional features (Schölkopf et al., 1998; Schölkopf and Smola, 2002).

kernel trick
kernel PCA

There are nonlinear dimensionality reduction techniques that are derived from PCA (Burges (2010) provides a good overview). The auto-encoder perspective of PCA that we discussed previously in this section can be used to render PCA as a special case of a *deep auto-encoder*. In the deep auto-encoder, both the encoder and the decoder are represented by multilayer feedforward neural networks, which themselves are nonlinear mappings. If we set the activation functions in these neural networks to be the identity, the model becomes equivalent to PCA. A different approach to nonlinear dimensionality reduction is the *Gaussian process latent-variable model* (*GP-LVM*) proposed by Lawrence (2005). The GP-LVM starts off with the latent-variable perspective that we used to derive PPCA and replaces the linear relationship between the latent variables $\boldsymbol{z}$ and the observations $\boldsymbol{x}$ with a Gaussian process (GP). Instead of estimating the parameters of the mapping (as we do in PPCA), the GP-LVM marginalizes out the model parameters and makes point estimates of the latent variables $\boldsymbol{z}$. Similar to Bayesian PCA, the *Bayesian GP-LVM* proposed by Titsias and Lawrence (2010) maintains a distribution on the latent variables $\boldsymbol{z}$ and uses approximate inference to integrate them out as well.

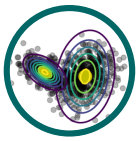
deep auto-encoder

Gaussian process latent-variable model
GP-LVM

Bayesian GP-LVM

11

Density Estimation with Gaussian Mixture Models



In earlier chapters, we covered already two fundamental problems in machine learning: regression (Chapter 9) and dimensionality reduction (Chapter 10). In this chapter, we will have a look at a third pillar of machine learning: density estimation. On our journey, we introduce important concepts, such as the expectation maximization (EM) algorithm and a latent variable perspective of density estimation with mixture models.

When we apply machine learning to data we often aim to represent data in some way. A straightforward way is to take the data points themselves as the representation of the data; see Figure 11.1 for an example. However, this approach may be unhelpful if the dataset is huge or if we are interested in representing characteristics of the data. In density estimation, we represent the data compactly using a density from a parametric family, e.g., a Gaussian or Beta distribution. For example, we may be looking for the mean and variance of a dataset in order to represent the data compactly using a Gaussian distribution. The mean and variance can be found using tools we discussed in Section 8.3: maximum likelihood or maximum a posteriori estimation. We can then use the mean and variance of this Gaussian to represent the distribution underlying the data, i.e., we think of the dataset to be a typical realization from this distribution if we were to sample from it.

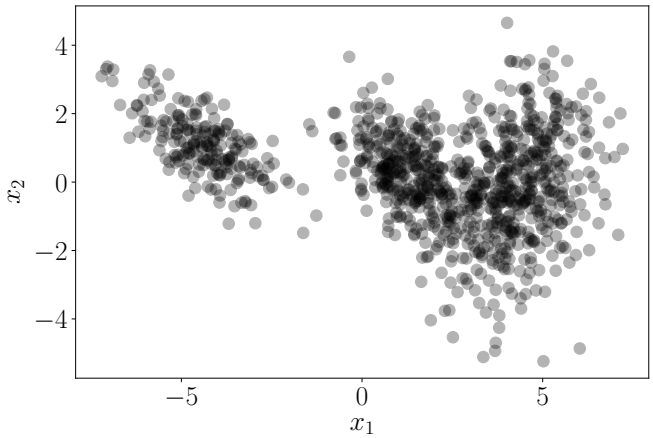


Figure 11.1 Two-dimensional dataset that cannot be meaningfully represented by a Gaussian.

In practice, the Gaussian (or similarly all other distributions we encountered so far) have limited modeling capabilities. For example, a Gaussian approximation of the density that generated the data in Figure 11.1 would be a poor approximation. In the following, we will look at a more expressive family of distributions, which we can use for density estimation: *mixture models*.

mixture model

Mixture models can be used to describe a distribution $p(\boldsymbol{x})$ by a convex combination of K simple (base) distributions

$$p(\boldsymbol{x}) = \sum_{k=1}^{K} \pi_k p_k(\boldsymbol{x}) \tag{11.1}$$

$$0 \leqslant \pi_k \leqslant 1\,, \quad \sum_{k=1}^{K} \pi_k = 1\,, \tag{11.2}$$

where the components p_k are members of a family of basic distributions, e.g., Gaussians, Bernoullis, or Gammas, and the π_k are *mixture weights*. Mixture models are more expressive than the corresponding base distributions because they allow for multimodal data representations, i.e., they can describe datasets with multiple "clusters", such as the example in Figure 11.1.

mixture weight

We will focus on Gaussian mixture models (GMMs), where the basic distributions are Gaussians. For a given dataset, we aim to maximize the likelihood of the model parameters to train the GMM. For this purpose, we will use results from Chapter 5, Chapter 6, and Section 7.2. However, unlike other applications we discussed earlier (linear regression or PCA), we will not find a closed-form maximum likelihood solution. Instead, we will arrive at a set of dependent simultaneous equations, which we can only solve iteratively.

11.1 Gaussian Mixture Model

A *Gaussian mixture model* is a density model where we combine a finite number of K Gaussian distributions $\mathcal{N}\big(\boldsymbol{x} \,|\, \boldsymbol{\mu}_k,\, \boldsymbol{\Sigma}_k\big)$ so that

Gaussian mixture model

$$p(\boldsymbol{x} \,|\, \boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k \mathcal{N}\big(\boldsymbol{x} \,|\, \boldsymbol{\mu}_k,\, \boldsymbol{\Sigma}_k\big) \tag{11.3}$$

$$0 \leqslant \pi_k \leqslant 1\,, \quad \sum_{k=1}^{K} \pi_k = 1\,, \tag{11.4}$$

where we defined $\boldsymbol{\theta} := \{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \pi_k : \; k = 1, \ldots, K\}$ as the collection of all parameters of the model. This convex combination of Gaussian distribution gives us significantly more flexibility for modeling complex densities than a simple Gaussian distribution (which we recover from (11.3) for $K = 1$). An illustration is given in Figure 11.2, displaying the weighted

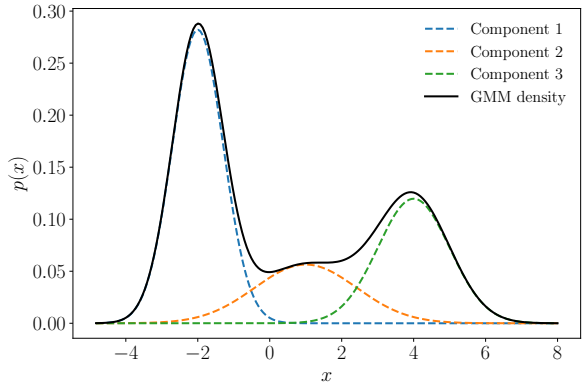


Figure 11.2 Gaussian mixture model. The Gaussian mixture distribution (black) is composed of a convex combination of Gaussian distributions and is more expressive than any individual component. Dashed lines represent the weighted Gaussian components.

components and the mixture density, which is given as

$$p(x \mid \boldsymbol{\theta}) = 0.5\mathcal{N}\left(x \mid -2, \tfrac{1}{2}\right) + 0.2\mathcal{N}\left(x \mid 1, 2\right) + 0.3\mathcal{N}\left(x \mid 4, 1\right). \quad (11.5)$$

11.2 Parameter Learning via Maximum Likelihood

Assume we are given a dataset $\mathcal{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$, where $\boldsymbol{x}_n$, $n = 1, \ldots, N$, are drawn i.i.d. from an unknown distribution $p(\boldsymbol{x})$. Our objective is to find a good approximation/representation of this unknown distribution $p(\boldsymbol{x})$ by means of a GMM with K mixture components. The parameters of the GMM are the K means $\boldsymbol{\mu}_k$, the covariances $\boldsymbol{\Sigma}_k$, and mixture weights π_k. We summarize all these free parameters in $\boldsymbol{\theta} := \{\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k : k = 1, \ldots, K\}$.

Example 11.1 (Initial Setting)

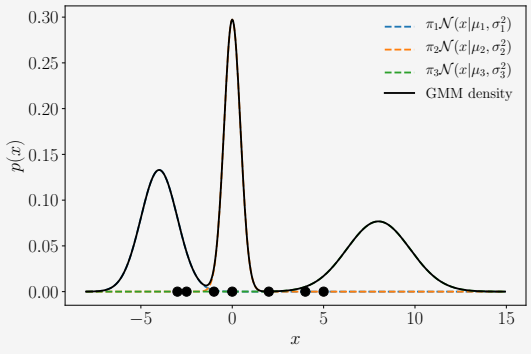


Figure 11.3 Initial setting: GMM (black) with mixture three mixture components (dashed) and seven data points (discs).

Throughout this chapter, we will have a simple running example that helps us illustrate and visualize important concepts.

We consider a one-dimensional dataset $\mathcal{X} = \{-3, -2.5, -1, 0, 2, 4, 5\}$ consisting of seven data points and wish to find a GMM with $K = 3$ components that models the density of the data. We initialize the mixture components as

$$p_1(x) = \mathcal{N}\big(x \,|\, -4,\, 1\big) \tag{11.6}$$

$$p_2(x) = \mathcal{N}\big(x \,|\, 0,\, 0.2\big) \tag{11.7}$$

$$p_3(x) = \mathcal{N}\big(x \,|\, 8,\, 3\big) \tag{11.8}$$

and assign them equal weights $\pi_1 = \pi_2 = \pi_3 = \frac{1}{3}$. The corresponding model (and the data points) are shown in Figure 11.3.

In the following, we detail how to obtain a maximum likelihood estimate $\boldsymbol{\theta}_{\text{ML}}$ of the model parameters $\boldsymbol{\theta}$. We start by writing down the likelihood, i.e., the predictive distribution of the training data given the parameters. We exploit our i.i.d. assumption, which leads to the factorized likelihood

$$p(\mathcal{X} \,|\, \boldsymbol{\theta}) = \prod_{n=1}^{N} p(\boldsymbol{x}_n \,|\, \boldsymbol{\theta}), \quad p(\boldsymbol{x}_n \,|\, \boldsymbol{\theta}) = \sum_{k=1}^{K} \pi_k \mathcal{N}\big(\boldsymbol{x}_n \,|\, \boldsymbol{\mu}_k,\, \boldsymbol{\Sigma}_k\big), \tag{11.9}$$

where every individual likelihood term $p(\boldsymbol{x}_n \,|\, \boldsymbol{\theta})$ is a Gaussian mixture density. Then we obtain the log-likelihood as

$$\log p(\mathcal{X} \,|\, \boldsymbol{\theta}) = \sum_{n=1}^{N} \log p(\boldsymbol{x}_n \,|\, \boldsymbol{\theta}) = \underbrace{\sum_{n=1}^{N} \log \sum_{k=1}^{K} \pi_k \mathcal{N}\big(\boldsymbol{x}_n \,|\, \boldsymbol{\mu}_k,\, \boldsymbol{\Sigma}_k\big)}_{=:\mathcal{L}}. \tag{11.10}$$

We aim to find parameters $\boldsymbol{\theta}^*_{\text{ML}}$ that maximize the log-likelihood $\mathcal{L}$ defined in (11.10). Our "normal" procedure would be to compute the gradient $\mathrm{d}\mathcal{L}/\mathrm{d}\boldsymbol{\theta}$ of the log-likelihood with respect to the model parameters $\boldsymbol{\theta}$, set it to $\mathbf{0}$, and solve for $\boldsymbol{\theta}$. However, unlike our previous examples for maximum likelihood estimation (e.g., when we discussed linear regression in Section 9.2), we cannot obtain a closed-form solution. However, we can exploit an iterative scheme to find good model parameters $\boldsymbol{\theta}_{\text{ML}}$, which will turn out to be the EM algorithm for GMMs. The key idea is to update one model parameter at a time while keeping the others fixed.

Remark. If we were to consider a single Gaussian as the desired density, the sum over k in (11.10) vanishes, and the $\log$ can be applied directly to the Gaussian component, such that we get

$$\log \mathcal{N}\big(\boldsymbol{x} \,|\, \boldsymbol{\mu},\, \boldsymbol{\Sigma}\big) = -\tfrac{D}{2}\log(2\pi) - \tfrac{1}{2}\log\det(\boldsymbol{\Sigma}) - \tfrac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\boldsymbol{x}-\boldsymbol{\mu}). \tag{11.11}$$

This simple form allows us to find closed-form maximum likelihood estimates of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$, as discussed in Chapter 8. In (11.10), we cannot move

the log into the sum over k so that we cannot obtain a simple closed-form maximum likelihood solution. ♢

Any local optimum of a function exhibits the property that its gradient with respect to the parameters must vanish (necessary condition); see Chapter 7. In our case, we obtain the following necessary conditions when we optimize the log-likelihood in (11.10) with respect to the GMM parameters $\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \pi_k$:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\mu}_k} = \mathbf{0}^\top \iff \sum_{n=1}^{N} \frac{\partial \log p(\boldsymbol{x}_n \mid \boldsymbol{\theta})}{\partial \boldsymbol{\mu}_k} = \mathbf{0}^\top, \tag{11.12}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\Sigma}_k} = \mathbf{0} \iff \sum_{n=1}^{N} \frac{\partial \log p(\boldsymbol{x}_n \mid \boldsymbol{\theta})}{\partial \boldsymbol{\Sigma}_k} = \mathbf{0}, \tag{11.13}$$

$$\frac{\partial \mathcal{L}}{\partial \pi_k} = 0 \iff \sum_{n=1}^{N} \frac{\partial \log p(\boldsymbol{x}_n \mid \boldsymbol{\theta})}{\partial \pi_k} = 0. \tag{11.14}$$

For all three necessary conditions, by applying the chain rule (see Section 5.2.2), we require partial derivatives of the form

$$\frac{\partial \log p(\boldsymbol{x}_n \mid \boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \frac{1}{p(\boldsymbol{x}_n \mid \boldsymbol{\theta})} \frac{\partial p(\boldsymbol{x}_n \mid \boldsymbol{\theta})}{\partial \boldsymbol{\theta}}, \tag{11.15}$$

where $\boldsymbol{\theta} = \{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \pi_k, k = 1, \dots, K\}$ are the model parameters and

$$\frac{1}{p(\boldsymbol{x}_n \mid \boldsymbol{\theta})} = \frac{1}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}. \tag{11.16}$$

In the following, we will compute the partial derivatives (11.12) through (11.14). But before we do this, we introduce a quantity that will play a central role in the remainder of this chapter: responsibilities.

11.2.1 Responsibilities

We define the quantity

$$r_{nk} := \frac{\pi_k \mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)} \tag{11.17}$$

responsibility

as the *responsibility* of the kth mixture component for the nth data point. The responsibility r_{nk} of the kth mixture component for data point $\boldsymbol{x}_n$ is proportional to the likelihood

$$p(\boldsymbol{x}_n \mid \pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) = \pi_k \mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \tag{11.18}$$

$\boldsymbol{r}_n$ follows a Boltzmann/Gibbs distribution.

of the mixture component given the data point. Therefore, mixture components have a high responsibility for a data point when the data point could be a plausible sample from that mixture component. Note that $\boldsymbol{r}_n := [r_{n1}, \dots, r_{nK}]^\top \in \mathbb{R}^K$ is a (normalized) probability vector, i.e.,

$\sum_k r_{nk} = 1$ with $r_{nk} \geqslant 0$. This probability vector distributes probability mass among the K mixture components, and we can think of $\boldsymbol{r}_n$ as a "soft assignment" of $\boldsymbol{x}_n$ to the K mixture components. Therefore, the responsibility r_{nk} from (11.17) represents the probability that $\boldsymbol{x}_n$ has been generated by the kth mixture component.

The responsibility r_{nk} is the probability that the kth mixture component generated the nth data point.

Example 11.2 (Responsibilities)
For our example from Figure 11.3, we compute the responsibilities r_{nk}

$$\begin{bmatrix} 1.0 & 0.0 & 0.0 \\ 1.0 & 0.0 & 0.0 \\ 0.057 & 0.943 & 0.0 \\ 0.001 & 0.999 & 0.0 \\ 0.0 & 0.066 & 0.934 \\ 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 1.0 \end{bmatrix} \in \mathbb{R}^{N \times K}. \tag{11.19}$$

Here the nth row tells us the responsibilities of all mixture components for x_n. The sum of all K responsibilities for a data point (sum of every row) is 1. The kth column gives us an overview of the responsibility of the kth mixture component. We can see that the third mixture component (third column) is not responsible for any of the first four data points, but takes much responsibility of the remaining data points. The sum of all entries of a column gives us the values N_k, i.e., the total responsibility of the kth mixture component. In our example, we get $N_1 = 2.058,\ N_2 = 2.008,\ N_3 = 2.934$.

In the following, we determine the updates of the model parameters $\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \pi_k$ for given responsibilities. We will see that the update equations all depend on the responsibilities, which makes a closed-form solution to the maximum likelihood estimation problem impossible. However, for given responsibilities we will be updating one model parameter at a time, while keeping the others fixed. After this, we will recompute the responsibilities. Iterating these two steps will eventually converge to a local optimum and is a specific instantiation of the EM algorithm. We will discuss this in some more detail in Section 11.3.

11.2.2 Updating the Means

Theorem 11.1 (Update of the GMM Means)**.** *The update of the mean parameters $\boldsymbol{\mu}_k$, $k = 1, \ldots, K$, of the GMM is given by*

$$\boldsymbol{\mu}_k^{new} = \frac{\sum_{n=1}^N r_{nk} \boldsymbol{x}_n}{\sum_{n=1}^N r_{nk}}, \tag{11.20}$$

where the responsibilities r_{nk} are defined in (11.17).

Remark. The update of the means $\boldsymbol{\mu}_k$ of the individual mixture components in (11.20) depends on all means, covariance matrices $\boldsymbol{\Sigma}_k$, and mixture weights π_k via r_{nk} given in (11.17). Therefore, we cannot obtain a closed-form solution for all $\boldsymbol{\mu}_k$ at once. ♢

Proof From (11.15), we see that the gradient of the log-likelihood with respect to the mean parameters $\boldsymbol{\mu}_k$, $k = 1, \ldots, K$, requires us to compute the partial derivative

$$\frac{\partial p(\boldsymbol{x}_n \mid \boldsymbol{\theta})}{\partial \boldsymbol{\mu}_k} = \sum_{j=1}^{K} \pi_j \frac{\partial \mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}{\partial \boldsymbol{\mu}_k} = \pi_k \frac{\partial \mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\partial \boldsymbol{\mu}_k} \tag{11.21a}$$

$$= \pi_k (\boldsymbol{x}_n - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1} \mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \,, \tag{11.21b}$$

where we exploited that only the kth mixture component depends on $\boldsymbol{\mu}_k$.

We use our result from (11.21b) in (11.15) and put everything together so that the desired partial derivative of $\mathcal{L}$ with respect to $\boldsymbol{\mu}_k$ is given as

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\mu}_k} = \sum_{n=1}^{N} \frac{\partial \log p(\boldsymbol{x}_n \mid \boldsymbol{\theta})}{\partial \boldsymbol{\mu}_k} = \sum_{n=1}^{N} \frac{1}{p(\boldsymbol{x}_n \mid \boldsymbol{\theta})} \frac{\partial p(\boldsymbol{x}_n \mid \boldsymbol{\theta})}{\partial \boldsymbol{\mu}_k} \tag{11.22a}$$

$$= \sum_{n=1}^{N} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1} \underbrace{\boxed{\frac{\pi_k \mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}}}_{=r_{nk}} \tag{11.22b}$$

$$= \sum_{n=1}^{N} r_{nk} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1} \,. \tag{11.22c}$$

Here we used the identity from (11.16) and the result of the partial derivative in (11.21b) to get to (11.22b). The values r_{nk} are the responsibilities we defined in (11.17).

We now solve (11.22c) for $\boldsymbol{\mu}_k^{\text{new}}$ so that $\frac{\partial \mathcal{L}(\boldsymbol{\mu}_k^{\text{new}})}{\partial \boldsymbol{\mu}_k} = \mathbf{0}^\top$ and obtain

$$\sum_{n=1}^{N} r_{nk} \boldsymbol{x}_n = \sum_{n=1}^{N} r_{nk} \boldsymbol{\mu}_k^{\text{new}} \iff \boldsymbol{\mu}_k^{\text{new}} = \frac{\sum_{n=1}^{N} r_{nk} \boldsymbol{x}_n}{\boxed{\sum_{n=1}^{N} r_{nk}}} = \frac{1}{\boxed{N_k}} \sum_{n=1}^{N} r_{nk} \boldsymbol{x}_n \,, \tag{11.23}$$

where we defined

$$N_k := \sum_{n=1}^{N} r_{nk} \tag{11.24}$$

as the total responsibility of the kth mixture component for the entire dataset. This concludes the proof of Theorem 11.1. □

Intuitively, (11.20) can be interpreted as an importance-weighted Monte Carlo estimate of the mean, where the importance weights of data point $\boldsymbol{x}_n$ are the responsibilities r_{nk} of the kth cluster for $\boldsymbol{x}_n$, $k = 1, \ldots, K$.

Therefore, the mean $\boldsymbol{\mu}_k$ is pulled toward a data point $\boldsymbol{x}_n$ with strength given by r_{nk}. The means are pulled stronger toward data points for which the corresponding mixture component has a high responsibility, i.e., a high likelihood. Figure 11.4 illustrates this. We can also interpret the mean update in (11.20) as the expected value of all data points under the distribution given by

$$\boldsymbol{r}_k := [r_{1k}, \ldots, r_{Nk}]^\top / N_k \,, \tag{11.25}$$

which is a normalized probability vector, i.e.,

$$\boldsymbol{\mu}_k \leftarrow \mathbb{E}_{\boldsymbol{r}_k}[\mathcal{X}] \,. \tag{11.26}$$

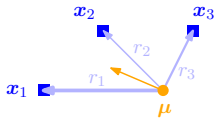
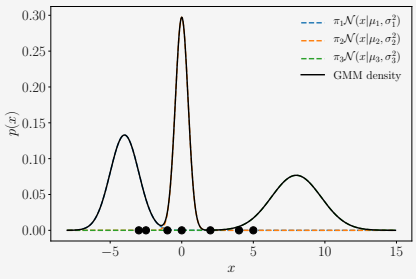
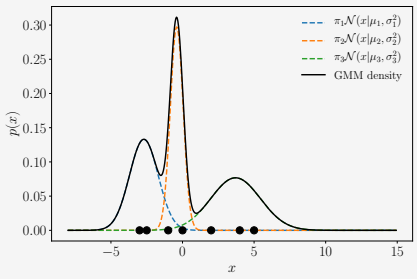


Figure 11.4 Update of the mean parameter of mixture component in a GMM. The mean $\boldsymbol{\mu}$ is being pulled toward individual data points with the weights given by the corresponding responsibilities.

Example 11.3 (Mean Updates)



(a) GMM density and individual components prior to updating the mean values.



(b) GMM density and individual components after updating the mean values.

Figure 11.5 Effect of updating the mean values in a GMM. (a) GMM before updating the mean values; (b) GMM after updating the mean values μ_k while retaining the variances and mixture weights.

In our example from Figure 11.3, the mean values are updated as follows:

$$\mu_1 : -4 \to -2.7 \tag{11.27}$$
$$\mu_2 : 0 \to -0.4 \tag{11.28}$$
$$\mu_3 : 8 \to 3.7 \tag{11.29}$$

Here we see that the means of the first and third mixture component move toward the regime of the data, whereas the mean of the second component does not change so dramatically. Figure 11.5 illustrates this change, where Figure 11.5(a) shows the GMM density prior to updating the means and Figure 11.5(b) shows the GMM density after updating the mean values μ_k.

The update of the mean parameters in (11.20) look fairly straightforward. However, note that the responsibilities r_{nk} are a function of $\pi_j, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j$ for all $j = 1, \ldots, K$, such that the updates in (11.20) depend on all parameters of the GMM, and a closed-form solution, which we obtained for linear regression in Section 9.2 or PCA in Chapter 10, cannot be obtained.

11.2.3 Updating the Covariances

Theorem 11.2 (Updates of the GMM Covariances)**.** *The update of the covariance parameters $\boldsymbol{\Sigma}_k$, $k = 1, \ldots, K$ of the GMM is given by*

$$\boldsymbol{\Sigma}_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} r_{nk} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^\top , \tag{11.30}$$

where r_{nk} and N_k are defined in (11.17) *and* (11.24)*, respectively.*

Proof To prove Theorem 11.2, our approach is to compute the partial derivatives of the log-likelihood $\mathcal{L}$ with respect to the covariances $\boldsymbol{\Sigma}_k$, set them to $\mathbf{0}$, and solve for $\boldsymbol{\Sigma}_k$. We start with our general approach

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\Sigma}_k} = \sum_{n=1}^{N} \frac{\partial \log p(\boldsymbol{x}_n \,|\, \boldsymbol{\theta})}{\partial \boldsymbol{\Sigma}_k} = \sum_{n=1}^{N} \frac{1}{p(\boldsymbol{x}_n \,|\, \boldsymbol{\theta})} \frac{\partial p(\boldsymbol{x}_n \,|\, \boldsymbol{\theta})}{\partial \boldsymbol{\Sigma}_k} . \tag{11.31}$$

We already know $1/p(\boldsymbol{x}_n \,|\, \boldsymbol{\theta})$ from (11.16). To obtain the remaining partial derivative $\partial p(\boldsymbol{x}_n \,|\, \boldsymbol{\theta})/\partial \boldsymbol{\Sigma}_k$, we write down the definition of the Gaussian distribution $p(\boldsymbol{x}_n \,|\, \boldsymbol{\theta})$ (see (11.9)) and drop all terms but the kth. We then obtain

$$\frac{\partial p(\boldsymbol{x}_n \,|\, \boldsymbol{\theta})}{\partial \boldsymbol{\Sigma}_k} \tag{11.32a}$$

$$= \frac{\partial}{\partial \boldsymbol{\Sigma}_k} \left(\pi_k (2\pi)^{-\frac{D}{2}} \det(\boldsymbol{\Sigma}_k)^{-\frac{1}{2}} \exp\left(-\tfrac{1}{2} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1} (\boldsymbol{x}_n - \boldsymbol{\mu}_k) \right) \right) \tag{11.32b}$$

$$= \pi_k (2\pi)^{-\frac{D}{2}} \left[\frac{\partial}{\partial \boldsymbol{\Sigma}_k} \det(\boldsymbol{\Sigma}_k)^{-\frac{1}{2}} \exp\left(-\tfrac{1}{2} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1} (\boldsymbol{x}_n - \boldsymbol{\mu}_k) \right) \right.$$
$$\left. + \det(\boldsymbol{\Sigma}_k)^{-\frac{1}{2}} \frac{\partial}{\partial \boldsymbol{\Sigma}_k} \exp\left(-\tfrac{1}{2} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1} (\boldsymbol{x}_n - \boldsymbol{\mu}_k) \right) \right] . \tag{11.32c}$$

We now use the identities

$$\frac{\partial}{\partial \boldsymbol{\Sigma}_k} \det(\boldsymbol{\Sigma}_k)^{-\frac{1}{2}} \overset{(5.101)}{=} -\frac{1}{2} \det(\boldsymbol{\Sigma}_k)^{-\frac{1}{2}} \boldsymbol{\Sigma}_k^{-1} , \tag{11.33}$$

$$\frac{\partial}{\partial \boldsymbol{\Sigma}_k} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1} (\boldsymbol{x}_n - \boldsymbol{\mu}_k) \overset{(5.103)}{=} -\boldsymbol{\Sigma}_k^{-1} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1} \tag{11.34}$$

and obtain (after some rearranging) the desired partial derivative required in (11.31) as

$$\frac{\partial p(\boldsymbol{x}_n \,|\, \boldsymbol{\theta})}{\partial \boldsymbol{\Sigma}_k} = \pi_k \mathcal{N}\left(\boldsymbol{x}_n \,|\, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right)$$
$$\cdot \left[-\tfrac{1}{2} (\boldsymbol{\Sigma}_k^{-1} - \boldsymbol{\Sigma}_k^{-1} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1}) \right] . \tag{11.35}$$

Putting everything together, the partial derivative of the log-likelihood

with respect to $\boldsymbol{\Sigma}_k$ is given by

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{\Sigma}_k} = \sum_{n=1}^{N} \frac{\partial \log p(\boldsymbol{x}_n \mid \boldsymbol{\theta})}{\partial \boldsymbol{\Sigma}_k} = \sum_{n=1}^{N} \frac{1}{p(\boldsymbol{x}_n \mid \boldsymbol{\theta})} \frac{\partial p(\boldsymbol{x}_n \mid \boldsymbol{\theta})}{\partial \boldsymbol{\Sigma}_k} \tag{11.36a}$$

$$= \sum_{n=1}^{N} \underbrace{\frac{\pi_k \mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\boldsymbol{x}_n \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)}}_{=r_{nk}}$$

$$\cdot \left[-\tfrac{1}{2}(\boldsymbol{\Sigma}_k^{-1} - \boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1})\right] \tag{11.36b}$$

$$= -\frac{1}{2} \sum_{n=1}^{N} r_{nk} (\boldsymbol{\Sigma}_k^{-1} - \boldsymbol{\Sigma}_k^{-1}(\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^\top \boldsymbol{\Sigma}_k^{-1}) \tag{11.36c}$$

$$= -\frac{1}{2} \boldsymbol{\Sigma}_k^{-1} \underbrace{\sum_{n=1}^{N} r_{nk}}_{=N_k} + \frac{1}{2} \boldsymbol{\Sigma}_k^{-1} \left(\sum_{n=1}^{N} r_{nk} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^\top \right) \boldsymbol{\Sigma}_k^{-1} . \tag{11.36d}$$

We see that the responsibilities r_{nk} also appear in this partial derivative. Setting this partial derivative to $\mathbf{0}$, we obtain the necessary optimality condition

$$N_k \boldsymbol{\Sigma}_k^{-1} = \boldsymbol{\Sigma}_k^{-1} \left(\sum_{n=1}^{N} r_{nk} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^\top \right) \boldsymbol{\Sigma}_k^{-1} \tag{11.37a}$$

$$\iff N_k \boldsymbol{I} = \left(\sum_{n=1}^{N} r_{nk} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^\top \right) \boldsymbol{\Sigma}_k^{-1} . \tag{11.37b}$$

By solving for $\boldsymbol{\Sigma}_k$, we obtain

$$\boldsymbol{\Sigma}_k^{\text{new}} = \frac{1}{N_k} \sum_{n=1}^{N} r_{nk} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^\top , \tag{11.38}$$

where $\boldsymbol{r}_k$ is the probability vector defined in (11.25). This gives us a simple update rule for $\boldsymbol{\Sigma}_k$ for $k = 1, \ldots, K$ and proves Theorem 11.2. □

Similar to the update of $\boldsymbol{\mu}_k$ in (11.20), we can interpret the update of the covariance in (11.30) as an importance-weighted expected value of the square of the centered data $\tilde{\mathcal{X}}_k := \{\boldsymbol{x}_1 - \boldsymbol{\mu}_k, \ldots, \boldsymbol{x}_N - \boldsymbol{\mu}_k\}$.

Example 11.4 (Variance Updates)
In our example from Figure 11.3, the variances are updated as follows:

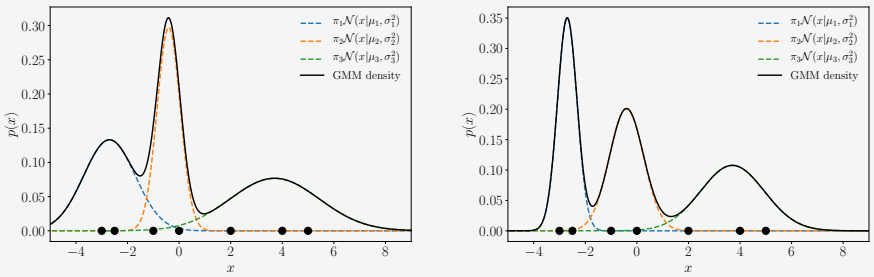
$$\sigma_1^2 : 1 \to 0.14 \tag{11.39}$$

$$\sigma_2^2 : 0.2 \to 0.44 \tag{11.40}$$

$$\sigma_3^2 : 3 \to 1.53 \tag{11.41}$$

Here we see that the variances of the first and third component shrink significantly, whereas the variance of the second component increases slightly.

Figure 11.6 illustrates this setting. Figure 11.6(a) is identical (but zoomed in) to Figure 11.5(b) and shows the GMM density and its individual components prior to updating the variances. Figure 11.6(b) shows the GMM density after updating the variances.



(a) GMM density and individual components prior to updating the variances.

(b) GMM density and individual components after updating the variances.

Figure 11.6 Effect of updating the variances in a GMM. (a) GMM before updating the variances; (b) GMM after updating the variances while retaining the means and mixture weights.

Similar to the update of the mean parameters, we can interpret (11.30) as a Monte Carlo estimate of the weighted covariance of data points $\boldsymbol{x}_n$ associated with the kth mixture component, where the weights are the responsibilities r_{nk}. As with the updates of the mean parameters, this update depends on all $\pi_j, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j,\ j = 1, \ldots, K$, through the responsibilities r_{nk}, which prohibits a closed-form solution.

11.2.4 Updating the Mixture Weights

Theorem 11.3 (Update of the GMM Mixture Weights)**.** *The mixture weights of the GMM are updated as*

$$\pi_k^{new} = \frac{N_k}{N}, \quad k = 1, \ldots, K, \tag{11.42}$$

where N is the number of data points and N_k is defined in (11.24)*.*

Proof To find the partial derivative of the log-likelihood with respect to the weight parameters π_k, $k = 1, \ldots, K$, we account for the constraint $\sum_k \pi_k = 1$ by using Lagrange multipliers (see Section 7.2). The Lagrangian is

$$\mathfrak{L} = \mathcal{L} + \lambda \left(\sum_{k=1}^{K} \pi_k - 1 \right) \tag{11.43a}$$

$$= \sum_{n=1}^{N} \log \sum_{k=1}^{K} \pi_k \mathcal{N}\big(\boldsymbol{x}_n \,|\, \boldsymbol{\mu}_k,\, \boldsymbol{\Sigma}_k\big) + \lambda \left(\sum_{k=1}^{K} \pi_k - 1 \right) , \tag{11.43b}$$

where $\mathcal{L}$ is the log-likelihood from (11.10) and the second term encodes for the equality constraint that all the mixture weights need to sum up to 1. We obtain the partial derivative with respect to π_k as

$$\frac{\partial \mathfrak{L}}{\partial \pi_k} = \sum_{n=1}^{N} \frac{\mathcal{N}\big(\boldsymbol{x}_n \,|\, \boldsymbol{\mu}_k,\, \boldsymbol{\Sigma}_k\big)}{\sum_{j=1}^{K} \pi_j \mathcal{N}\big(\boldsymbol{x}_n \,|\, \boldsymbol{\mu}_j,\, \boldsymbol{\Sigma}_j\big)} + \lambda \tag{11.44a}$$

$$= \frac{1}{\pi_k} \underbrace{\sum_{n=1}^{N} \frac{\pi_k \mathcal{N}\big(\boldsymbol{x}_n \,|\, \boldsymbol{\mu}_k,\, \boldsymbol{\Sigma}_k\big)}{\sum_{j=1}^{K} \pi_j \mathcal{N}\big(\boldsymbol{x}_n \,|\, \boldsymbol{\mu}_j,\, \boldsymbol{\Sigma}_j\big)}}_{=N_k} + \lambda = \frac{N_k}{\pi_k} + \lambda \,, \tag{11.44b}$$

and the partial derivative with respect to the Lagrange multiplier λ as

$$\frac{\partial \mathfrak{L}}{\partial \lambda} = \sum_{k=1}^{K} \pi_k - 1 \,. \tag{11.45}$$

Setting both partial derivatives to $\mathbf{0}$ (necessary condition for optimum) yields the system of equations

$$\pi_k = -\frac{N_k}{\lambda} \,, \tag{11.46}$$

$$1 = \sum_{k=1}^{K} \pi_k \,. \tag{11.47}$$

Using (11.46) in (11.47) and solving for π_k, we obtain

$$\sum_{k=1}^{K} \pi_k = 1 \iff -\sum_{k=1}^{K} \frac{N_k}{\lambda} = 1 \iff -\frac{N}{\lambda} = 1 \iff \lambda = -N \,. \tag{11.48}$$

This allows us to substitute $-N$ for λ in (11.46) to obtain

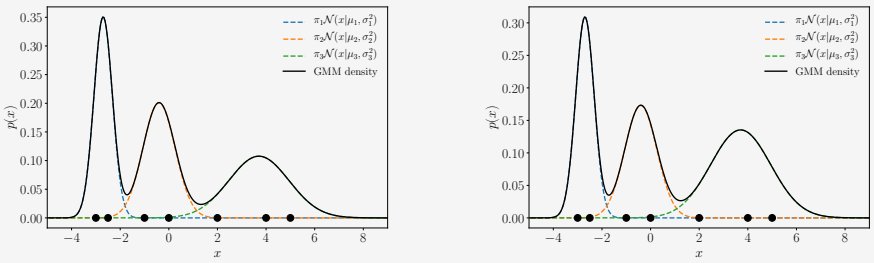
$$\pi_k^{\text{new}} = \frac{N_k}{N} \,, \tag{11.49}$$

which gives us the update for the weight parameters π_k and proves Theorem 11.3. □

We can identify the mixture weight in (11.42) as the ratio of the total responsibility of the kth cluster and the number of data points. Since $N = \sum_k N_k$, the number of data points can also be interpreted as the total responsibility of all mixture components together, such that π_k is the relative importance of the kth mixture component for the dataset.

Remark. Since $N_k = \sum_{i=1}^{N} r_{nk}$, the update equation (11.42) for the mixture weights π_k also depends on all $\pi_j, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j, j = 1, \ldots, K$ via the responsibilities r_{nk}. ♢

Example 11.5 (Weight Parameter Updates)



(a) GMM density and individual components prior to updating the mixture weights.

(b) GMM density and individual components after updating the mixture weights.

Figure 11.7 Effect of updating the mixture weights in a GMM. (a) GMM before updating the mixture weights; (b) GMM after updating the mixture weights while retaining the means and variances. Note the different scales of the vertical axes.

In our running example from Figure 11.3, the mixture weights are updated as follows:

$$\pi_1 : \frac{1}{3} \to 0.29 \tag{11.50}$$
$$\pi_2 : \frac{1}{3} \to 0.29 \tag{11.51}$$
$$\pi_3 : \frac{1}{3} \to 0.42 \tag{11.52}$$

Here we see that the third component gets more weight/importance, while the other components become slightly less important. Figure 11.7 illustrates the effect of updating the mixture weights. Figure 11.7(a) is identical to Figure 11.6(b) and shows the GMM density and its individual components prior to updating the mixture weights. Figure 11.7(b) shows the GMM density after updating the mixture weights.

Overall, having updated the means, the variances, and the weights once, we obtain the GMM shown in Figure 11.7(b). Compared with the initialization shown in Figure 11.3, we can see that the parameter updates caused the GMM density to shift some of its mass toward the data points.

After updating the means, variances, and weights once, the GMM fit in Figure 11.7(b) is already remarkably better than its initialization from Figure 11.3. This is also evidenced by the log-likelihood values, which increased from 28.3 (initialization) to 14.4 after one complete update cycle.

11.3 EM Algorithm

Unfortunately, the updates in (11.20), (11.30), and (11.42) do not constitute a closed-form solution for the updates of the parameters $\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \pi_k$ of the mixture model because the responsibilities r_{nk} depend on those parameters in a complex way. However, the results suggest a simple *iterative scheme* for finding a solution to the parameters estimation problem via maximum likelihood. The expectation maximization algorithm (*EM algo-*

EM algorithm

rithm) was proposed by Dempster et al. (1977) and is a general iterative scheme for learning parameters (maximum likelihood or MAP) in mixture models and, more generally, latent-variable models.

In our example of the Gaussian mixture model, we choose initial values for $\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \pi_k$ and alternate until convergence between

- *E-step:* Evaluate the responsibilities r_{nk} (posterior probability of data point n belonging to mixture component k).
- *M-step:* Use the updated responsibilities to reestimate the parameters $\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \pi_k$.

Every step in the EM algorithm increases the log-likelihood function (Neal and Hinton, 1999). For convergence, we can check the log-likelihood or the parameters directly. A concrete instantiation of the EM algorithm for estimating the parameters of a GMM is as follows:

1. Initialize $\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \pi_k$.
2. *E-step:* Evaluate responsibilities r_{nk} for every data point $\boldsymbol{x}_n$ using current parameters $\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$:

$$r_{nk} = \frac{\pi_k \mathcal{N}\left(\boldsymbol{x}_n \,|\, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right)}{\sum_j \pi_j \mathcal{N}\left(\boldsymbol{x}_n \,|\, \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\right)} . \tag{11.53}$$

3. *M-step:* Reestimate parameters $\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$ using the current responsibilities r_{nk} (from E-step):

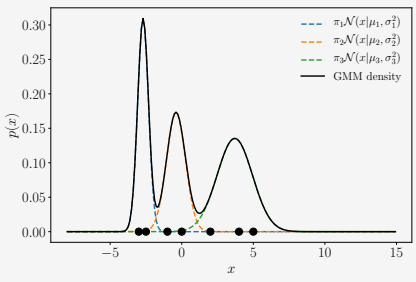
$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{n=1}^{N} r_{nk} \boldsymbol{x}_n , \tag{11.54}$$

$$\boldsymbol{\Sigma}_k = \frac{1}{N_k} \sum_{n=1}^{N} r_{nk} (\boldsymbol{x}_n - \boldsymbol{\mu}_k)(\boldsymbol{x}_n - \boldsymbol{\mu}_k)^\top , \tag{11.55}$$

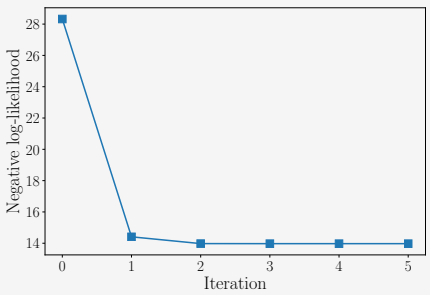
$$\pi_k = \frac{N_k}{N} . \tag{11.56}$$

Having updated the means $\boldsymbol{\mu}_k$ in (11.54), they are subsequently used in (11.55) to update the corresponding covariances.

Example 11.6 (GMM Fit)

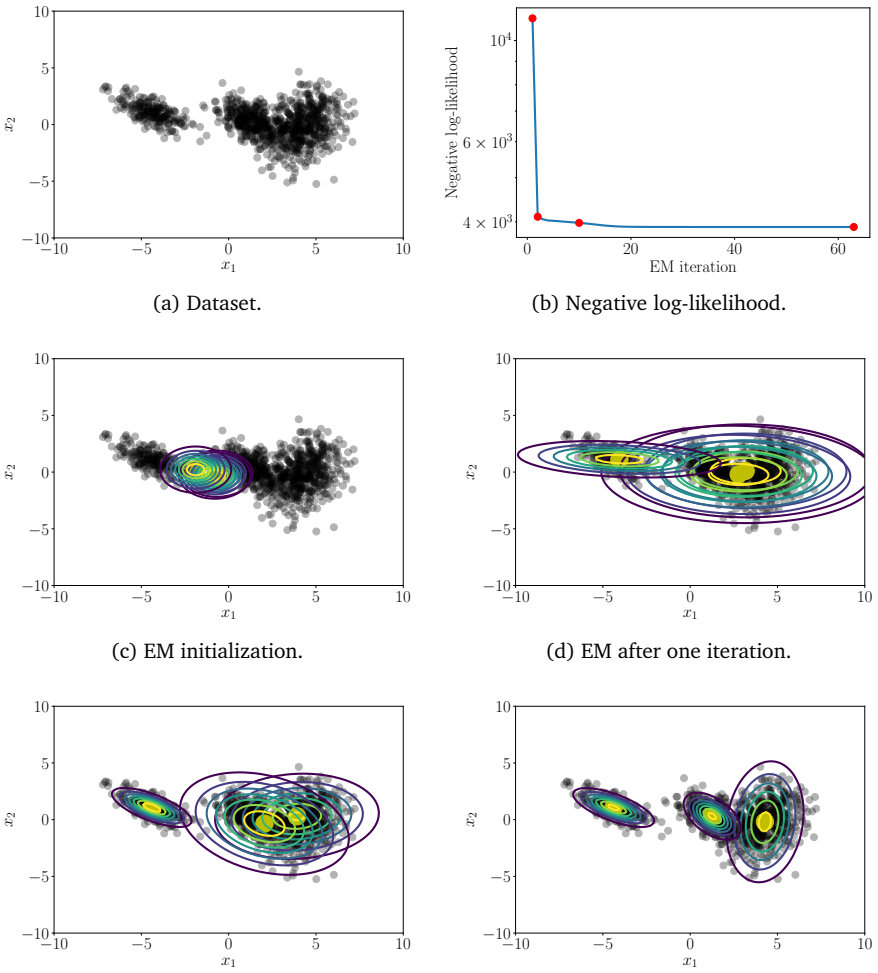


(a) Final GMM fit. After five iterations, the EM algorithm converges and returns this GMM.



(b) Negative log-likelihood as a function of the EM iterations.

Figure 11.8 EM algorithm applied to the GMM from Figure 11.2. (a) Final GMM fit; (b) negative log-likelihood as a function of the EM iteration.



(a) Dataset.

(b) Negative log-likelihood.

(c) EM initialization.

(d) EM after one iteration.

(e) EM after 10 iterations.

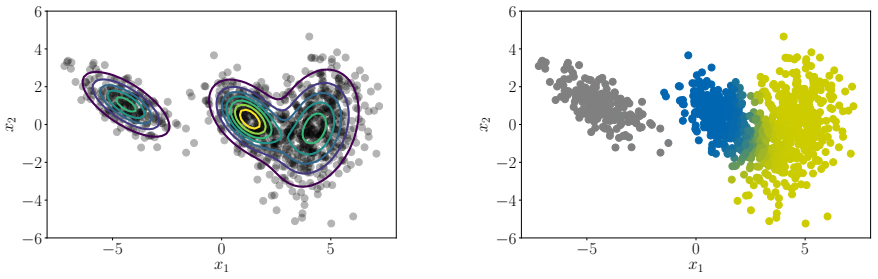
(f) EM after 62 iterations.

Figure 11.9 Illustration of the EM algorithm for fitting a Gaussian mixture model with three components to a two-dimensional dataset. (a) Dataset; (b) negative log-likelihood (lower is better) as a function of the EM iterations. The red dots indicate the iterations for which the mixture components of the corresponding GMM fits are shown in (c) through (f). The yellow discs indicate the means of the Gaussian mixture components. Figure 11.10(a) shows the final GMM fit.

When we run EM on our example from Figure 11.3, we obtain the final result shown in Figure 11.8(a) after five iterations, and Figure 11.8(b) shows how the negative log-likelihood evolves as a function of the EM iterations. The final GMM is given as

$$\begin{aligned} p(x) = &\, 0.29\mathcal{N}\big(x \,|\, -2.75,\, 0.06\big) + 0.28\mathcal{N}\big(x \,|\, -0.50,\, 0.25\big) \\ &+ 0.43\mathcal{N}\big(x \,|\, 3.64,\, 1.63\big)\,. \end{aligned} \tag{11.57}$$

We applied the EM algorithm to the two-dimensional dataset shown in Figure 11.1 with $K = 3$ mixture components. Figure 11.9 illustrates some steps of the EM algorithm and shows the negative log-likelihood as a function of the EM iteration (Figure 11.9(b)). Figure 11.10(a) shows



(a) GMM fit after 62 iterations.

(b) Dataset colored according to the responsibilities of the mixture components.

Figure 11.10 GMM fit and responsibilities when EM converges. (a) GMM fit when EM converges; (b) each data point is colored according to the responsibilities of the mixture components.

the corresponding final GMM fit. Figure 11.10(b) visualizes the final responsibilities of the mixture components for the data points. The dataset is colored according to the responsibilities of the mixture components when EM converges. While a single mixture component is clearly responsible for the data on the left, the overlap of the two data clusters on the right could have been generated by two mixture components. It becomes clear that there are data points that cannot be uniquely assigned to a single component (either blue or yellow), such that the responsibilities of these two clusters for those points are around 0.5.

11.4 Latent-Variable Perspective

We can look at the GMM from the perspective of a discrete latent-variable model, i.e., where the latent variable $\boldsymbol{z}$ can attain only a finite set of values. This is in contrast to PCA, where the latent variables were continuous-valued numbers in $\mathbb{R}^M$.

The advantages of the probabilistic perspective are that (i) it will justify some ad hoc decisions we made in the previous sections, (ii) it allows for a concrete interpretation of the responsibilities as posterior probabilities, and (iii) the iterative algorithm for updating the model parameters can be derived in a principled manner as the EM algorithm for maximum likelihood parameter estimation in latent-variable models.

11.4.1 Generative Process and Probabilistic Model

To derive the probabilistic model for GMMs, it is useful to think about the generative process, i.e., the process that allows us to generate data, using a probabilistic model.

We assume a mixture model with K components and that a data point $\boldsymbol{x}$ can be generated by exactly one mixture component. We introduce a binary indicator variable $z_k \in \{0, 1\}$ with two states (see Section 6.2) that indicates whether the kth mixture component generated that data point

so that

$$p(\boldsymbol{x} \mid z_k = 1) = \mathcal{N}\big(\boldsymbol{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\big)\,. \tag{11.58}$$

We define $\boldsymbol{z} := [z_1, \ldots, z_K]^\top \in \mathbb{R}^K$ as a probability vector consisting of $K-1$ many 0s and exactly one 1. For example, for $K = 3$, a valid $\boldsymbol{z}$ would be $\boldsymbol{z} = [z_1, z_2, z_3]^\top = [0, 1, 0]^\top$, which would select the second mixture component since $z_2 = 1$.

Remark. Sometimes this kind of probability distribution is called "multinoulli", a generalization of the Bernoulli distribution to more than two values (Murphy, 2012). ♢

one-hot encoding
1-of-K representation

The properties of $\boldsymbol{z}$ imply that $\sum_{k=1}^K z_k = 1$. Therefore, $\boldsymbol{z}$ is a *one-hot encoding* (also: 1-*of-*K *representation*).

Thus far, we assumed that the indicator variables z_k are known. However, in practice, this is not the case, and we place a prior distribution

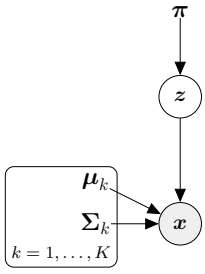
$$p(\boldsymbol{z}) = \boldsymbol{\pi} = [\pi_1, \ldots, \pi_K]^\top\,, \quad \sum_{k=1}^K \pi_k = 1\,, \tag{11.59}$$

on the latent variable $\boldsymbol{z}$. Then the kth entry

$$\pi_k = p(z_k = 1) \tag{11.60}$$

of this probability vector describes the probability that the kth mixture component generated data point $\boldsymbol{x}$.

Figure 11.11 Graphical model for a GMM with a single data point.



Remark (Sampling from a GMM). The construction of this latent-variable model (see the corresponding graphical model in Figure 11.11) lends itself to a very simple sampling procedure (generative process) to generate data:

1. Sample $z^{(i)} \sim p(\boldsymbol{z})$.
2. Sample $\boldsymbol{x}^{(i)} \sim p(\boldsymbol{x} \mid z^{(i)} = 1)$.

In the first step, we select a mixture component i (via the one-hot encoding $\boldsymbol{z}$) at random according to $p(\boldsymbol{z}) = \boldsymbol{\pi}$; in the second step we draw a sample from the corresponding mixture component. When we discard the samples of the latent variable so that we are left with the $\boldsymbol{x}^{(i)}$, we have valid samples from the GMM. This kind of sampling, where samples of random variables depend on samples from the variable's parents in the graphical model, is called *ancestral sampling*. ♢

ancestral sampling

Generally, a probabilistic model is defined by the joint distribution of the data and the latent variables (see Section 8.4). With the prior $p(\boldsymbol{z})$ defined in (11.59) and (11.60) and the conditional $p(\boldsymbol{x} \mid \boldsymbol{z})$ from (11.58), we obtain all K components of this joint distribution via

$$p(\boldsymbol{x}, z_k = 1) = p(\boldsymbol{x} \mid z_k = 1)p(z_k = 1) = \pi_k \mathcal{N}\big(\boldsymbol{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\big) \tag{11.61}$$

for $k = 1, \ldots, K$, so that

$$p(\boldsymbol{x}, \boldsymbol{z}) = \begin{bmatrix} p(\boldsymbol{x}, z_1 = 1) \\ \vdots \\ p(\boldsymbol{x}, z_K = 1) \end{bmatrix} = \begin{bmatrix} \pi_1 \mathcal{N}\big(\boldsymbol{x} \,|\, \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1\big) \\ \vdots \\ \pi_K \mathcal{N}\big(\boldsymbol{x} \,|\, \boldsymbol{\mu}_K, \boldsymbol{\Sigma}_K\big) \end{bmatrix}, \tag{11.62}$$

which fully specifies the probabilistic model.

11.4.2 Likelihood

To obtain the likelihood $p(\boldsymbol{x} \,|\, \boldsymbol{\theta})$ in a latent-variable model, we need to marginalize out the latent variables (see Section 8.4.3). In our case, this can be done by summing out all latent variables from the joint $p(\boldsymbol{x}, \boldsymbol{z})$ in (11.62) so that

$$p(\boldsymbol{x} \,|\, \boldsymbol{\theta}) = \sum_{\boldsymbol{z}} p(\boldsymbol{x} \,|\, \boldsymbol{\theta}, \boldsymbol{z}) p(\boldsymbol{z} \,|\, \boldsymbol{\theta}), \quad \boldsymbol{\theta} := \{\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \pi_k : \; k = 1, \ldots, K\}. \tag{11.63}$$

We now explicitly condition on the parameters $\boldsymbol{\theta}$ of the probabilistic model, which we previously omitted. In (11.63), we sum over all K possible one-hot encodings of $\boldsymbol{z}$, which is denoted by $\sum_{\boldsymbol{z}}$. Since there is only a single nonzero single entry in each $\boldsymbol{z}$ there are only K possible configurations/ settings of $\boldsymbol{z}$. For example, if $K = 3$, then $\boldsymbol{z}$ can have the configurations

$$\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}. \tag{11.64}$$

Summing over all possible configurations of $\boldsymbol{z}$ in (11.63) is equivalent to looking at the nonzero entry of the $\boldsymbol{z}$-vector and writing

$$p(\boldsymbol{x} \,|\, \boldsymbol{\theta}) = \sum_{\boldsymbol{z}} p(\boldsymbol{x} \,|\, \boldsymbol{\theta}, \boldsymbol{z}) p(\boldsymbol{z} \,|\, \boldsymbol{\theta}) \tag{11.65a}$$

$$= \sum_{k=1}^{K} p(\boldsymbol{x} \,|\, \boldsymbol{\theta}, z_k = 1) p(z_k = 1 \,|\, \boldsymbol{\theta}) \tag{11.65b}$$

so that the desired marginal distribution is given as

$$p(\boldsymbol{x} \,|\, \boldsymbol{\theta}) \overset{(11.65b)}{=} \sum_{k=1}^{K} p(\boldsymbol{x} \,|\, \boldsymbol{\theta}, z_k = 1) p(z_k = 1 | \boldsymbol{\theta}) \tag{11.66a}$$

$$= \sum_{k=1}^{K} \pi_k \mathcal{N}\big(\boldsymbol{x} \,|\, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\big), \tag{11.66b}$$

which we identify as the GMM model from (11.3). Given a dataset $\mathcal{X}$, we immediately obtain the likelihood

$$p(\mathcal{X} \,|\, \boldsymbol{\theta}) = \prod_{n=1}^{N} p(\boldsymbol{x}_n \,|\, \boldsymbol{\theta}) \overset{(11.66b)}{=} \prod_{n=1}^{N} \sum_{k=1}^{K} \pi_k \mathcal{N}\big(\boldsymbol{x}_n \,|\, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\big), \tag{11.67}$$

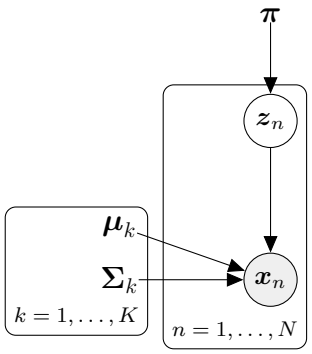


Figure 11.12 Graphical model for a GMM with N data points.

which is exactly the GMM likelihood from (11.9). Therefore, the latent-variable model with latent indicators z_k is an equivalent way of thinking about a Gaussian mixture model.

11.4.3 Posterior Distribution

Let us have a brief look at the posterior distribution on the latent variable $\boldsymbol{z}$. According to Bayes' theorem, the posterior of the kth component having generated data point $\boldsymbol{x}$

$$p(z_k = 1 \mid \boldsymbol{x}) = \frac{p(z_k = 1)p(\boldsymbol{x} \mid z_k = 1)}{p(\boldsymbol{x})}, \tag{11.68}$$

where the marginal $p(\boldsymbol{x})$ is given in (11.66b). This yields the posterior distribution for the kth indicator variable z_k

$$p(z_k = 1 \mid \boldsymbol{x}) = \frac{p(z_k = 1)p(\boldsymbol{x} \mid z_k = 1)}{\sum_{j=1}^{K} p(z_j = 1)p(\boldsymbol{x} \mid z_j = 1)} = \frac{\pi_k \mathcal{N}\big(\boldsymbol{x} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\big)}{\sum_{j=1}^{K} \pi_j \mathcal{N}\big(\boldsymbol{x} \mid \boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j\big)}, \tag{11.69}$$

which we identify as the responsibility of the kth mixture component for data point $\boldsymbol{x}$. Note that we omitted the explicit conditioning on the GMM parameters $\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$ where $k = 1, \ldots, K$.

11.4.4 Extension to a Full Dataset

Thus far, we have only discussed the case where the dataset consists only of a single data point $\boldsymbol{x}$. However, the concepts of the prior and posterior can be directly extended to the case of N data points $\mathcal{X} := \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N\}$.

In the probabilistic interpretation of the GMM, every data point $\boldsymbol{x}_n$ possesses its own latent variable

$$\boldsymbol{z}_n = [z_{n1}, \ldots, z_{nK}]^\top \in \mathbb{R}^K. \tag{11.70}$$

Previously (when we only considered a single data point $\boldsymbol{x}$), we omitted the index n, but now this becomes important.

We share the same prior distribution $\boldsymbol{\pi}$ across all latent variables $\boldsymbol{z}_n$. The corresponding graphical model is shown in Figure 11.12, where we use the plate notation.

The conditional distribution $p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \mid \boldsymbol{z}_1, \ldots, \boldsymbol{z}_N)$ factorizes over the data points and is given as

$$p(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \mid \boldsymbol{z}_1, \ldots, \boldsymbol{z}_N) = \prod_{n=1}^{N} p(\boldsymbol{x}_n \mid \boldsymbol{z}_n)\,. \tag{11.71}$$

To obtain the posterior distribution $p(z_{nk} = 1 \mid \boldsymbol{x}_n)$, we follow the same reasoning as in Section 11.4.3 and apply Bayes' theorem to obtain

$$p(z_{nk} = 1 \mid \boldsymbol{x}_n) = \frac{p(\boldsymbol{x}_n \mid z_{nk} = 1)p(z_{nk} = 1)}{\sum_{j=1}^{K} p(\boldsymbol{x}_n \mid z_{nj} = 1)p(z_{nj} = 1)} \tag{11.72a}$$

$$= \frac{\pi_k \mathcal{N}\big(\boldsymbol{x}_n \mid \boldsymbol{\mu}_k,\, \boldsymbol{\Sigma}_k\big)}{\sum_{j=1}^{K} \pi_j \mathcal{N}\big(\boldsymbol{x}_n \mid \boldsymbol{\mu}_j,\, \boldsymbol{\Sigma}_j\big)} = r_{nk}\,. \tag{11.72b}$$

This means that $p(z_k = 1 \mid \boldsymbol{x}_n)$ is the (posterior) probability that the kth mixture component generated data point $\boldsymbol{x}_n$ and corresponds to the responsibility r_{nk} we introduced in (11.17). Now the responsibilities also have not only an intuitive but also a mathematically justified interpretation as posterior probabilities.

11.4.5 EM Algorithm Revisited

The EM algorithm that we introduced as an iterative scheme for maximum likelihood estimation can be derived in a principled way from the latent-variable perspective. Given a current setting $\boldsymbol{\theta}^{(t)}$ of model parameters, the E-step calculates the expected log-likelihood

$$Q(\boldsymbol{\theta} \mid \boldsymbol{\theta}^{(t)}) = \mathbb{E}_{\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{\theta}^{(t)}}[\log p(\boldsymbol{x}, \boldsymbol{z} \mid \boldsymbol{\theta})] \tag{11.73a}$$

$$= \int \log p(\boldsymbol{x}, \boldsymbol{z} \mid \boldsymbol{\theta}) p(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{\theta}^{(t)}) \mathrm{d}\boldsymbol{z}\,, \tag{11.73b}$$

where the expectation of $\log p(\boldsymbol{x}, \boldsymbol{z} \mid \boldsymbol{\theta})$ is taken with respect to the posterior $p(\boldsymbol{z} \mid \boldsymbol{x}, \boldsymbol{\theta}^{(t)})$ of the latent variables. The M-step selects an updated set of model parameters $\boldsymbol{\theta}^{(t+1)}$ by maximizing (11.73b).

Although an EM iteration does increase the log-likelihood, there are no guarantees that EM converges to the maximum likelihood solution. It is possible that the EM algorithm converges to a local maximum of the log-likelihood. Different initializations of the parameters $\boldsymbol{\theta}$ could be used in multiple EM runs to reduce the risk of ending up in a bad local optimum. We do not go into further details here, but refer to the excellent expositions by Rogers and Girolami (2016) and Bishop (2006).

11.5 Further Reading

The GMM can be considered a generative model in the sense that it is straightforward to generate new data using ancestral sampling (Bishop, 2006). For given GMM parameters $\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$, $k = 1, \ldots, K$, we sample an index k from the probability vector $[\pi_1, \ldots, \pi_K]^\top$ and then sample a data point $\boldsymbol{x} \sim \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$. If we repeat this N times, we obtain a dataset that has been generated by a GMM. Figure 11.1 was generated using this procedure.

Throughout this chapter, we assumed that the number of components K is known. In practice, this is often not the case. However, we could use nested cross-validation, as discussed in Section 8.6.1, to find good models.

Gaussian mixture models are closely related to the K-means clustering algorithm. K-means also uses the EM algorithm to assign data points to clusters. If we treat the means in the GMM as cluster centers and ignore the covariances (or set them to $\boldsymbol{I}$), we arrive at K-means. As also nicely described by MacKay (2003), K-means makes a "hard" assignment of data points to cluster centers $\boldsymbol{\mu}_k$, whereas a GMM makes a "soft" assignment via the responsibilities.

We only touched upon the latent-variable perspective of GMMs and the EM algorithm. Note that EM can be used for parameter learning in general latent-variable models, e.g., nonlinear state-space models (Ghahramani and Roweis, 1999; Roweis and Ghahramani, 1999) and for reinforcement learning as discussed by Barber (2012). Therefore, the latent-variable perspective of a GMM is useful to derive the corresponding EM algorithm in a principled way (Bishop, 2006; Barber, 2012; Murphy, 2012).

We only discussed maximum likelihood estimation (via the EM algorithm) for finding GMM parameters. The standard criticisms of maximum likelihood also apply here:

- As in linear regression, maximum likelihood can suffer from severe overfitting. In the GMM case, this happens when the mean of a mixture component is identical to a data point and the covariance tends to $\mathbf{0}$. Then, the likelihood approaches infinity. Bishop (2006) and Barber (2012) discuss this issue in detail.
- We only obtain a point estimate of the parameters $\pi_k, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k$ for $k = 1, \ldots, K$, which does not give any indication of uncertainty in the parameter values. A Bayesian approach would place a prior on the parameters, which can be used to obtain a posterior distribution on the parameters. This posterior allows us to compute the model evidence (marginal likelihood), which can be used for model comparison, which gives us a principled way to determine the number of mixture components. Unfortunately, closed-form inference is not possible in this setting because there is no conjugate prior for this model. However, approximations, such as variational inference, can be used to obtain an approximate posterior (Bishop, 2006).

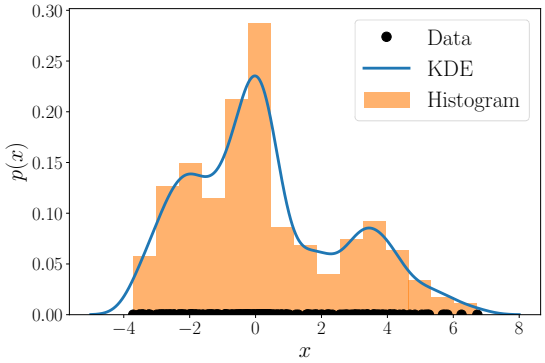


Figure 11.13 Histogram (orange bars) and kernel density estimation (blue line). The kernel density estimator produces a smooth estimate of the underlying density, whereas the histogram is an unsmoothed count measure of how many data points (black) fall into a single bin.

In this chapter, we discussed mixture models for density estimation. There is a plethora of density estimation techniques available. In practice, we often use histograms and kernel density estimation.

histogram

Histograms provide a nonparametric way to represent continuous densities and have been proposed by Pearson (1895). A histogram is constructed by "binning" the data space and count, how many data points fall into each bin. Then a bar is drawn at the center of each bin, and the height of the bar is proportional to the number of data points within that bin. The bin size is a critical hyperparameter, and a bad choice can lead to overfitting and underfitting. Cross-validation, as discussed in Section 8.2.4, can be used to determine a good bin size.

kernel density estimation

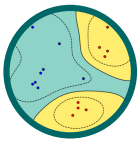
Kernel density estimation, independently proposed by Rosenblatt (1956) and Parzen (1962), is a nonparametric way for density estimation. Given N i.i.d. samples, the kernel density estimator represents the underlying distribution as

$$p(\boldsymbol{x}) = \frac{1}{Nh}\sum_{n=1}^{N} k\left(\frac{\boldsymbol{x}-\boldsymbol{x}_n}{h}\right), \tag{11.74}$$

where k is a kernel function, i.e., a nonnegative function that integrates to 1 and $h > 0$ is a smoothing/bandwidth parameter, which plays a similar role as the bin size in histograms. Note that we place a kernel on every single data point $\boldsymbol{x}_n$ in the dataset. Commonly used kernel functions are the uniform distribution and the Gaussian distribution. Kernel density estimates are closely related to histograms, but by choosing a suitable kernel, we can guarantee smoothness of the density estimate. Figure 11.13 illustrates the difference between a histogram and a kernel density estimator (with a Gaussian-shaped kernel) for a given dataset of 250 data points.

12

Classification with Support Vector Machines



In many situations, we want our machine learning algorithm to predict one of a number of (discrete) outcomes. For example, an email client sorts mail into personal mail and junk mail, which has two outcomes. Another example is a telescope that identifies whether an object in the night sky is a galaxy, star, or planet. There are usually a small number of outcomes, and more importantly there is usually no additional structure on these outcomes. In this chapter, we consider predictors that output binary values, i.e., there are only two possible outcomes. This machine learning task is called *binary classification*. This is in contrast to Chapter 9, where we considered a prediction problem with continuous-valued outputs.

An example of structure is if the outcomes were ordered, like in the case of small, medium, and large t-shirts.

binary classification

For binary classification, the set of possible values that the label/output can attain is binary, and for this chapter we denote them by $\{+1, -1\}$. In other words, we consider predictors of the form

$$f : \mathbb{R}^D \to \{+1, -1\}\,. \tag{12.1}$$

Input example $\boldsymbol{x}_n$ may also be referred to as inputs, data points, features, or instances.

class

For probabilistic models, it is mathematically convenient to use $\{0, 1\}$ as a binary representation; see the remark after Example 6.12.

Recall from Chapter 8 that we represent each example (data point) $\boldsymbol{x}_n$ as a feature vector of D real numbers. The labels are often referred to as the positive and negative *classes*, respectively. One should be careful not to infer intuitive attributes of positiveness of the $+1$ class. For example, in a cancer detection task, a patient with cancer is often labeled $+1$. In principle, any two distinct values can be used, e.g., $\{\text{True}, \text{False}\}$, $\{0, 1\}$ or $\{\text{red}, \text{blue}\}$. The problem of binary classification is well studied, and we defer a survey of other approaches to Section 12.6.

We present an approach known as the support vector machine (SVM), which solves the binary classification task. As in regression, we have a supervised learning task, where we have a set of examples $\boldsymbol{x}_n \in \mathbb{R}^D$ along with their corresponding (binary) labels $y_n \in \{+1, -1\}$. Given a training data set consisting of example–label pairs $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$, we would like to estimate parameters of the model that will give the smallest classification error. Similar to Chapter 9, we consider a linear model, and hide away the nonlinearity in a transformation ϕ of the examples (9.13). We will revisit ϕ in Section 12.4.

The SVM provides state-of-the-art results in many applications, with sound theoretical guarantees (Steinwart and Christmann, 2008). There are two main reasons why we chose to illustrate binary classification using

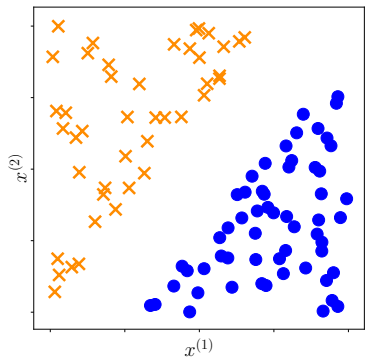


Figure 12.1 Example 2D data, illustrating the intuition of data where we can find a linear classifier that separates orange crosses from blue discs.

SVMs. First, the SVM allows for a geometric way to think about supervised machine learning. While in Chapter 9 we considered the machine learning problem in terms of probabilistic models and attacked it using maximum likelihood estimation and Bayesian inference, here we will consider an alternative approach where we reason geometrically about the machine learning task. It relies heavily on concepts, such as inner products and projections, which we discussed in Chapter 3. The second reason why we find SVMs instructive is that in contrast to Chapter 9, the optimization problem for SVM does not admit an analytic solution so that we need to resort to a variety of optimization tools introduced in Chapter 7.

The SVM view of machine learning is subtly different from the maximum likelihood view of Chapter 9. The maximum likelihood view proposes a model based on a probabilistic view of the data distribution, from which an optimization problem is derived. In contrast, the SVM view starts by designing a particular function that is to be optimized during training, based on geometric intuitions. We have seen something similar already in Chapter 10, where we derived PCA from geometric principles. In the SVM case, we start by designing a loss function that is to be minimized on training data, following the principles of empirical risk minimization (Section 8.2).

Let us derive the optimization problem corresponding to training an SVM on example–label pairs. Intuitively, we imagine binary classification data, which can be separated by a hyperplane as illustrated in Figure 12.1. Here, every example $\boldsymbol{x}_n$ (a vector of dimension 2) is a two-dimensional location ($x_n^{(1)}$ and $x_n^{(2)}$), and the corresponding binary label y_n is one of two different symbols (orange cross or blue disc). "Hyperplane" is a word that is commonly used in machine learning, and we encountered hyperplanes already in Section 2.8. A hyperplane is an affine subspace of dimension $D-1$ (if the corresponding vector space is of dimension D). The examples consist of two classes (there are two possible labels) that have features (the components of the vector representing the example) arranged in such a way as to allow us to separate/classify them by drawing a straight line.

In the following, we formalize the idea of finding a linear separator of the two classes. We introduce the idea of the margin and then extend linear separators to allow for examples to fall on the "wrong" side, incurring a classification error. We present two equivalent ways of formalizing the SVM: the geometric view (Section 12.2.4) and the loss function view (Section 12.2.5). We derive the dual version of the SVM using Lagrange multipliers (Section 7.2). The dual SVM allows us to observe a third way of formalizing the SVM: in terms of the convex hulls of the examples of each class (Section 12.3.2). We conclude by briefly describing kernels and how to numerically solve the nonlinear kernel-SVM optimization problem.

12.1 Separating Hyperplanes

Given two examples represented as vectors $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, one way to compute the similarity between them is using an inner product $\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$. Recall from Section 3.2 that inner products are closely related to the angle between two vectors. The value of the inner product between two vectors depends on the length (norm) of each vector. Furthermore, inner products allow us to rigorously define geometric concepts such as orthogonality and projections.

The main idea behind many classification algorithms is to represent data in $\mathbb{R}^D$ and then partition this space, ideally in a way that examples with the same label (and no other examples) are in the same partition. In the case of binary classification, the space would be divided into two parts corresponding to the positive and negative classes, respectively. We consider a particularly convenient partition, which is to (linearly) split the space into two halves using a hyperplane. Let example $\boldsymbol{x} \in \mathbb{R}^D$ be an element of the data space. Consider a function

$$f : \mathbb{R}^D \to \mathbb{R} \tag{12.2a}$$

$$\boldsymbol{x} \mapsto f(\boldsymbol{x}) := \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b\,, \tag{12.2b}$$

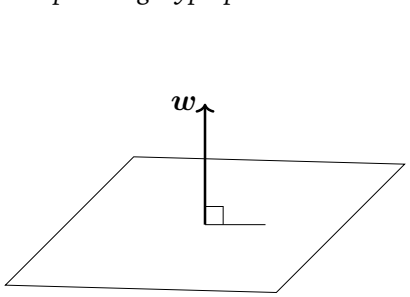
parametrized by $\boldsymbol{w} \in \mathbb{R}^D$ and $b \in \mathbb{R}$. Recall from Section 2.8 that hyperplanes are affine subspaces. Therefore, we define the hyperplane that separates the two classes in our binary classification problem as

$$\left\{ \boldsymbol{x} \in \mathbb{R}^D : f(\boldsymbol{x}) = 0 \right\} . \tag{12.3}$$

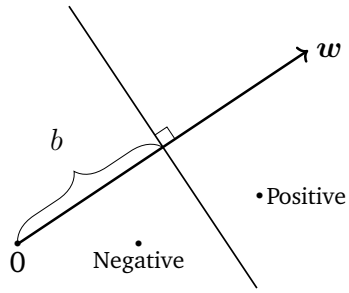
An illustration of the hyperplane is shown in Figure 12.2, where the vector $\boldsymbol{w}$ is a vector normal to the hyperplane and b the intercept. We can derive that $\boldsymbol{w}$ is a normal vector to the hyperplane in (12.3) by choosing any two examples $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ on the hyperplane and showing that the vector between them is orthogonal to $\boldsymbol{w}$. In the form of an equation,

$$f(\boldsymbol{x}_a) - f(\boldsymbol{x}_b) = \langle \boldsymbol{w}, \boldsymbol{x}_a \rangle + b - (\langle \boldsymbol{w}, \boldsymbol{x}_b \rangle + b) \tag{12.4a}$$

$$= \langle \boldsymbol{w}, \boldsymbol{x}_a - \boldsymbol{x}_b \rangle \ , \tag{12.4b}$$



(a) Separating hyperplane in 3D



(b) Projection of the setting in (a) onto a plane

Figure 12.2 Equation of a separating hyperplane (12.3). (a) The standard way of representing the equation in 3D. (b) For ease of drawing, we look at the hyperplane edge on.

where the second line is obtained by the linearity of the inner product (Section 3.2). Since we have chosen $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ to be on the hyperplane, this implies that $f(\boldsymbol{x}_a) = 0$ and $f(\boldsymbol{x}_b) = 0$ and hence $\langle \boldsymbol{w}, \boldsymbol{x}_a - \boldsymbol{x}_b \rangle = 0$. Recall that two vectors are orthogonal when their inner product is zero. Therefore, we obtain that $\boldsymbol{w}$ is orthogonal to any vector on the hyperplane.

$\boldsymbol{w}$ is orthogonal to any vector on the hyperplane.

Remark. Recall from Chapter 2 that we can think of vectors in different ways. In this chapter, we think of the parameter vector $\boldsymbol{w}$ as an arrow indicating a direction, i.e., we consider $\boldsymbol{w}$ to be a geometric vector. In contrast, we think of the example vector $\boldsymbol{x}$ as a data point (as indicated by its coordinates), i.e., we consider $\boldsymbol{x}$ to be the coordinates of a vector with respect to the standard basis. ♢

When presented with a test example, we classify the example as positive or negative depending on the side of the hyperplane on which it occurs. Note that (12.3) not only defines a hyperplane; it additionally defines a direction. In other words, it defines the positive and negative side of the hyperplane. Therefore, to classify a test example $\boldsymbol{x}_{\text{test}}$, we calculate the value of the function $f(\boldsymbol{x}_{\text{test}})$ and classify the example as $+1$ if $f(\boldsymbol{x}_{\text{test}}) \geqslant 0$ and -1 otherwise. Thinking geometrically, the positive examples lie "above" the hyperplane and the negative examples "below" the hyperplane.

When training the classifier, we want to ensure that the examples with positive labels are on the positive side of the hyperplane, i.e.,

$$\langle \boldsymbol{w}, \boldsymbol{x}_n \rangle + b \geqslant 0 \quad \text{when} \quad y_n = +1 \tag{12.5}$$

and the examples with negative labels are on the negative side, i.e.,

$$\langle \boldsymbol{w}, \boldsymbol{x}_n \rangle + b < 0 \quad \text{when} \quad y_n = -1\,. \tag{12.6}$$

Refer to Figure 12.2 for a geometric intuition of positive and negative examples. These two conditions are often presented in a single equation

$$y_n(\langle \boldsymbol{w}, \boldsymbol{x}_n \rangle + b) \geqslant 0\,. \tag{12.7}$$

Equation (12.7) is equivalent to (12.5) and (12.6) when we multiply both sides of (12.5) and (12.6) with $y_n = 1$ and $y_n = -1$, respectively.

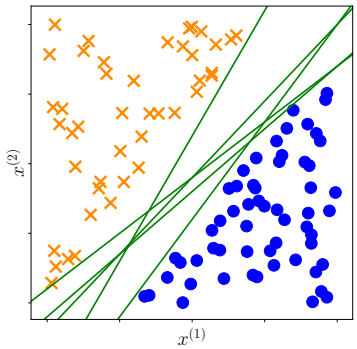


Figure 12.3 Possible separating hyperplanes. There are many linear classifiers (green lines) that separate orange crosses from blue discs.

12.2 Primal Support Vector Machine

Based on the concept of distances from points to a hyperplane, we now are in a position to discuss the support vector machine. For a dataset $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$ that is linearly separable, we have infinitely many candidate hyperplanes (refer to Figure 12.3), and therefore classifiers, that solve our classification problem without any (training) errors. To find a unique solution, one idea is to choose the separating hyperplane that maximizes the margin between the positive and negative examples. In other words, we want the positive and negative examples to be separated by a large margin (Section 12.2.1). In the following, we compute the distance between an example and a hyperplane to derive the margin. Recall that the closest point on the hyperplane to a given point (example $\boldsymbol{x}_n$) is obtained by the orthogonal projection (Section 3.8).

A classifier with large margin turns out to generalize well (Steinwart and Christmann, 2008).

12.2.1 Concept of the Margin

margin

There could be two or more closest examples to a hyperplane.

The concept of the *margin* is intuitively simple: It is the distance of the separating hyperplane to the closest examples in the dataset, assuming that the dataset is linearly separable. However, when trying to formalize this distance, there is a technical wrinkle that may be confusing. The technical wrinkle is that we need to define a scale at which to measure the distance. A potential scale is to consider the scale of the data, i.e., the raw values of $\boldsymbol{x}_n$. There are problems with this, as we could change the units of measurement of $\boldsymbol{x}_n$ and change the values in $\boldsymbol{x}_n$, and, hence, change the distance to the hyperplane. As we will see shortly, we define the scale based on the equation of the hyperplane (12.3) itself.

Consider a hyperplane $\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b$, and an example $\boldsymbol{x}_a$ as illustrated in Figure 12.4. Without loss of generality, we can consider the example $\boldsymbol{x}_a$ to be on the positive side of the hyperplane, i.e., $\langle \boldsymbol{w}, \boldsymbol{x}_a \rangle + b > 0$. We would like to compute the distance $r > 0$ of $\boldsymbol{x}_a$ from the hyperplane. We do so by considering the orthogonal projection (Section 3.8) of $\boldsymbol{x}_a$ onto the hyperplane, which we denote by $\boldsymbol{x}'_a$. Since $\boldsymbol{w}$ is orthogonal to the

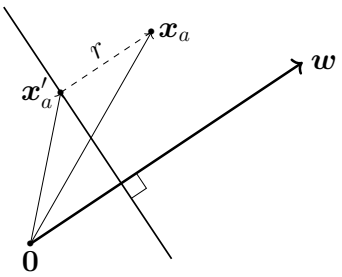


Figure 12.4 Vector addition to express distance to hyperplane: $\boldsymbol{x}_a = \boldsymbol{x}'_a + r\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}$.

hyperplane, we know that the distance r is just a scaling of this vector $\boldsymbol{w}$. If the length of $\boldsymbol{w}$ is known, then we can use this scaling factor r factor to work out the absolute distance between $\boldsymbol{x}_a$ and $\boldsymbol{x}'_a$. For convenience, we choose to use a vector of unit length (its norm is 1) and obtain this by dividing $\boldsymbol{w}$ by its norm, $\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}$. Using vector addition (Section 2.4), we obtain

$$\boldsymbol{x}_a = \boldsymbol{x}'_a + r\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}\,. \tag{12.8}$$

Another way of thinking about r is that it is the coordinate of $\boldsymbol{x}_a$ in the subspace spanned by $\boldsymbol{w}/\|\boldsymbol{w}\|$. We have now expressed the distance of $\boldsymbol{x}_a$ from the hyperplane as r, and if we choose $\boldsymbol{x}_a$ to be the point closest to the hyperplane, this distance r is the margin.

Recall that we would like the positive examples to be further than r from the hyperplane, and the negative examples to be further than distance r (in the negative direction) from the hyperplane. Analogously to the combination of (12.5) and (12.6) into (12.7), we formulate this objective as

$$y_n(\langle \boldsymbol{w}, \boldsymbol{x}_n\rangle + b) \geqslant r\,. \tag{12.9}$$

In other words, we combine the requirements that examples are at least r away from the hyperplane (in the positive and negative direction) into one single inequality.

Since we are interested only in the direction, we add an assumption to our model that the parameter vector $\boldsymbol{w}$ is of unit length, i.e., $\|\boldsymbol{w}\| = 1$, where we use the Euclidean norm $\|\boldsymbol{w}\| = \sqrt{\boldsymbol{w}^\top\boldsymbol{w}}$ (Section 3.1). This assumption also allows a more intuitive interpretation of the distance r (12.8) since it is the scaling factor of a vector of length 1.

We will see other choices of inner products (Section 3.2) in Section 12.4.

Remark. A reader familiar with other presentations of the margin would notice that our definition of $\|\boldsymbol{w}\| = 1$ is different from the standard presentation if the SVM was the one provided by Schölkopf and Smola (2002), for example. In Section 12.2.3, we will show the equivalence of both approaches. ♢

Collecting the three requirements into a single constrained optimization

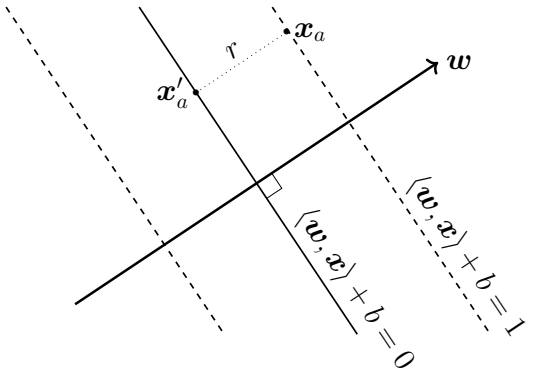


Figure 12.5 Derivation of the margin: $r = \frac{1}{\|\boldsymbol{w}\|}$.

problem, we obtain the objective

$$\begin{aligned} &\max_{\boldsymbol{w}, b, r} \quad \underbrace{r}_{\text{margin}} \\ &\text{subject to} \quad \underbrace{y_n(\langle \boldsymbol{w}, \boldsymbol{x}_n \rangle + b) \geqslant r}_{\text{data fitting}}, \underbrace{\|\boldsymbol{w}\| = 1}_{\text{normalization}}, \quad r > 0\,, \end{aligned} \tag{12.10}$$

which says that we want to maximize the margin r while ensuring that the data lies on the correct side of the hyperplane.

Remark. The concept of the margin turns out to be highly pervasive in machine learning. It was used by Vladimir Vapnik and Alexey Chervonenkis to show that when the margin is large, the "complexity" of the function class is low, and hence learning is possible (Vapnik, 2000). It turns out that the concept is useful for various different approaches for theoretically analyzing generalization error (Steinwart and Christmann, 2008; Shalev-Shwartz and Ben-David, 2014). ♢

12.2.2 Traditional Derivation of the Margin

In the previous section, we derived (12.10) by making the observation that we are only interested in the direction of $\boldsymbol{w}$ and not its length, leading to the assumption that $\|\boldsymbol{w}\| = 1$. In this section, we derive the margin maximization problem by making a different assumption. Instead of choosing that the parameter vector is normalized, we choose a scale for the data. We choose this scale such that the value of the predictor $\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b$ is 1 at the closest example. Let us also denote the example in the dataset that is closest to the hyperplane by $\boldsymbol{x}_a$.

Recall that we currently consider linearly separable data.

Figure 12.5 is identical to Figure 12.4, except that now we rescaled the axes, such that the example $\boldsymbol{x}_a$ lies exactly on the margin, i.e., $\langle \boldsymbol{w}, \boldsymbol{x}_a \rangle + b = 1$. Since $\boldsymbol{x}'_a$ is the orthogonal projection of $\boldsymbol{x}_a$ onto the hyperplane, it must by definition lie on the hyperplane, i.e.,

$$\langle \boldsymbol{w}, \boldsymbol{x}'_a \rangle + b = 0\,. \tag{12.11}$$

By substituting (12.8) into (12.11), we obtain

$$\left\langle \boldsymbol{w}, \boldsymbol{x}_a - r\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|} \right\rangle + b = 0\,. \tag{12.12}$$

Exploiting the bilinearity of the inner product (see Section 3.2), we get

$$\langle \boldsymbol{w}, \boldsymbol{x}_a \rangle + b - r\frac{\langle \boldsymbol{w}, \boldsymbol{w} \rangle}{\|\boldsymbol{w}\|} = 0\,. \tag{12.13}$$

Observe that the first term is 1 by our assumption of scale, i.e., $\langle \boldsymbol{w}, \boldsymbol{x}_a \rangle + b = 1$. From (3.16) in Section 3.1, we know that $\langle \boldsymbol{w}, \boldsymbol{w} \rangle = \|\boldsymbol{w}\|^2$. Hence, the second term reduces to $r\|\boldsymbol{w}\|$. Using these simplifications, we obtain

$$r = \frac{1}{\|\boldsymbol{w}\|}\,. \tag{12.14}$$

This means we derived the distance r in terms of the normal vector $\boldsymbol{w}$ of the hyperplane. At first glance, this equation is counterintuitive as we seem to have derived the distance from the hyperplane in terms of the length of the vector $\boldsymbol{w}$, but we do not yet know this vector. One way to think about it is to consider the distance r to be a temporary variable that we only use for this derivation. Therefore, for the rest of this section we will denote the distance to the hyperplane by $\frac{1}{\|\boldsymbol{w}\|}$. In Section 12.2.3, we will see that the choice that the margin equals 1 is equivalent to our previous assumption of $\|\boldsymbol{w}\| = 1$ in Section 12.2.1.

We can also think of the distance as the projection error that incurs when projecting $\boldsymbol{x}_a$ onto the hyperplane.

Similar to the argument to obtain (12.9), we want the positive and negative examples to be at least 1 away from the hyperplane, which yields the condition

$$y_n(\langle \boldsymbol{w}, x_n \rangle + b) \geqslant 1\,. \tag{12.15}$$

Combining the margin maximization with the fact that examples need to be on the correct side of the hyperplane (based on their labels) gives us

$$\max_{\boldsymbol{w},b} \quad \frac{1}{\|\boldsymbol{w}\|} \tag{12.16}$$

$$\text{subject to } y_n(\langle \boldsymbol{w}, \boldsymbol{x}_n \rangle + b) \geqslant 1 \quad \text{for all} \quad n = 1, \ldots, N. \tag{12.17}$$

Instead of maximizing the reciprocal of the norm as in (12.16), we often minimize the squared norm. We also often include a constant $\frac{1}{2}$ that does not affect the optimal $\boldsymbol{w}, b$ but yields a tidier form when we compute the gradient. Then, our objective becomes

The squared norm results in a convex quadratic programming problem for the SVM (Section 12.5).

$$\min_{\boldsymbol{w},b} \quad \frac{1}{2}\|\boldsymbol{w}\|^2 \tag{12.18}$$

$$\text{subject to } y_n(\langle \boldsymbol{w}, \boldsymbol{x}_n \rangle + b) \geqslant 1 \quad \text{for all} \quad n = 1, \ldots, N\,. \tag{12.19}$$

Equation (12.18) is known as the *hard margin SVM*. The reason for the expression "hard" is because the formulation does not allow for any violations of the margin condition. We will see in Section 12.2.4 that this

hard margin SVM

"hard" condition can be relaxed to accommodate violations if the data is not linearly separable.

12.2.3 Why We Can Set the Margin to 1

In Section 12.2.1, we argued that we would like to maximize some value r, which represents the distance of the closest example to the hyperplane. In Section 12.2.2, we scaled the data such that the closest example is of distance 1 to the hyperplane. In this section, we relate the two derivations, and show that they are equivalent.

Theorem 12.1. *Maximizing the margin r, where we consider normalized weights as in* (12.10)*,*

$$\begin{aligned} &\max_{\boldsymbol{w},b,r} \quad \underbrace{r}_{\text{margin}} \\ &\text{subject to} \quad \underbrace{y_n(\langle \boldsymbol{w}, \boldsymbol{x}_n \rangle + b) \geqslant r}_{\text{data fitting}}\,, \quad \underbrace{\|\boldsymbol{w}\| = 1}_{\text{normalization}}\,, \quad r > 0\,, \end{aligned} \tag{12.20}$$

is equivalent to scaling the data, such that the margin is unity:

$$\begin{aligned} &\min_{\boldsymbol{w},b} \quad \underbrace{\frac{1}{2}\|\boldsymbol{w}\|^2}_{\text{margin}} \\ &\text{subject to} \quad \underbrace{y_n(\langle \boldsymbol{w}, \boldsymbol{x}_n \rangle + b) \geqslant 1}_{\text{data fitting}}\,. \end{aligned} \tag{12.21}$$

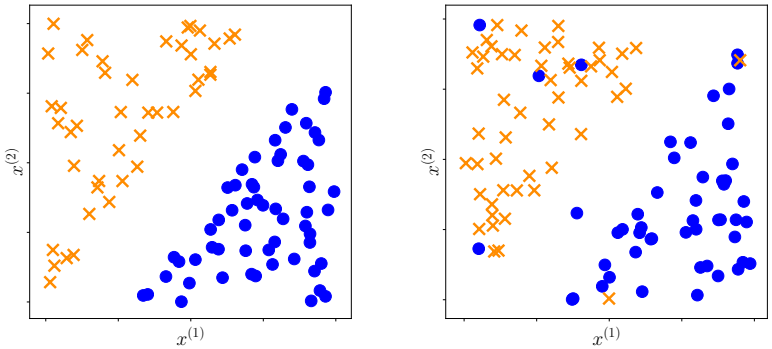
Proof Consider (12.20). Since the square is a strictly monotonic transformation for non-negative arguments, the maximum stays the same if we consider r^2 in the objective. Since $\|\boldsymbol{w}\| = 1$ we can reparametrize the equation with a new weight vector $\boldsymbol{w}'$ that is not normalized by explicitly using $\frac{\boldsymbol{w}'}{\|\boldsymbol{w}'\|}$. We obtain

$$\begin{aligned} &\max_{\boldsymbol{w}',b,r} \quad r^2 \\ &\text{subject to} \quad y_n\left(\left\langle \frac{\boldsymbol{w}'}{\|\boldsymbol{w}'\|}, \boldsymbol{x}_n \right\rangle + b\right) \geqslant r, \quad r > 0\,. \end{aligned} \tag{12.22}$$

Equation (12.22) explicitly states that the distance r is positive. Therefore, we can divide the first constraint by r, which yields

Note that $r > 0$ because we assumed linear separability, and hence there is no issue to divide by r.

$$\begin{aligned} &\max_{\boldsymbol{w}',b,r} \quad r^2 \\ &\text{subject to} \quad y_n\left(\left\langle \underbrace{\frac{\boldsymbol{w}'}{\|\boldsymbol{w}'\|\, r}}_{\boldsymbol{w}''}, \boldsymbol{x}_n \right\rangle + \underbrace{\frac{b}{r}}_{b''}\right) \geqslant 1, \quad r > 0 \end{aligned} \tag{12.23}$$



(a) Linearly separable data, with a large margin

(b) Non-linearly separable data

Figure 12.6 (a) Linearly separable and (b) non-linearly separable data.

renaming the parameters to $\boldsymbol{w}''$ and b''. Since $\boldsymbol{w}'' = \frac{\boldsymbol{w}'}{\|\boldsymbol{w}'\| r}$, rearranging for r gives

$$\|\boldsymbol{w}''\| = \left\| \frac{\boldsymbol{w}'}{\|\boldsymbol{w}'\| r} \right\| = \frac{1}{r} \cdot \left\| \frac{\boldsymbol{w}'}{\|\boldsymbol{w}'\|} \right\| = \frac{1}{r} . \tag{12.24}$$

By substituting this result into (12.23), we obtain

$$\begin{aligned} &\max_{\boldsymbol{w}'', b''} && \frac{1}{\|\boldsymbol{w}''\|^2} \\ &\text{subject to} && y_n \left(\langle \boldsymbol{w}'', \boldsymbol{x}_n \rangle + b'' \right) \geqslant 1 . \end{aligned} \tag{12.25}$$

The final step is to observe that maximizing $\frac{1}{\|\boldsymbol{w}''\|^2}$ yields the same solution as minimizing $\frac{1}{2} \|\boldsymbol{w}''\|^2$, which concludes the proof of Theorem 12.1. □

12.2.4 Soft Margin SVM: Geometric View

In the case where data is not linearly separable, we may wish to allow some examples to fall within the margin region, or even to be on the wrong side of the hyperplane as illustrated in Figure 12.6.

The model that allows for some classification errors is called the *soft margin SVM*. In this section, we derive the resulting optimization problem using geometric arguments. In Section 12.2.5, we will derive an equivalent optimization problem using the idea of a loss function. Using Lagrange multipliers (Section 7.2), we will derive the dual optimization problem of the SVM in Section 12.3. This dual optimization problem allows us to observe a third interpretation of the SVM: as a hyperplane that bisects the line between convex hulls corresponding to the positive and negative data examples (Section 12.3.2).

soft margin SVM

The key geometric idea is to introduce a *slack variable* ξ_n corresponding to each example–label pair $(\boldsymbol{x}_n, y_n)$ that allows a particular example to be within the margin or even on the wrong side of the hyperplane (refer to

slack variable

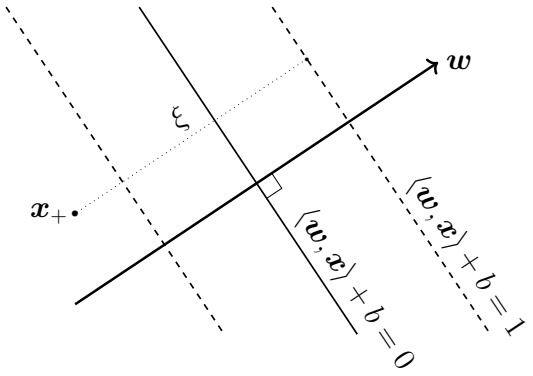


Figure 12.7 Soft margin SVM allows examples to be within the margin or on the wrong side of the hyperplane. The slack variable ξ measures the distance of a positive example $\boldsymbol{x}_+$ to the positive margin hyperplane $\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b = 1$ when $\boldsymbol{x}_+$ is on the wrong side.

Figure 12.7). We subtract the value of ξ_n from the margin, constraining ξ_n to be non-negative. To encourage correct classification of the samples, we add ξ_n to the objective

$$\min_{\boldsymbol{w}, b, \boldsymbol{\xi}} \quad \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{n=1}^{N} \xi_n \tag{12.26a}$$

$$\text{subject to} \quad y_n(\langle \boldsymbol{w}, \boldsymbol{x}_n \rangle + b) \geqslant 1 - \xi_n \tag{12.26b}$$

$$\xi_n \geqslant 0 \tag{12.26c}$$

for $n = 1, \ldots, N$. In contrast to the optimization problem (12.18) for the hard margin SVM, this one is called the *soft margin SVM*. The parameter $C > 0$ trades off the size of the margin and the total amount of slack that we have. This parameter is called the *regularization parameter* since, as we will see in the following section, the margin term in the objective function (12.26a) is a regularization term. The margin term $\|\boldsymbol{w}\|^2$ is called the *regularizer*, and in many books on numerical optimization, the regularization parameter is multiplied with this term (Section 8.2.3). This is in contrast to our formulation in this section. Here a large value of C implies low regularization, as we give the slack variables larger weight, hence giving more priority to examples that do not lie on the correct side of the margin.

soft margin SVM

regularization parameter

regularizer

There are alternative parametrizations of this regularization, which is why (12.26a) is also often referred to as the C-SVM.

Remark. In the formulation of the soft margin SVM (12.26a) $\boldsymbol{w}$ is regularized, but b is not regularized. We can see this by observing that the regularization term does not contain b. The unregularized term b complicates theoretical analysis (Steinwart and Christmann, 2008, chapter 1) and decreases computational efficiency (Fan et al., 2008). ♢

12.2.5 Soft Margin SVM: Loss Function View

Let us consider a different approach for deriving the SVM, following the principle of empirical risk minimization (Section 8.2). For the SVM, we

choose hyperplanes as the hypothesis class, that is

$$f(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b. \tag{12.27}$$

We will see in this section that the margin corresponds to the regularization term. The remaining question is, what is the *loss function*? In contrast to Chapter 9, where we consider regression problems (the output of the predictor is a real number), in this chapter, we consider binary classification problems (the output of the predictor is one of two labels $\{+1, -1\}$). Therefore, the error/loss function for each single example–label pair needs to be appropriate for binary classification. For example, the squared loss that is used for regression (9.10b) is not suitable for binary classification. loss function

Remark. The ideal loss function between binary labels is to count the number of mismatches between the prediction and the label. This means that for a predictor f applied to an example $\boldsymbol{x}_n$, we compare the output $f(\boldsymbol{x}_n)$ with the label y_n. We define the loss to be zero if they match, and one if they do not match. This is denoted by $\mathbf{1}(f(\boldsymbol{x}_n) \neq y_n)$ and is called the *zero-one loss*. Unfortunately, the zero-one loss results in a combinatorial optimization problem for finding the best parameters $\boldsymbol{w}, b$. Combinatorial optimization problems (in contrast to continuous optimization problems discussed in Chapter 7) are in general more challenging to solve. ♢ zero-one loss

What is the loss function corresponding to the SVM? Consider the error between the output of a predictor $f(\boldsymbol{x}_n)$ and the label y_n. The loss describes the error that is made on the training data. An equivalent way to derive (12.26a) is to use the *hinge loss* hinge loss

$$\ell(t) = \max\{0, 1 - t\} \quad \text{where} \quad t = yf(\boldsymbol{x}) = y(\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b)\,. \tag{12.28}$$

If $f(\boldsymbol{x})$ is on the correct side (based on the corresponding label y) of the hyperplane, and further than distance 1, this means that $t \geqslant 1$ and the hinge loss returns a value of zero. If $f(\boldsymbol{x})$ is on the correct side but too close to the hyperplane ($0 < t < 1$), the example $\boldsymbol{x}$ is within the margin, and the hinge loss returns a positive value. When the example is on the wrong side of the hyperplane ($t < 0$), the hinge loss returns an even larger value, which increases linearly. In other words, we pay a penalty once we are closer than the margin to the hyperplane, even if the prediction is correct, and the penalty increases linearly. An alternative way to express the hinge loss is by considering it as two linear pieces

$$\ell(t) = \begin{cases} 0 & \text{if} \quad t \geqslant 1 \\ 1 - t & \text{if} \quad t < 1 \end{cases}, \tag{12.29}$$

as illustrated in Figure 12.8. The loss corresponding to the hard margin SVM 12.18 is defined as

$$\ell(t) = \begin{cases} 0 & \text{if} \quad t \geqslant 1 \\ \infty & \text{if} \quad t < 1 \end{cases}. \tag{12.30}$$

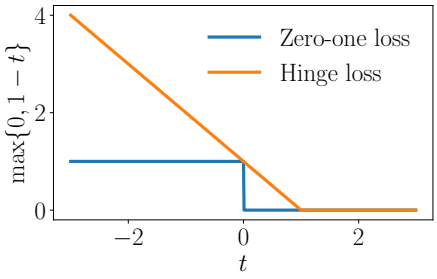


Figure 12.8 The hinge loss is a convex upper bound of zero-one loss.

This loss can be interpreted as never allowing any examples inside the margin.

For a given training set $\{(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)\}$, we seek to minimize the total loss, while regularizing the objective with ℓ_2-regularization (see Section 8.2.3). Using the hinge loss (12.28) gives us the unconstrained optimization problem

$$\min_{\boldsymbol{w}, b} \underbrace{\frac{1}{2}\|\boldsymbol{w}\|^2}_{\text{regularizer}} + \underbrace{C \sum_{n=1}^{N} \max\{0, 1 - y_n(\langle \boldsymbol{w}, \boldsymbol{x}_n \rangle + b)\}}_{\text{error term}} . \tag{12.31}$$

regularizer
loss term
error term

The first term in (12.31) is called the regularization term or the *regularizer* (see Section 8.2.3), and the second term is called the *loss term* or the *error term*. Recall from Section 12.2.4 that the term $\frac{1}{2}\|\boldsymbol{w}\|^2$ arises directly from the margin. In other words, margin maximization can be interpreted as *regularization*.

regularization

In principle, the unconstrained optimization problem in (12.31) can be directly solved with (sub-)gradient descent methods as described in Section 7.1. To see that (12.31) and (12.26a) are equivalent, observe that the hinge loss (12.28) essentially consists of two linear parts, as expressed in (12.29). Consider the hinge loss for a single example-label pair (12.28). We can equivalently replace minimization of the hinge loss over t with a minimization of a slack variable ξ with two constraints. In equation form,

$$\min_{t} \max\{0, 1 - t\} \tag{12.32}$$

is equivalent to

$$\begin{aligned} \min_{\xi, t} \quad & \xi \\ \text{subject to} \quad & \xi \geqslant 0\,, \quad \xi \geqslant 1 - t\,. \end{aligned} \tag{12.33}$$

By substituting this expression into (12.31) and rearranging one of the constraints, we obtain exactly the soft margin SVM (12.26a).

Remark. Let us contrast our choice of the loss function in this section to the loss function for linear regression in Chapter 9. Recall from Section 9.2.1 that for finding maximum likelihood estimators, we usually minimize the

negative log-likelihood. Furthermore, since the likelihood term for linear regression with Gaussian noise is Gaussian, the negative log-likelihood for each example is a squared error function. The squared error function is the loss function that is minimized when looking for the maximum likelihood solution. ♢

12.3 Dual Support Vector Machine

The description of the SVM in the previous sections, in terms of the variables $\boldsymbol{w}$ and b, is known as the primal SVM. Recall that we consider inputs $\boldsymbol{x} \in \mathbb{R}^D$ with D features. Since $\boldsymbol{w}$ is of the same dimension as $\boldsymbol{x}$, this means that the number of parameters (the dimension of $\boldsymbol{w}$) of the optimization problem grows linearly with the number of features.

In the following, we consider an equivalent optimization problem (the so-called dual view), which is independent of the number of features. Instead, the number of parameters increases with the number of examples in the training set. We saw a similar idea appear in Chapter 10, where we expressed the learning problem in a way that does not scale with the number of features. This is useful for problems where we have more features than the number of examples in the training dataset. The dual SVM also has the additional advantage that it easily allows kernels to be applied, as we shall see at the end of this chapter. The word "dual" appears often in mathematical literature, and in this particular case it refers to convex duality. The following subsections are essentially an application of convex duality, which we discussed in Section 7.2.

12.3.1 Convex Duality via Lagrange Multipliers

Recall the primal soft margin SVM (12.26a). We call the variables $\boldsymbol{w}$, b, and ξ corresponding to the primal SVM the primal variables. We use $\alpha_n \geqslant 0$ as the Lagrange multiplier corresponding to the constraint (12.26b) that the examples are classified correctly and $\gamma_n \geqslant 0$ as the Lagrange multiplier corresponding to the non-negativity constraint of the slack variable; see (12.26c). The Lagrangian is then given by

In Chapter 7, we used λ as Lagrange multipliers. In this section, we follow the notation commonly chosen in SVM literature, and use α and γ.

$$\mathfrak{L}(\boldsymbol{w}, b, \xi, \alpha, \gamma) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{n=1}^{N}\xi_n \underbrace{-\sum_{n=1}^{N}\alpha_n(y_n(\langle \boldsymbol{w}, \boldsymbol{x}_n\rangle + b) - 1 + \xi_n)}_{\text{constraint (12.26b)}} \underbrace{-\sum_{n=1}^{N}\gamma_n\xi_n}_{\text{constraint (12.26c)}} . \tag{12.34}$$

By differentiating the Lagrangian (12.34) with respect to the three primal variables $\boldsymbol{w}$, b, and ξ respectively, we obtain

$$\frac{\partial \mathfrak{L}}{\partial \boldsymbol{w}} = \boldsymbol{w}^\top - \sum_{n=1}^{N} \alpha_n y_n \boldsymbol{x}_n{}^\top , \tag{12.35}$$

$$\frac{\partial \mathfrak{L}}{\partial b} = -\sum_{n=1}^{N} \alpha_n y_n , \tag{12.36}$$

$$\frac{\partial \mathfrak{L}}{\partial \xi_n} = C - \alpha_n - \gamma_n . \tag{12.37}$$

We now find the maximum of the Lagrangian by setting each of these partial derivatives to zero. By setting (12.35) to zero, we find

$$\boldsymbol{w} = \sum_{n=1}^{N} \alpha_n y_n \boldsymbol{x}_n , \tag{12.38}$$

representer theorem

The representer theorem is actually a collection of theorems saying that the solution of minimizing empirical risk lies in the subspace (Section 2.4.3) defined by the examples.

which is a particular instance of the *representer theorem* (Kimeldorf and Wahba, 1970). Equation (12.38) states that the optimal weight vector in the primal is a linear combination of the examples $\boldsymbol{x}_n$. Recall from Section 2.6.1 that this means that the solution of the optimization problem lies in the span of training data. Additionally, the constraint obtained by setting (12.36) to zero implies that the optimal weight vector is an affine combination of the examples. The representer theorem turns out to hold for very general settings of regularized empirical risk minimization (Hofmann et al., 2008; Argyriou and Dinuzzo, 2014). The theorem has more general versions (Schölkopf et al., 2001), and necessary and sufficient conditions on its existence can be found in Yu et al. (2013).

Remark. The representer theorem (12.38) also provides an explanation of the name "support vector machine." The examples $\boldsymbol{x}_n$, for which the corresponding parameters $\alpha_n = 0$, do not contribute to the solution $\boldsymbol{w}$ at all. The other examples, where $\alpha_n > 0$, are called *support vectors* since they "support" the hyperplane. ♢

support vector

By substituting the expression for $\boldsymbol{w}$ into the Lagrangian (12.34), we obtain the dual

$$\begin{aligned}\mathfrak{D}(\xi, \alpha, \gamma) = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j \left\langle \boldsymbol{x}_i, \boldsymbol{x}_j \right\rangle - \sum_{i=1}^{N} y_i \alpha_i \left\langle \sum_{j=1}^{N} y_j \alpha_j \boldsymbol{x}_j, \boldsymbol{x}_i \right\rangle \\ + C\sum_{i=1}^{N} \xi_i - b\sum_{i=1}^{N} y_i \alpha_i + \sum_{i=1}^{N} \alpha_i - \sum_{i=1}^{N} \alpha_i \xi_i - \sum_{i=1}^{N} \gamma_i \xi_i .\end{aligned} \tag{12.39}$$

Note that there are no longer any terms involving the primal variable $\boldsymbol{w}$. By setting (12.36) to zero, we obtain $\sum_{n=1}^{N} y_n \alpha_n = 0$. Therefore, the term involving b also vanishes. Recall that inner products are symmetric and

bilinear (see Section 3.2). Therefore, the first two terms in (12.39) are over the same objects. These terms (colored blue) can be simplified, and we obtain the Lagrangian

$$\mathfrak{D}(\xi, \alpha, \gamma) = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle + \sum_{i=1}^{N} \alpha_i + \sum_{i=1}^{N} (C - \alpha_i - \gamma_i)\xi_i \,. \tag{12.40}$$

The last term in this equation is a collection of all terms that contain slack variables ξ_i. By setting (12.37) to zero, we see that the last term in (12.40) is also zero. Furthermore, by using the same equation and recalling that the Lagrange multipliers γ_i are non-negative, we conclude that $\alpha_i \leqslant C$. We now obtain the dual optimization problem of the SVM, which is expressed exclusively in terms of the Lagrange multipliers α_i. Recall from Lagrangian duality (Definition 7.1) that we maximize the dual problem. This is equivalent to minimizing the negative dual problem, such that we end up with the *dual SVM*

dual SVM

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} y_i y_j \alpha_i \alpha_j \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle - \sum_{i=1}^{N} \alpha_i \\ \text{subject to} \quad & \sum_{i=1}^{N} y_i \alpha_i = 0 \\ & 0 \leqslant \alpha_i \leqslant C \quad \text{for all} \quad i = 1, \ldots, N \,. \end{aligned} \tag{12.41}$$

The equality constraint in (12.41) is obtained from setting (12.36) to zero. The inequality constraint $\alpha_i \geqslant 0$ is the condition imposed on Lagrange multipliers of inequality constraints (Section 7.2). The inequality constraint $\alpha_i \leqslant C$ is discussed in the previous paragraph.

The set of inequality constraints in the SVM are called "box constraints" because they limit the vector $\boldsymbol{\alpha} = [\alpha_1, \cdots, \alpha_N]^\top \in \mathbb{R}^N$ of Lagrange multipliers to be inside the box defined by 0 and C on each axis. These axis-aligned boxes are particularly efficient to implement in numerical solvers (Dostál, 2009, chapter 5).

Once we obtain the dual parameters $\boldsymbol{\alpha}$, we can recover the primal parameters $\boldsymbol{w}$ by using the representer theorem (12.38). Let us call the optimal primal parameter $\boldsymbol{w}^*$. However, there remains the question on how to obtain the parameter b^*. Consider an example $\boldsymbol{x}_n$ that lies exactly on the margin's boundary, i.e., $\langle \boldsymbol{w}^*, \boldsymbol{x}_n \rangle + b = y_n$. Recall that y_n is either $+1$ or -1. Therefore, the only unknown is b, which can be computed by

$$b^* = y_n - \langle \boldsymbol{w}^*, \boldsymbol{x}_n \rangle \,. \tag{12.42}$$

It turns out that examples that lie exactly on the margin are examples whose dual parameters lie strictly inside the box constraints, $0 < \alpha_i < C$. This is derived using the Karush Kuhn Tucker conditions, for example in Schölkopf and Smola (2002).

Remark. In principle, there may be no examples that lie exactly on the margin. In this case, we should compute $|y_n - \langle \boldsymbol{w}^*, \boldsymbol{x}_n \rangle|$ for all support vectors and take the median value of this absolute value difference to be

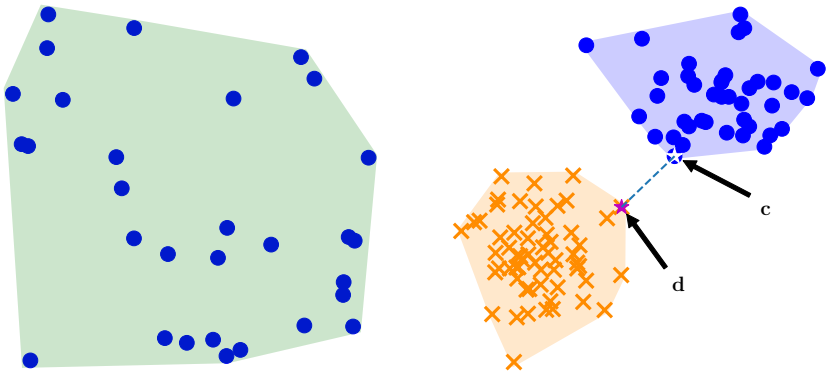


Figure 12.9 Convex hulls. (a) Convex hull of points, some of which lie within the boundary; (b) convex hulls around positive and negative examples.

(a) Convex hull.

(b) Convex hulls around positive (blue) and negative (orange) examples. The distance between the two convex sets is the length of the difference vector $\boldsymbol{c} - \boldsymbol{d}$.

the value of b^*. A derivation of this can be found in `http://fouryears.eu/2012/06/07/the-svm-bias-term-conspiracy/`. ♢

12.3.2 Dual SVM: Convex Hull View

Another approach to obtain the dual SVM is to consider an alternative geometric argument. Consider the set of examples $\boldsymbol{x}_n$ with the same label. We would like to build a convex set that contains all the examples such that it is the smallest possible set. This is called the convex hull and is illustrated in Figure 12.9.

Let us first build some intuition about a convex combination of points. Consider two points $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ and corresponding non-negative weights $\alpha_1, \alpha_2 \geqslant 0$ such that $\alpha_1 + \alpha_2 = 1$. The equation $\alpha_1\boldsymbol{x}_1 + \alpha_2\boldsymbol{x}_2$ describes each point on a line between $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$. Consider what happens when we add a third point $\boldsymbol{x}_3$ along with a weight $\alpha_3 \geqslant 0$ such that $\sum_{n=1}^{3} \alpha_n = 1$. The convex combination of these three points $\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3$ spans a two-dimensional area. The *convex hull* of this area is the triangle formed by the edges corresponding to each pair of of points. As we add more points, and the number of points becomes greater than the number of dimensions, some of the points will be inside the convex hull, as we can see in Figure 12.9(a).

convex hull

In general, building a convex convex hull can be done by introducing non-negative weights $\alpha_n \geqslant 0$ corresponding to each example $\boldsymbol{x}_n$. Then the convex hull can be described as the set

$$\operatorname{conv}(\boldsymbol{X}) = \left\{ \sum_{n=1}^{N} \alpha_n \boldsymbol{x}_n \right\} \quad \text{with} \quad \sum_{n=1}^{N} \alpha_n = 1 \quad \text{and} \quad \alpha_n \geqslant 0, \tag{12.43}$$

for all $n = 1, \ldots, N$. If the two clouds of points corresponding to the positive and negative classes are separated, then the convex hulls do not overlap. Given the training data $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)$, we form two convex hulls, corresponding to the positive and negative classes respectively. We pick a point $\boldsymbol{c}$, which is in the convex hull of the set of positive examples, and is closest to the negative class distribution. Similarly, we pick a point $\boldsymbol{d}$ in the convex hull of the set of negative examples and is closest to the positive class distribution; see Figure 12.9(b). We define a difference vector between $\boldsymbol{d}$ and $\boldsymbol{c}$ as

$$\boldsymbol{w} := \boldsymbol{c} - \boldsymbol{d}\,. \tag{12.44}$$

Picking the points $\boldsymbol{c}$ and $\boldsymbol{d}$ as in the preceding cases, and requiring them to be closest to each other is equivalent to minimizing the length/norm of $\boldsymbol{w}$, so that we end up with the corresponding optimization problem

$$\arg\min_{\boldsymbol{w}} \|\boldsymbol{w}\| = \arg\min_{\boldsymbol{w}} \frac{1}{2}\|\boldsymbol{w}\|^2\,. \tag{12.45}$$

Since $\boldsymbol{c}$ must be in the positive convex hull, it can be expressed as a convex combination of the positive examples, i.e., for non-negative coefficients α_n^+

$$\boldsymbol{c} = \sum_{n:y_n=+1} \alpha_n^+ \boldsymbol{x}_n\,. \tag{12.46}$$

In (12.46), we use the notation $n : y_n = +1$ to indicate the set of indices n for which $y_n = +1$. Similarly, for the examples with negative labels, we obtain

$$\boldsymbol{d} = \sum_{n:y_n=-1} \alpha_n^- \boldsymbol{x}_n\,. \tag{12.47}$$

By substituting (12.44), (12.46), and (12.47) into (12.45), we obtain the objective

$$\min_{\boldsymbol{\alpha}} \frac{1}{2}\left\| \sum_{n:y_n=+1} \alpha_n^+ \boldsymbol{x}_n - \sum_{n:y_n=-1} \alpha_n^- \boldsymbol{x}_n \right\|^2\,. \tag{12.48}$$

Let $\boldsymbol{\alpha}$ be the set of all coefficients, i.e., the concatenation of $\boldsymbol{\alpha}^+$ and $\boldsymbol{\alpha}^-$. Recall that we require that for each convex hull that their coefficients sum to one,

$$\sum_{n:y_n=+1} \alpha_n^+ = 1 \quad \text{and} \quad \sum_{n:y_n=-1} \alpha_n^- = 1\,. \tag{12.49}$$

This implies the constraint

$$\sum_{n=1}^{N} y_n \alpha_n = 0\,. \tag{12.50}$$

This result can be seen by multiplying out the individual classes

$$\sum_{n=1}^{N} y_n \alpha_n = \sum_{n:y_n=+1} (+1)\alpha_n^+ + \sum_{n:y_n=-1} (-1)\alpha_n^- \tag{12.51a}$$

$$= \sum_{n:y_n=+1} \alpha_n^+ - \sum_{n:y_n=-1} \alpha_n^- = 1 - 1 = 0\,. \tag{12.51b}$$

The objective function (12.48) and the constraint (12.50), along with the assumption that $\boldsymbol{\alpha} \geqslant \mathbf{0}$, give us a constrained (convex) optimization problem. This optimization problem can be shown to be the same as that of the dual hard margin SVM (Bennett and Bredensteiner, 2000a).

reduced hull

Remark. To obtain the soft margin dual, we consider the reduced hull. The *reduced hull* is similar to the convex hull but has an upper bound to the size of the coefficients $\boldsymbol{\alpha}$. The maximum possible value of the elements of $\boldsymbol{\alpha}$ restricts the size that the convex hull can take. In other words, the bound on $\boldsymbol{\alpha}$ shrinks the convex hull to a smaller volume (Bennett and Bredensteiner, 2000b). ♢

12.4 Kernels

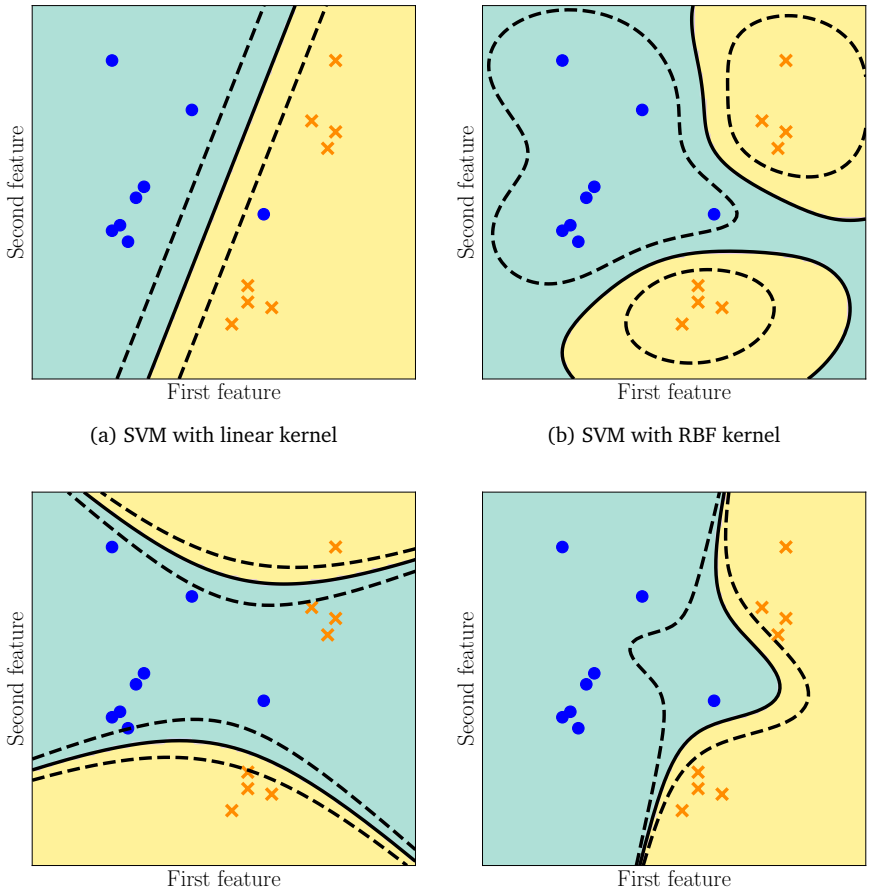
Consider the formulation of the dual SVM (12.41). Notice that the inner product in the objective occurs only between examples $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. There are no inner products between the examples and the parameters. Therefore, if we consider a set of features $\boldsymbol{\phi}(\boldsymbol{x}_i)$ to represent $\boldsymbol{x}_i$, the only change in the dual SVM will be to replace the inner product. This modularity, where the choice of the classification method (the SVM) and the choice of the feature representation $\boldsymbol{\phi}(\boldsymbol{x})$ can be considered separately, provides flexibility for us to explore the two problems independently. In this section, we discuss the representation $\boldsymbol{\phi}(\boldsymbol{x})$ and briefly introduce the idea of kernels, but do not go into the technical details.

Since $\boldsymbol{\phi}(\boldsymbol{x})$ could be a non-linear function, we can use the SVM (which assumes a linear classifier) to construct classifiers that are nonlinear in the examples $\boldsymbol{x}_n$. This provides a second avenue, in addition to the soft margin, for users to deal with a dataset that is not linearly separable. It turns out that there are many algorithms and statistical methods that have this property that we observed in the dual SVM: the only inner products are those that occur between examples. Instead of *explicitly* defining a non-linear feature map $\boldsymbol{\phi}(\cdot)$ and computing the resulting inner product between examples $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, we define a similarity function $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. For a certain class of similarity functions, called *kernels*, the similarity function *implicitly* defines a non-linear feature map $\boldsymbol{\phi}(\cdot)$. Kernels are by definition functions $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ for which there exists a Hilbert space $\mathcal{H}$ and $\boldsymbol{\phi} : \mathcal{X} \to \mathcal{H}$ a feature map such that

kernel

The inputs $\mathcal{X}$ of the kernel function can be very general and are not necessarily restricted to $\mathbb{R}^D$.

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \boldsymbol{\phi}(\boldsymbol{x}_i), \boldsymbol{\phi}(\boldsymbol{x}_j) \rangle_{\mathcal{H}}\,. \tag{12.52}$$



(a) SVM with linear kernel

(b) SVM with RBF kernel

(c) SVM with polynomial (degree 2) kernel

(d) SVM with polynomial (degree 3) kernel

Figure 12.10 SVM with different kernels. Note that while the decision boundary is nonlinear, the underlying problem being solved is for a linear separating hyperplane (albeit with a nonlinear kernel).

There is a unique reproducing kernel Hilbert space associated with every kernel k (Aronszajn, 1950; Berlinet and Thomas-Agnan, 2004). In this unique association, $\phi(\boldsymbol{x}) = k(\cdot, \boldsymbol{x})$ is called the *canonical feature map*. The generalization from an inner product to a kernel function (12.52) is known as the *kernel trick* (Schölkopf and Smola, 2002; Shawe-Taylor and Cristianini, 2004), as it hides away the explicit non-linear feature map.

canonical feature map

kernel trick

The matrix $\boldsymbol{K} \in \mathbb{R}^{N \times N}$, resulting from the inner products or the application of $k(\cdot, \cdot)$ to a dataset, is called the *Gram matrix*, and is often just referred to as the *kernel matrix*. Kernels must be symmetric and positive semidefinite functions so that every kernel matrix $\boldsymbol{K}$ is symmetric and positive semidefinite (Section 3.2.3):

Gram matrix

kernel matrix

$$\forall \boldsymbol{z} \in \mathbb{R}^N : \boldsymbol{z}^\top \boldsymbol{K} \boldsymbol{z} \geqslant 0\,. \tag{12.53}$$

Some popular examples of kernels for multivariate real-valued data $\boldsymbol{x}_i \in \mathbb{R}^D$ are the polynomial kernel, the Gaussian radial basis function kernel, and the rational quadratic kernel (Schölkopf and Smola, 2002; Rasmussen

and Williams, 2006). Figure 12.10 illustrates the effect of different kernels on separating hyperplanes on an example dataset. Note that we are still solving for hyperplanes, that is, the hypothesis class of functions are still linear. The non-linear surfaces are due to the kernel function.

Remark. Unfortunately for the fledgling machine learner, there are multiple meanings of the word "kernel." In this chapter, the word "kernel" comes from the idea of the reproducing kernel Hilbert space (RKHS) (Aronszajn, 1950; Saitoh, 1988). We have discussed the idea of the kernel in linear algebra (Section 2.7.3), where the kernel is another word for the null space. The third common use of the word "kernel" in machine learning is the smoothing kernel in kernel density estimation (Section 11.5). ♢

Since the explicit representation $\boldsymbol{\phi}(\boldsymbol{x})$ is mathematically equivalent to the kernel representation $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$, a practitioner will often design the kernel function such that it can be computed more efficiently than the inner product between explicit feature maps. For example, consider the polynomial kernel (Schölkopf and Smola, 2002), where the number of terms in the explicit expansion grows very quickly (even for polynomials of low degree) when the input dimension is large. The kernel function only requires one multiplication per input dimension, which can provide significant computational savings. Another example is the Gaussian radial basis function kernel (Schölkopf and Smola, 2002; Rasmussen and Williams, 2006), where the corresponding feature space is infinite dimensional. In this case, we cannot explicitly represent the feature space but can still compute similarities between a pair of examples using the kernel.

The choice of kernel, as well as the parameters of the kernel, is often chosen using nested cross-validation (Section 8.6.1).

Another useful aspect of the kernel trick is that there is no need for the original data to be already represented as multivariate real-valued data. Note that the inner product is defined on the output of the function $\boldsymbol{\phi}(\cdot)$, but does not restrict the input to real numbers. Hence, the function $\boldsymbol{\phi}(\cdot)$ and the kernel function $k(\cdot, \cdot)$ can be defined on any object, e.g., sets, sequences, strings, graphs, and distributions (Ben-Hur et al., 2008; Gärtner, 2008; Shi et al., 2009; Sriperumbudur et al., 2010; Vishwanathan et al., 2010).

12.5 Numerical Solution

We conclude our discussion of SVMs by looking at how to express the problems derived in this chapter in terms of the concepts presented in Chapter 7. We consider two different approaches for finding the optimal solution for the SVM. First we consider the loss view of SVM 8.2.2 and express this as an unconstrained optimization problem. Then we express the constrained versions of the primal and dual SVMs as quadratic programs in standard form 7.3.2.

Consider the loss function view of the SVM (12.31). This is a convex unconstrained optimization problem, but the hinge loss (12.28) is not dif-

ferentiable. Therefore, we apply a subgradient approach for solving it. However, the hinge loss is differentiable almost everywhere, except for one single point at the hinge $t = 1$. At this point, the gradient is a set of possible values that lie between 0 and -1. Therefore, the subgradient g of the hinge loss is given by

$$g(t) = \begin{cases} -1 & t < 1 \\ [-1, 0] & t = 1 \\ 0 & t > 1 \end{cases} . \tag{12.54}$$

Using this subgradient, we can apply the optimization methods presented in Section 7.1.

Both the primal and the dual SVM result in a convex quadratic programming problem (constrained optimization). Note that the primal SVM in (12.26a) has optimization variables that have the size of the dimension D of the input examples. The dual SVM in (12.41) has optimization variables that have the size of the number N of examples.

To express the primal SVM in the standard form (7.45) for quadratic programming, let us assume that we use the dot product (3.5) as the inner product. We rearrange the equation for the primal SVM (12.26a), such that the optimization variables are all on the right and the inequality of the constraint matches the standard form. This yields the optimization

Recall from Section 3.2 that we use the phrase dot product to mean the inner product on Euclidean vector space.

$$\begin{aligned} \min_{\boldsymbol{w}, b, \boldsymbol{\xi}} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{n=1}^{N} \xi_n \\ \text{subject to} \quad & -y_n \boldsymbol{x}_n^\top \boldsymbol{w} - y_n b - \xi_n \leqslant -1 \\ & -\xi_n \leqslant 0 \end{aligned} \tag{12.55}$$

$n = 1, \ldots, N$. By concatenating the variables $\boldsymbol{w}, b, \boldsymbol{x}_n$ into a single vector, and carefully collecting the terms, we obtain the following matrix form of the soft margin SVM:

$$\begin{aligned} \min_{\boldsymbol{w}, b, \boldsymbol{\xi}} \quad & \frac{1}{2} \begin{bmatrix} \boldsymbol{w} \\ b \\ \boldsymbol{\xi} \end{bmatrix}^\top \begin{bmatrix} \boldsymbol{I}_D & \boldsymbol{0}_{D,N+1} \\ \boldsymbol{0}_{N+1,D} & \boldsymbol{0}_{N+1,N+1} \end{bmatrix} \begin{bmatrix} \boldsymbol{w} \\ b \\ \boldsymbol{\xi} \end{bmatrix} + \begin{bmatrix} \boldsymbol{0}_{D+1,1} & C\boldsymbol{1}_{N,1} \end{bmatrix}^\top \begin{bmatrix} \boldsymbol{w} \\ b \\ \boldsymbol{\xi} \end{bmatrix} \\ \text{subject to} \quad & \begin{bmatrix} -\boldsymbol{Y}\boldsymbol{X} & -\boldsymbol{y} & -\boldsymbol{I}_N \\ \boldsymbol{0}_{N,D+1} & & -\boldsymbol{I}_N \end{bmatrix} \begin{bmatrix} \boldsymbol{w} \\ b \\ \boldsymbol{\xi} \end{bmatrix} \leqslant \begin{bmatrix} -\boldsymbol{1}_{N,1} \\ \boldsymbol{0}_{N,1} \end{bmatrix} . \end{aligned} \tag{12.56}$$

In the preceding optimization problem, the minimization is over the parameters $[\boldsymbol{w}^\top, b, \boldsymbol{\xi}^\top]^\top \in \mathbb{R}^{D+1+N}$, and we use the notation: $\boldsymbol{I}_m$ to represent the identity matrix of size $m \times m$, $\boldsymbol{0}_{m,n}$ to represent the matrix of zeros of size $m \times n$, and $\boldsymbol{1}_{m,n}$ to represent the matrix of ones of size $m \times n$. In addition, $\boldsymbol{y}$ is the vector of labels $[y_1, \cdots, y_N]^\top$, $\boldsymbol{Y} = \operatorname{diag}(\boldsymbol{y})$

is an N by N matrix where the elements of the diagonal are from $\boldsymbol{y}$, and $\boldsymbol{X} \in \mathbb{R}^{N \times D}$ is the matrix obtained by concatenating all the examples.

We can similarly perform a collection of terms for the dual version of the SVM (12.41). To express the dual SVM in standard form, we first have to express the kernel matrix $\boldsymbol{K}$ such that each entry is $K_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$. If we have an explicit feature representation $\boldsymbol{x}_i$ then we define $K_{ij} = \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$. For convenience of notation we introduce a matrix with zeros everywhere except on the diagonal, where we store the labels, that is, $\boldsymbol{Y} = \operatorname{diag}(\boldsymbol{y})$. The dual SVM can be written as

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^\top \boldsymbol{Y}\boldsymbol{K}\boldsymbol{Y}\boldsymbol{\alpha} - \mathbf{1}_{N,1}^\top \boldsymbol{\alpha} \\ \text{subject to} \quad & \begin{bmatrix} \boldsymbol{y}^\top \\ -\boldsymbol{y}^\top \\ -\boldsymbol{I}_N \\ \boldsymbol{I}_N \end{bmatrix} \boldsymbol{\alpha} \leqslant \begin{bmatrix} \mathbf{0}_{N+2,1} \\ C\mathbf{1}_{N,1} \end{bmatrix} . \end{aligned} \tag{12.57}$$

Remark. In Sections 7.3.1 and 7.3.2, we introduced the standard forms of the constraints to be inequality constraints. We will express the dual SVM's equality constraint as two inequality constraints, i.e.,

$$\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} \quad \text{is replaced by} \quad \boldsymbol{A}\boldsymbol{x} \leqslant \boldsymbol{b} \quad \text{and} \quad \boldsymbol{A}\boldsymbol{x} \geqslant \boldsymbol{b}\,. \tag{12.58}$$

Particular software implementations of convex optimization methods may provide the ability to express equality constraints. ♢

Since there are many different possible views of the SVM, there are many approaches for solving the resulting optimization problem. The approach presented here, expressing the SVM problem in standard convex optimization form, is not often used in practice. The two main implementations of SVM solvers are Chang and Lin (2011) (which is open source) and Joachims (1999). Since SVMs have a clear and well-defined optimization problem, many approaches based on numerical optimization techniques (Nocedal and Wright, 2006) can be applied (Shawe-Taylor and Sun, 2011).

12.6 Further Reading

The SVM is one of many approaches for studying binary classification. Other approaches include the perceptron, logistic regression, Fisher discriminant, nearest neighbor, naive Bayes, and random forest (Bishop, 2006; Murphy, 2012). A short tutorial on SVMs and kernels on discrete sequences can be found in Ben-Hur et al. (2008). The development of SVMs is closely linked to empirical risk minimization, discussed in Section 8.2. Hence, the SVM has strong theoretical properties (Vapnik, 2000; Steinwart and Christmann, 2008). The book about kernel methods (Schölkopf and Smola, 2002) includes many details of support vector machines and

how to optimize them. A broader book about kernel methods (Shawe-Taylor and Cristianini, 2004) also includes many linear algebra approaches for different machine learning problems.

An alternative derivation of the dual SVM can be obtained using the idea of the Legendre–Fenchel transform (Section 7.3.3). The derivation considers each term of the unconstrained formulation of the SVM (12.31) separately and calculates their convex conjugates (Rifkin and Lippert, 2007). Readers interested in the functional analysis view (also the regularization methods view) of SVMs are referred to the work by Wahba (1990). Theoretical exposition of kernels (Aronszajn, 1950; Schwartz, 1964; Saitoh, 1988; Manton and Amblard, 2015) requires a basic grounding in linear operators (Akhiezer and Glazman, 1993). The idea of kernels have been generalized to Banach spaces (Zhang et al., 2009) and Kreĭn spaces (Ong et al., 2004; Loosli et al., 2016).

Observe that the hinge loss has three equivalent representations, as shown in (12.28) and (12.29), as well as the constrained optimization problem in (12.33). The formulation (12.28) is often used when comparing the SVM loss function with other loss functions (Steinwart, 2007). The two-piece formulation (12.29) is convenient for computing subgradients, as each piece is linear. The third formulation (12.33), as seen in Section 12.5, enables the use of convex quadratic programming (Section 7.3.2) tools.

Since binary classification is a well-studied task in machine learning, other words are also sometimes used, such as discrimination, separation, and decision. Furthermore, there are three quantities that can be the output of a binary classifier. First is the output of the linear function itself (often called the score), which can take any real value. This output can be used for ranking the examples, and binary classification can be thought of as picking a threshold on the ranked examples (Shawe-Taylor and Cristianini, 2004). The second quantity that is often considered the output of a binary classifier is the output determined after it is passed through a non-linear function to constrain its value to a bounded range, for example in the interval $[0, 1]$. A common non-linear function is the sigmoid function (Bishop, 2006). When the non-linearity results in well-calibrated probabilities (Gneiting and Raftery, 2007; Reid and Williamson, 2011), this is called class probability estimation. The third output of a binary classifier is the final binary decision $\{+1, -1\}$, which is the one most commonly assumed to be the output of the classifier.

The SVM is a binary classifier that does not naturally lend itself to a probabilistic interpretation. There are several approaches for converting the raw output of the linear function (the score) into a calibrated class probability estimate ($P(Y = 1|X = \boldsymbol{x})$) that involve an additional calibration step (Platt, 2000; Zadrozny and Elkan, 2001; Lin et al., 2007). From the training perspective, there are many related probabilistic approaches. We mentioned at the end of Section 12.2.5 that there is a re-

lationship between loss function and the likelihood (also compare Sections 8.2 and 8.3). The maximum likelihood approach corresponding to a well-calibrated transformation during training is called logistic regression, which comes from a class of methods called generalized linear models. Details of logistic regression from this point of view can be found in Agresti (2002, chapter 5) and McCullagh and Nelder (1989, chapter 4). Naturally, one could take a more Bayesian view of the classifier output by estimating a posterior distribution using Bayesian logistic regression. The Bayesian view also includes the specification of the prior, which includes design choices such as conjugacy (Section 6.6.1) with the likelihood. Additionally, one could consider latent functions as priors, which results in Gaussian process classification (Rasmussen and Williams, 2006, chapter 3).

References

Abel, Niels H. 1826. *Démonstration de l'Impossibilité de la Résolution Algébrique des Équations Générales qui Passent le Quatrième Degré*. Grøndahl and Søn.

Adhikari, Ani, and DeNero, John. 2018. *Computational and Inferential Thinking: The Foundations of Data Science*. Gitbooks.

Agarwal, Arvind, and Daumé III, Hal. 2010. A Geometric View of Conjugate Priors. *Machine Learning*, **81**(1), 99–113.

Agresti, A. 2002. *Categorical Data Analysis*. Wiley.

Akaike, Hirotugu. 1974. A New Look at the Statistical Model Identification. *IEEE Transactions on Automatic Control*, **19**(6), 716–723.

Akhiezer, Naum I., and Glazman, Izrail M. 1993. *Theory of Linear Operators in Hilbert Space*. Dover Publications.

Alpaydin, Ethem. 2010. *Introduction to Machine Learning*. MIT Press.

Amari, Shun-ichi. 2016. *Information Geometry and Its Applications*. Springer.

Argyriou, Andreas, and Dinuzzo, Francesco. 2014. A Unifying View of Representer Theorems. In: *Proceedings of the International Conference on Machine Learning*.

Aronszajn, Nachman. 1950. Theory of Reproducing Kernels. *Transactions of the American Mathematical Society*, **68**, 337–404.

Axler, Sheldon. 2015. *Linear Algebra Done Right*. Springer.

Bakir, Gökhan, Hofmann, Thomas, Schölkopf, Bernhard, Smola, Alexander J., Taskar, Ben, and Vishwanathan, S. V. N. (eds). 2007. *Predicting Structured Data*. MIT Press.

Barber, David. 2012. *Bayesian Reasoning and Machine Learning*. Cambridge University Press.

Barndorff-Nielsen, Ole. 2014. *Information and Exponential Families: In Statistical Theory*. Wiley.

Bartholomew, David, Knott, Martin, and Moustaki, Irini. 2011. *Latent Variable Models and Factor Analysis: A Unified Approach*. Wiley.

Baydin, Atılım G., Pearlmutter, Barak A., Radul, Alexey A., and Siskind, Jeffrey M. 2018. Automatic Differentiation in Machine Learning: A Survey. *Journal of Machine Learning Research*, **18**, 1–43.

Beck, Amir, and Teboulle, Marc. 2003. Mirror Descent and Nonlinear Projected Subgradient Methods for Convex Optimization. *Operations Research Letters*, **31**(3), 167–175.

Belabbas, Mohamed-Ali, and Wolfe, Patrick J. 2009. Spectral Methods in Machine Learning and New Strategies for Very Large Datasets. *Proceedings of the National Academy of Sciences*, 0810600105.

Belkin, Mikhail, and Niyogi, Partha. 2003. Laplacian Eigenmaps for Dimensionality Reduction and Data Representation. *Neural Computation*, **15**(6), 1373–1396.

Ben-Hur, Asa, Ong, Cheng Soon, Sonnenburg, Sören, Schölkopf, Bernhard, and Rätsch, Gunnar. 2008. Support Vector Machines and Kernels for Computational Biology. *PLoS Computational Biology*, **4**(10), e1000173.

Bennett, Kristin P., and Bredensteiner, Erin J. 2000a. Duality and Geometry in SVM Classifiers. In: *Proceedings of the International Conference on Machine Learning*.

Bennett, Kristin P., and Bredensteiner, Erin J. 2000b. Geometry in Learning. Pages 132–145 of: *Geometry at Work*. Mathematical Association of America.

Berlinet, Alain, and Thomas-Agnan, Christine. 2004. *Reproducing Kernel Hilbert Spaces in Probability and Statistics*. Springer.

Bertsekas, Dimitri P. 1999. *Nonlinear Programming*. Athena Scientific.

Bertsekas, Dimitri P. 2009. *Convex Optimization Theory*. Athena Scientific.

Bickel, Peter J., and Doksum, Kjell. 2006. *Mathematical Statistics, Basic Ideas and Selected Topics*. Vol. 1. Prentice Hall.

Bickson, Danny, Dolev, Danny, Shental, Ori, Siegel, Paul H., and Wolf, Jack K. 2007. Linear Detection via Belief Propagation. In: *Proceedings of the Annual Allerton Conference on Communication, Control, and Computing*.

Billingsley, Patrick. 1995. *Probability and Measure*. Wiley.

Bishop, Christopher M. 1995. *Neural Networks for Pattern Recognition*. Clarendon Press.

Bishop, Christopher M. 1999. Bayesian PCA. In: *Advances in Neural Information Processing Systems*.

Bishop, Christopher M. 2006. *Pattern Recognition and Machine Learning*. Springer.

Blei, David M., Kucukelbir, Alp, and McAuliffe, Jon D. 2017. Variational Inference: A Review for Statisticians. *Journal of the American Statistical Association*, **112**(518), 859–877.

Blum, Arvim, and Hardt, Moritz. 2015. The Ladder: A Reliable Leaderboard for Machine Learning Competitions. In: *International Conference on Machine Learning*.

Bonnans, J. Frédéric, Gilbert, J. Charles, Lemaréchal, Claude, and Sagastizábal, Claudia A. 2006. *Numerical Optimization: Theoretical and Practical Aspects*. Springer.

Borwein, Jonathan M., and Lewis, Adrian S. 2006. *Convex Analysis and Nonlinear Optimization*. 2nd edn. Canadian Mathematical Society.

Bottou, Léon. 1998. Online Algorithms and Stochastic Approximations. Pages 9–42 of: *Online Learning and Neural Networks*. Cambridge University Press.

Bottou, Léon, Curtis, Frank E., and Nocedal, Jorge. 2018. Optimization Methods for Large-Scale Machine Learning. *SIAM Review*, **60**(2), 223–311.

Boucheron, Stephane, Lugosi, Gabor, and Massart, Pascal. 2013. *Concentration Inequalities: A Nonasymptotic Theory of Independence*. Oxford University Press.

Boyd, Stephen, and Vandenberghe, Lieven. 2004. *Convex Optimization*. Cambridge University Press.

Boyd, Stephen, and Vandenberghe, Lieven. 2018. *Introduction to Applied Linear Algebra*. Cambridge University Press.

Brochu, Eric, Cora, Vlad M., and de Freitas, Nando. 2009. *A Tutorial on Bayesian Optimization of Expensive Cost Functions, with Application to Active User Modeling and Hierarchical Reinforcement Learning*. Tech. rept. TR-2009-023. Department of Computer Science, University of British Columbia.

Brooks, Steve, Gelman, Andrew, Jones, Galin L., and Meng, Xiao-Li (eds). 2011. *Handbook of Markov Chain Monte Carlo*. Chapman and Hall/CRC.

Brown, Lawrence D. 1986. *Fundamentals of Statistical Exponential Families: With Applications in Statistical Decision Theory*. Institute of Mathematical Statistics.

Bryson, Arthur E. 1961. A Gradient Method for Optimizing Multi-Stage Allocation Processes. In: *Proceedings of the Harvard University Symposium on Digital Computers and Their Applications*.

Bubeck, Sébastien. 2015. Convex Optimization: Algorithms and Complexity. *Foundations and Trends in Machine Learning*, **8**(3-4), 231–357.

Bühlmann, Peter, and Van De Geer, Sara. 2011. *Statistics for High-Dimensional Data*. Springer.

Burges, Christopher. 2010. Dimension Reduction: A Guided Tour. *Foundations and Trends in Machine Learning*, **2**(4), 275–365.

Carroll, J Douglas, and Chang, Jih-Jie. 1970. Analysis of Individual Differences in Multidimensional Scaling via an N-Way Generalization of "Eckart-Young" Decomposition. *Psychometrika*, **35**(3), 283–319.

Casella, George, and Berger, Roger L. 2002. *Statistical Inference*. Duxbury.

Çinlar, Erhan. 2011. *Probability and Stochastics*. Springer.

Chang, Chih-Chung, and Lin, Chih-Jen. 2011. LIBSVM: A Library for Support Vector Machines. *ACM Transactions on Intelligent Systems and Technology*, **2**, 27:1–27:27.

Cheeseman, Peter. 1985. In Defense of Probability. In: *Proceedings of the International Joint Conference on Artificial Intelligence*.

Chollet, Francois, and Allaire, J. J. 2018. *Deep Learning with R*. Manning Publications.

Codd, Edgar F. 1990. *The Relational Model for Database Management*. Addison-Wesley Longman Publishing.

Cunningham, John P., and Ghahramani, Zoubin. 2015. Linear Dimensionality Reduction: Survey, Insights, and Generalizations. *Journal of Machine Learning Research*, **16**, 2859–2900.

Datta, Biswa N. 2010. *Numerical Linear Algebra and Applications*. SIAM.

Davidson, Anthony C., and Hinkley, David V. 1997. *Bootstrap Methods and Their Application*. Cambridge University Press.

Dean, Jeffrey, Corrado, Greg S., Monga, Rajat, and Chen, et al. 2012. Large Scale Distributed Deep Networks. In: *Advances in Neural Information Processing Systems*.

Deisenroth, Marc P., and Mohamed, Shakir. 2012. Expectation Propagation in Gaussian Process Dynamical Systems. Pages 2618–2626 of: *Advances in Neural Information Processing Systems*.

Deisenroth, Marc P., and Ohlsson, Henrik. 2011. A General Perspective on Gaussian Filtering and Smoothing: Explaining Current and Deriving New Algorithms. In: *Proceedings of the American Control Conference*.

Deisenroth, Marc P., Fox, Dieter, and Rasmussen, Carl E. 2015. Gaussian Processes for Data-Efficient Learning in Robotics and Control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **37**(2), 408–423.

Dempster, Arthur P., Laird, Nan M., and Rubin, Donald B. 1977. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society*, **39**(1), 1–38.

Deng, Li, Seltzer, Michael L., Yu, Dong, Acero, Alex, Mohamed, Abdel-rahman, and Hinton, Geoffrey E. 2010. Binary Coding of Speech Spectrograms Using a Deep Auto-Encoder. In: *Proceedings of Interspeech*.

Devroye, Luc. 1986. *Non-Uniform Random Variate Generation*. Springer.

Donoho, David L., and Grimes, Carrie. 2003. Hessian Eigenmaps: Locally Linear Embedding Techniques for High-Dimensional Data. *Proceedings of the National Academy of Sciences*, **100**(10), 5591–5596.

Dostál, Zdeněk. 2009. *Optimal Quadratic Programming Algorithms: With Applications to Variational Inequalities*. Springer.

Douven, Igor. 2017. Abduction. In: *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University.

Downey, Allen B. 2014. *Think Stats: Exploratory Data Analysis*. 2nd edn. O'Reilly Media.

Dreyfus, Stuart. 1962. The Numerical Solution of Variational Problems. *Journal of Mathematical Analysis and Applications*, **5**(1), 30–45.

Drumm, Volker, and Weil, Wolfgang. 2001. *Lineare Algebra und Analytische Geometrie*. Lecture Notes, Universität Karlsruhe (TH).

Dudley, Richard M. 2002. *Real Analysis and Probability*. Cambridge University Press.

Eaton, Morris L. 2007. *Multivariate Statistics: A Vector Space Approach*. Institute of Mathematical Statistics Lecture Notes.

Eckart, Carl, and Young, Gale. 1936. The Approximation of One Matrix by Another of Lower Rank. *Psychometrika*, **1**(3), 211–218.

Efron, Bradley, and Hastie, Trevor. 2016. *Computer Age Statistical Inference: Algorithms, Evidence and Data Science*. Cambridge University Press.

Efron, Bradley, and Tibshirani, Robert J. 1993. *An Introduction to the Bootstrap*. Chapman and Hall/CRC.

Elliott, Conal. 2009. Beautiful Differentiation. In: *International Conference on Functional Programming*.

Evgeniou, Theodoros, Pontil, Massimiliano, and Poggio, Tomaso. 2000. Statistical Learning Theory: A Primer. *International Journal of Computer Vision*, **38**(1), 9–13.

Fan, Rong-En, Chang, Kai-Wei, Hsieh, Cho-Jui, Wang, Xiang-Rui, and Lin, Chih-Jen. 2008. LIBLINEAR: A Library for Large Linear Classification. *Journal of Machine Learning Research*, **9**, 1871–1874.

Gal, Yarin, van der Wilk, Mark, and Rasmussen, Carl E. 2014. Distributed Variational Inference in Sparse Gaussian Process Regression and Latent Variable Models. In: *Advances in Neural Information Processing Systems*.

Gärtner, Thomas. 2008. *Kernels for Structured Data*. World Scientific.

Gavish, Matan, and Donoho, David L. 2014. The Optimal Hard Threshold for Singular Values is $4\sqrt{3}$. *IEEE Transactions on Information Theory*, **60**(8), 5040–5053.

Gelman, Andrew, Carlin, John B., Stern, Hal S., and Rubin, Donald B. 2004. *Bayesian Data Analysis*. Chapman and Hall/CRC.

Gentle, James E. 2004. *Random Number Generation and Monte Carlo Methods*. Springer.

Ghahramani, Zoubin. 2015. Probabilistic Machine Learning and Artificial Intelligence. *Nature*, **521**, 452–459.

Ghahramani, Zoubin, and Roweis, Sam T. 1999. Learning Nonlinear Dynamical Systems Using an EM Algorithm. In: *Advances in Neural Information Processing Systems*. MIT Press.

Gilks, Walter R., Richardson, Sylvia, and Spiegelhalter, David J. 1996. *Markov Chain Monte Carlo in Practice*. Chapman and Hall/CRC.

Gneiting, Tilmann, and Raftery, Adrian E. 2007. Strictly Proper Scoring Rules, Prediction, and Estimation. *Journal of the American Statistical Association*, **102**(477), 359–378.

Goh, Gabriel. 2017. Why Momentum Really Works. *Distill*.

Gohberg, Israel, Goldberg, Seymour, and Krupnik, Nahum. 2012. *Traces and Determinants of Linear Operators*. Birkhäuser.

Golan, Jonathan S. 2007. *The Linear Algebra a Beginning Graduate Student Ought to Know*. Springer.

Golub, Gene H., and Van Loan, Charles F. 2012. *Matrix Computations*. JHU Press.

Goodfellow, Ian, Bengio, Yoshua, and Courville, Aaron. 2016. *Deep Learning*. MIT Press.

Graepel, Thore, Candela, Joaquin Quiñonero-Candela, Borchert, Thomas, and Herbrich, Ralf. 2010. Web-Scale Bayesian Click-through Rate Prediction for Sponsored Search Advertising in Microsoft's Bing Search Engine. In: *Proceedings of the International Conference on Machine Learning*.

Griewank, Andreas, and Walther, Andrea. 2003. Introduction to Automatic Differentiation. In: *Proceedings in Applied Mathematics and Mechanics*.

Griewank, Andreas, and Walther, Andrea. 2008. *Evaluating Derivatives, Principles and Techniques of Algorithmic Differentiation*. SIAM.

Grimmett, Geoffrey R., and Welsh, Dominic. 2014. *Probability: An Introduction*. Oxford University Press.

Grinstead, Charles M., and Snell, J. Laurie. 1997. *Introduction to Probability*. American Mathematical Society.
Hacking, Ian. 2001. *Probability and Inductive Logic*. Cambridge University Press.
Hall, Peter. 1992. *The Bootstrap and Edgeworth Expansion*. Springer.
Hallin, Marc, Paindaveine, Davy, and Šiman, Miroslav. 2010. Multivariate Quantiles and Multiple-Output Regression Quantiles: From ℓ_1 Optimization to Halfspace Depth. *Annals of Statistics*, **38**, 635–669.
Hasselblatt, Boris, and Katok, Anatole. 2003. *A First Course in Dynamics with a Panorama of Recent Developments*. Cambridge University Press.
Hastie, Trevor, Tibshirani, Robert, and Friedman, Jerome. 2001. *The Elements of Statistical Learning – Data Mining, Inference, and Prediction*. Springer.
Hausman, Karol, Springenberg, Jost T., Wang, Ziyu, Heess, Nicolas, and Riedmiller, Martin. 2018. Learning an Embedding Space for Transferable Robot Skills. In: *Proceedings of the International Conference on Learning Representations*.
Hazan, Elad. 2015. Introduction to Online Convex Optimization. *Foundations and Trends in Optimization*, **2**(3–4), 157–325.
Hensman, James, Fusi, Nicolò, and Lawrence, Neil D. 2013. Gaussian Processes for Big Data. In: *Proceedings of the Conference on Uncertainty in Artificial Intelligence*.
Herbrich, Ralf, Minka, Tom, and Graepel, Thore. 2007. TrueSkill(TM): A Bayesian Skill Rating System. In: *Advances in Neural Information Processing Systems*.
Hiriart-Urruty, Jean-Baptiste, and Lemaréchal, Claude. 2001. *Fundamentals of Convex Analysis*. Springer.
Hoffman, Matthew D., Blei, David M., and Bach, Francis. 2010. Online Learning for Latent Dirichlet Allocation. *Advances in Neural Information Processing Systems*.
Hoffman, Matthew D., Blei, David M., Wang, Chong, and Paisley, John. 2013. Stochastic Variational Inference. *Journal of Machine Learning Research*, **14**(1), 1303–1347.
Hofmann, Thomas, Schölkopf, Bernhard, and Smola, Alexander J. 2008. Kernel Methods in Machine Learning. *Annals of Statistics*, **36**(3), 1171–1220.
Hogben, Leslie. 2013. *Handbook of Linear Algebra*. Chapman and Hall/CRC.
Horn, Roger A., and Johnson, Charles R. 2013. *Matrix Analysis*. Cambridge University Press.
Hotelling, Harold. 1933. Analysis of a Complex of Statistical Variables into Principal Components. *Journal of Educational Psychology*, **24**, 417–441.
Hyvarinen, Aapo, Oja, Erkki, and Karhunen, Juha. 2001. *Independent Component Analysis*. Wiley.
Imbens, Guido W., and Rubin, Donald B. 2015. *Causal Inference for Statistics, Social and Biomedical Sciences*. Cambridge University Press.
Jacod, Jean, and Protter, Philip. 2004. *Probability Essentials*. Springer.
Jaynes, Edwin T. 2003. *Probability Theory: The Logic of Science*. Cambridge University Press.
Jefferys, William H., and Berger, James O. 1992. Ockham's Razor and Bayesian Analysis. *American Scientist*, **80**, 64–72.
Jeffreys, Harold. 1961. *Theory of Probability*. Oxford University Press.
Jimenez Rezende, Danilo, and Mohamed, Shakir. 2015. Variational Inference with Normalizing Flows. In: *Proceedings of the International Conference on Machine Learning*.
Jimenez Rezende, Danilo, Mohamed, Shakir, and Wierstra, Daan. 2014. Stochastic Backpropagation and Approximate Inference in Deep Generative Models. In: *Proceedings of the International Conference on Machine Learning*.
Joachims, Thorsten. 1999. *Advances in Kernel Methods – Support Vector Learning*. MIT Press. Chap. Making Large-Scale SVM Learning Practical, pages 169–184.
Jordan, Michael I., Ghahramani, Zoubin, Jaakkola, Tommi S., and Saul, Lawrence K. 1999. An Introduction to Variational Methods for Graphical Models. *Machine Learning*, **37**, 183–233.

Julier, Simon J., and Uhlmann, Jeffrey K. 1997. A New Extension of the Kalman Filter to Nonlinear Systems. In: *Proceedings of AeroSense Symposium on Aerospace/Defense Sensing, Simulation and Controls*.

Kaiser, Marcus, and Hilgetag, Claus C. 2006. Nonoptimal Component Placement, but Short Processing Paths, Due to Long-Distance Projections in Neural Systems. *PLoS Computational Biology*, **2**(7), e95.

Kalman, Dan. 1996. A Singularly Valuable Decomposition: The SVD of a Matrix. *College Mathematics Journal*, **27**(1), 2–23.

Kalman, Rudolf E. 1960. A New Approach to Linear Filtering and Prediction Problems. *Transactions of the ASME – Journal of Basic Engineering*, **82**(Series D), 35–45.

Kamthe, Sanket, and Deisenroth, Marc P. 2018. Data-Efficient Reinforcement Learning with Probabilistic Model Predictive Control. In: *Proceedings of the International Conference on Artificial Intelligence and Statistics*.

Katz, Victor J. 2004. *A History of Mathematics*. Pearson/Addison-Wesley.

Kelley, Henry J. 1960. Gradient Theory of Optimal Flight Paths. *Ars Journal*, **30**(10), 947–954.

Kimeldorf, George S., and Wahba, Grace. 1970. A Correspondence between Bayesian Estimation on Stochastic Processes and Smoothing by Splines. *Annals of Mathematical Statistics*, **41**(2), 495–502.

Kingma, Diederik P., and Welling, Max. 2014. Auto-Encoding Variational Bayes. In: *Proceedings of the International Conference on Learning Representations*.

Kittler, Josef, and Föglein, Janos. 1984. Contextual Classification of Multispectral Pixel Data. *Image and Vision Computing*, **2**(1), 13–29.

Kolda, Tamara G., and Bader, Brett W. 2009. Tensor Decompositions and Applications. *SIAM Review*, **51**(3), 455–500.

Koller, Daphne, and Friedman, Nir. 2009. *Probabilistic Graphical Models*. MIT Press.

Kong, Linglong, and Mizera, Ivan. 2012. Quantile Tomography: Using Quantiles with Multivariate Data. *Statistica Sinica*, **22**, 1598–1610.

Lang, Serge. 1987. *Linear Algebra*. Springer.

Lawrence, Neil D. 2005. Probabilistic Non-Linear Principal Component Analysis with Gaussian Process Latent Variable Models. *Journal of Machine Learning Research*, **6**(Nov.), 1783–1816.

Leemis, Lawrence M., and McQueston, Jacquelyn T. 2008. Univariate Distribution Relationships. *American Statistician*, **62**(1), 45–53.

Lehmann, Erich L., and Romano, Joseph P. 2005. *Testing Statistical Hypotheses*. Springer.

Lehmann, Erich Leo, and Casella, George. 1998. *Theory of Point Estimation*. Springer.

Liesen, Jörg, and Mehrmann, Volker. 2015. *Linear Algebra*. Springer.

Lin, Hsuan-Tien, Lin, Chih-Jen, and Weng, Ruby C. 2007. A Note on Platt's Probabilistic Outputs for Support Vector Machines. *Machine Learning*, **68**, 267–276.

Ljung, Lennart. 1999. *System Identification: Theory for the User*. Prentice Hall.

Loosli, Gaëlle, Canu, Stéphane, and Ong, Cheng Soon. 2016. Learning SVM in Kreĭn Spaces. *IEEE Transactions of Pattern Analysis and Machine Intelligence*, **38**(6), 1204–1216.

Luenberger, David G. 1969. *Optimization by Vector Space Methods*. Wiley.

MacKay, David J. C. 1992. Bayesian Interpolation. *Neural Computation*, **4**, 415–447.

MacKay, David J. C. 1998. Introduction to Gaussian Processes. Pages 133–165 of: Bishop, C. M. (ed), *Neural Networks and Machine Learning*. Springer.

MacKay, David J. C. 2003. *Information Theory, Inference, and Learning Algorithms*. Cambridge University Press.

Magnus, Jan R., and Neudecker, Heinz. 2007. *Matrix Differential Calculus with Applications in Statistics and Econometrics*. Wiley.

Manton, Jonathan H., and Amblard, Pierre-Olivier. 2015. A Primer on Reproducing Kernel Hilbert Spaces. *Foundations and Trends in Signal Processing*, **8**(1–2), 1–126.
Markovsky, Ivan. 2011. *Low Rank Approximation: Algorithms, Implementation, Applications*. Springer.
Maybeck, Peter S. 1979. *Stochastic Models, Estimation, and Control*. Academic Press.
McCullagh, Peter, and Nelder, John A. 1989. *Generalized Linear Models*. CRC Press.
McEliece, Robert J., MacKay, David J. C., and Cheng, Jung-Fu. 1998. Turbo Decoding as an Instance of Pearl's "Belief Propagation" Algorithm. *IEEE Journal on Selected Areas in Communications*, **16**(2), 140–152.
Mika, Sebastian, Rätsch, Gunnar, Weston, Jason, Schölkopf, Bernhard, and Müller, Klaus-Robert. 1999. Fisher Discriminant Analysis with Kernels. Pages 41–48 of: *Proceedings of the Workshop on Neural Networks for Signal Processing*.
Minka, Thomas P. 2001a. *A Family of Algorithms for Approximate Bayesian Inference*. Ph.D. thesis, Massachusetts Institute of Technology.
Minka, Tom. 2001b. Automatic Choice of Dimensionality of PCA. In: *Advances in Neural Information Processing Systems*.
Mitchell, Tom. 1997. *Machine Learning*. McGraw-Hill.
Mnih, Volodymyr, Kavukcuoglu, Koray, and Silver, David, et al. 2015. Human-Level Control through Deep Reinforcement Learning. *Nature*, **518**, 529–533.
Moonen, Marc, and De Moor, Bart. 1995. *SVD and Signal Processing, III: Algorithms, Architectures and Applications*. Elsevier.
Moustaki, Irini, Knott, Martin, and Bartholomew, David J. 2015. *Latent-Variable Modeling*. American Cancer Society. Pages 1–10.
Müller, Andreas C., and Guido, Sarah. 2016. *Introduction to Machine Learning with Python: A Guide for Data Scientists*. O'Reilly Publishing.
Murphy, Kevin P. 2012. *Machine Learning: A Probabilistic Perspective*. MIT Press.
Neal, Radford M. 1996. *Bayesian Learning for Neural Networks*. Ph.D. thesis, Department of Computer Science, University of Toronto.
Neal, Radford M., and Hinton, Geoffrey E. 1999. A View of the EM Algorithm that Justifies Incremental, Sparse, and Other Variants. Pages 355–368 of: *Learning in Graphical Models*. MIT Press.
Nelsen, Roger. 2006. *An Introduction to Copulas*. Springer.
Nesterov, Yuri. 2018. *Lectures on Convex Optimization*. Springer.
Neumaier, Arnold. 1998. Solving Ill-Conditioned and Singular Linear Systems: A Tutorial on Regularization. *SIAM Review*, **40**, 636–666.
Nocedal, Jorge, and Wright, Stephen J. 2006. *Numerical Optimization*. Springer.
Nowozin, Sebastian, Gehler, Peter V., Jancsary, Jeremy, and Lampert, Christoph H. (eds). 2014. *Advanced Structured Prediction*. MIT Press.
O'Hagan, Anthony. 1991. Bayes-Hermite Quadrature. *Journal of Statistical Planning and Inference*, **29**, 245–260.
Ong, Cheng Soon, Mary, Xavier, Canu, Stéphane, and Smola, Alexander J. 2004. Learning with Non-Positive Kernels. In: *Proceedings of the International Conference on Machine Learning*.
Ormoneit, Dirk, Sidenbladh, Hedvig, Black, Michael J., and Hastie, Trevor. 2001. Learning and Tracking Cyclic Human Motion. In: *Advances in Neural Information Processing Systems*.
Page, Lawrence, Brin, Sergey, Motwani, Rajeev, and Winograd, Terry. 1999. *The PageRank Citation Ranking: Bringing Order to the Web*. Tech. rept. Stanford InfoLab.
Paquet, Ulrich. 2008. *Bayesian Inference for Latent Variable Models*. Ph.D. thesis, University of Cambridge.
Parzen, Emanuel. 1962. On Estimation of a Probability Density Function and Mode. *Annals of Mathematical Statistics*, **33**(3), 1065–1076.

Pearl, Judea. 1988. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann.

Pearl, Judea. 2009. *Causality: Models, Reasoning and Inference*. 2nd edn. Cambridge University Press.

Pearson, Karl. 1895. Contributions to the Mathematical Theory of Evolution. II. Skew Variation in Homogeneous Material. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, **186**, 343–414.

Pearson, Karl. 1901. On Lines and Planes of Closest Fit to Systems of Points in Space. *Philosophical Magazine*, **2**(11), 559–572.

Peters, Jonas, Janzing, Dominik, and Schölkopf, Bernhard. 2017. *Elements of Causal Inference: Foundations and Learning Algorithms*. MIT Press.

Petersen, Kaare B., and Pedersen, Michael S. 2012. *The Matrix Cookbook*. Tech. rept. Technical University of Denmark.

Platt, John C. 2000. Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods. In: *Advances in Large Margin Classifiers*.

Pollard, David. 2002. *A User's Guide to Measure Theoretic Probability*. Cambridge University Press.

Polyak, Roman A. 2016. The Legendre Transformation in Modern Optimization. Pages 437–507 of: Goldengorin, B. (ed), *Optimization and Its Applications in Control and Data Sciences*. Springer.

Press, William H., Teukolsky, Saul A., Vetterling, William T., and Flannery, Brian P. 2007. *Numerical Recipes: The Art of Scientific Computing*. Cambridge University Press.

Proschan, Michael A., and Presnell, Brett. 1998. Expect the Unexpected from Conditional Expectation. *American Statistician*, **52**(3), 248–252.

Raschka, Sebastian, and Mirjalili, Vahid. 2017. *Python Machine Learning: Machine Learning and Deep Learning with Python, scikit-learn, and TensorFlow*. Packt Publishing.

Rasmussen, Carl E., and Ghahramani, Zoubin. 2001. Occam's Razor. In: *Advances in Neural Information Processing Systems*.

Rasmussen, Carl E., and Ghahramani, Zoubin. 2003. Bayesian Monte Carlo. In: *Advances in Neural Information Processing Systems*.

Rasmussen, Carl E., and Williams, Christopher K. I. 2006. *Gaussian Processes for Machine Learning*. MIT Press.

Reid, Mark, and Williamson, Robert C. 2011. Information, Divergence and Risk for Binary Experiments. *Journal of Machine Learning Research*, **12**, 731–817.

Rifkin, Ryan M., and Lippert, Ross A. 2007. Value Regularization and Fenchel Duality. *Journal of Machine Learning Research*, **8**, 441–479.

Rockafellar, Ralph T. 1970. *Convex Analysis*. Princeton University Press.

Rogers, Simon, and Girolami, Mark. 2016. *A First Course in Machine Learning*. Chapman and Hall/CRC.

Rosenbaum, Paul R. 2017. *Observation and Experiment: An Introduction to Causal Inference*. Harvard University Press.

Rosenblatt, Murray. 1956. Remarks on Some Nonparametric Estimates of a Density Function. *Annals of Mathematical Statistics*, **27**(3), 832–837.

Roweis, Sam T. 1998. EM Algorithms for PCA and SPCA. Pages 626–632 of: *Advances in Neural Information Processing Systems*.

Roweis, Sam T., and Ghahramani, Zoubin. 1999. A Unifying Review of Linear Gaussian Models. *Neural Computation*, **11**(2), 305–345.

Roy, Anindya, and Banerjee, Sudipto. 2014. *Linear Algebra and Matrix Analysis for Statistics*. Chapman and Hall/CRC.

Rubinstein, Reuven Y., and Kroese, Dirk P. 2016. *Simulation and the Monte Carlo Method*. Wiley.

Ruffini, Paolo. 1799. *Teoria Generale delle Equazioni, in cui si Dimostra Impossibile la Soluzione Algebraica delle Equazioni Generali di Grado Superiore al Quarto*. Stamperia di S. Tommaso d'Aquino.

Rumelhart, David E., Hinton, Geoffrey E., and Williams, Ronald J. 1986. Learning Representations by Back-Propagating Errors. *Nature*, **323**(6088), 533–536.

Sæmundsson, Steindór, Hofmann, Katja, and Deisenroth, Marc P. 2018. Meta Reinforcement Learning with Latent Variable Gaussian Processes. In: *Proceedings of the Conference on Uncertainty in Artificial Intelligence*.

Saitoh, Saburou. 1988. *Theory of Reproducing Kernels and its Applications*. Longman Scientific and Technical.

Särkkä, Simo. 2013. *Bayesian Filtering and Smoothing*. Cambridge University Press.

Schölkopf, Bernhard, and Smola, Alexander J. 2002. *Learning with Kernels – Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press.

Schölkopf, Bernhard, Smola, Alexander J., and Müller, Klaus-Robert. 1997. Kernel Principal Component Analysis. In: *Proceedings of the International Conference on Artificial Neural Networks*.

Schölkopf, Bernhard, Smola, Alexander J., and Müller, Klaus-Robert. 1998. Nonlinear Component Analysis as a Kernel Eigenvalue Problem. *Neural Computation*, **10**(5), 1299–1319.

Schölkopf, Bernhard, Herbrich, Ralf, and Smola, Alexander J. 2001. A Generalized Representer Theorem. In: *Proceedings of the International Conference on Computational Learning Theory*.

Schwartz, Laurent. 1964. Sous Espaces Hilbertiens d'Espaces Vectoriels Topologiques et Noyaux Associés. *Journal d'Analyse Mathématique*, **13**, 115–256.

Schwarz, Gideon E. 1978. Estimating the Dimension of a Model. *Annals of Statistics*, **6**(2), 461–464.

Shahriari, Bobak, Swersky, Kevin, Wang, Ziyu, Adams, Ryan P., and De Freitas, Nando. 2016. Taking the Human out of the Loop: A Review of Bayesian Optimization. *Proceedings of the IEEE*, **104**(1), 148–175.

Shalev-Shwartz, Shai, and Ben-David, Shai. 2014. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press.

Shawe-Taylor, John, and Cristianini, Nello. 2004. *Kernel Methods for Pattern Analysis*. Cambridge University Press.

Shawe-Taylor, John, and Sun, Shiliang. 2011. A Review of Optimization Methodologies in Support Vector Machines. *Neurocomputing*, **74**(17), 3609–3618.

Shental, Ori, Siegel, Paul H., Wolf, Jack K., Bickson, Danny, and Dolev, Danny. 2008. Gaussian Belief Propagation Solver for Systems of Linear Equations. Pages 1863–1867 of: *Proceedings of the International Symposium on Information Theory*.

Shewchuk, Jonathan R. 1994. *An Introduction to the Conjugate Gradient Method without the Agonizing Pain*.

Shi, Jianbo, and Malik, Jitendra. 2000. Normalized Cuts and Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **22**(8), 888–905.

Shi, Qinfeng, Petterson, James, Dror, Gideon, Langford, John, Smola, Alexander J., and Vishwanathan, S. V. N. 2009. Hash Kernels for Structured Data. *Journal of Machine Learning Research*, 2615–2637.

Shiryayev, Albert N. 1984. *Probability*. Springer.

Shor, Naum Z. 1985. *Minimization Methods for Non-Differentiable Functions*. Springer.

Shotton, Jamie, Winn, John, Rother, Carsten, and Criminisi, Antonio. 2006. TextonBoost: Joint Appearance, Shape and Context Modeling for Multi-Class Object Recognition and Segmentation. In: *Proceedings of the European Conference on Computer Vision*.

Smith, Adrian F. M., and Spiegelhalter, David. 1980. Bayes Factors and Choice Criteria for Linear Models. *Journal of the Royal Statistical Society B*, **42**(2), 213–220.

Snoek, Jasper, Larochelle, Hugo, and Adams, Ryan P. 2012. Practical Bayesian Optimization of Machine Learning Algorithms. In: *Advances in Neural Information Processing Systems*.

Spearman, Charles. 1904. "General Intelligence," Objectively Determined and Measured. *American Journal of Psychology*, **15**(2), 201–292.

Sriperumbudur, Bharath K., Gretton, Arthur, Fukumizu, Kenji, Schölkopf, Bernhard, and Lanckriet, Gert R. G. 2010. Hilbert Space Embeddings and Metrics on Probability Measures. *Journal of Machine Learning Research*, **11**, 1517–1561.

Steinwart, Ingo. 2007. How to Compare Different Loss Functions and Their Risks. *Constructive Approximation*, **26**, 225–287.

Steinwart, Ingo, and Christmann, Andreas. 2008. *Support Vector Machines*. Springer.

Stoer, Josef, and Burlirsch, Roland. 2002. *Introduction to Numerical Analysis*. Springer.

Strang, Gilbert. 1993. The Fundamental Theorem of Linear Algebra. *The American Mathematical Monthly*, **100**(9), 848–855.

Strang, Gilbert. 2003. *Introduction to Linear Algebra*. Wellesley-Cambridge Press.

Stray, Jonathan. 2016. *The Curious Journalist's Guide to Data*. Tow Center for Digital Journalism at Columbia's Graduate School of Journalism.

Strogatz, Steven. 2014. Writing about Math for the Perplexed and the Traumatized. *Notices of the American Mathematical Society*, **61**(3), 286–291.

Sucar, Luis E., and Gillies, Duncan F. 1994. Probabilistic Reasoning in High-Level Vision. *Image and Vision Computing*, **12**(1), 42–60.

Szeliski, Richard, Zabih, Ramin, and Scharstein, Daniel, et al. 2008. A Comparative Study of Energy Minimization Methods for Markov Random Fields with Smoothness-Based Priors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **30**(6), 1068–1080.

Tandra, Haryono. 2014. The Relationship between the Change of Variable Theorem and the Fundamental Theorem of Calculus for the Lebesgue Integral. *Teaching of Mathematics*, **17**(2), 76–83.

Tenenbaum, Joshua B., De Silva, Vin, and Langford, John C. 2000. A Global Geometric Framework for Nonlinear Dimensionality Reduction. *Science*, **290**(5500), 2319–2323.

Tibshirani, Robert. 1996. Regression Selection and Shrinkage via the Lasso. *Journal of the Royal Statistical Society B*, **58**(1), 267–288.

Tipping, Michael E., and Bishop, Christopher M. 1999. Probabilistic Principal Component Analysis. *Journal of the Royal Statistical Society: Series B*, **61**(3), 611–622.

Titsias, Michalis K., and Lawrence, Neil D. 2010. Bayesian Gaussian Process Latent Variable Model. In: *Proceedings of the International Conference on Artificial Intelligence and Statistics*.

Toussaint, Marc. 2012. *Some Notes on Gradient Descent*. https://ipvs.informatik.uni-stuttgart.de/mlr/marc/notes/gradientDescent.pdf.

Trefethen, Lloyd N., and Bau III, David. 1997. *Numerical Linear Algebra*. SIAM.

Tucker, Ledyard R. 1966. Some Mathematical Notes on Three-Mode Factor Analysis. *Psychometrika*, **31**(3), 279–311.

Vapnik, Vladimir N. 1998. *Statistical Learning Theory*. Wiley.

Vapnik, Vladimir N. 1999. An Overview of Statistical Learning Theory. *IEEE Transactions on Neural Networks*, **10**(5), 988–999.

Vapnik, Vladimir N. 2000. *The Nature of Statistical Learning Theory*. Springer.

Vishwanathan, S. V. N., Schraudolph, Nicol N., Kondor, Risi, and Borgwardt, Karsten M. 2010. Graph Kernels. *Journal of Machine Learning Research*, **11**, 1201–1242.

von Luxburg, Ulrike, and Schölkopf, Bernhard. 2011. Statistical Learning Theory: Models, Concepts, and Results. Pages 651–706 of: D. M. Gabbay, S. Hartmann, J. Woods (ed), *Handbook of the History of Logic*, vol. 10. Elsevier.

Wahba, Grace. 1990. *Spline Models for Observational Data*. Society for Industrial and Applied Mathematics.

Walpole, Ronald E., Myers, Raymond H., Myers, Sharon L., and Ye, Keying. 2011. *Probability and Statistics for Engineers and Scientists*. Prentice Hall.

Wasserman, Larry. 2004. *All of Statistics*. Springer.

Wasserman, Larry. 2007. *All of Nonparametric Statistics*. Springer.

Whittle, Peter. 2000. *Probability via Expectation*. Springer.

Wickham, Hadley. 2014. Tidy Data. *Journal of Statistical Software*, **59**, 1–23.

Williams, Christopher K. I. 1997. Computing with Infinite Networks. In: *Advances in Neural Information Processing Systems*.

Yu, Yaoliang, Cheng, Hao, Schuurmans, Dale, and Szepesvári, Csaba. 2013. Characterizing the Representer Theorem. In: *Proceedings of the International Conference on Machine Learning*.

Zadrozny, Bianca, and Elkan, Charles. 2001. Obtaining Calibrated Probability Estimates from Decision Trees and Naive Bayesian Classifiers. In: *Proceedings of the International Conference on Machine Learning*.

Zhang, Haizhang, Xu, Yuesheng, and Zhang, Jun. 2009. Reproducing Kernel Banach Spaces for Machine Learning. *Journal of Machine Learning Research*, **10**, 2741–2775.

Zia, Royce K. P., Redish, Edward F., and McKay, Susan R. 2009. Making Sense of the Legendre Transform. *American Journal of Physics*, **77**(614), 614–622.