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

i i d samples from an "unknown" distribution

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.

$x_n \in \mathbb{R}^D, n = 1..N$

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

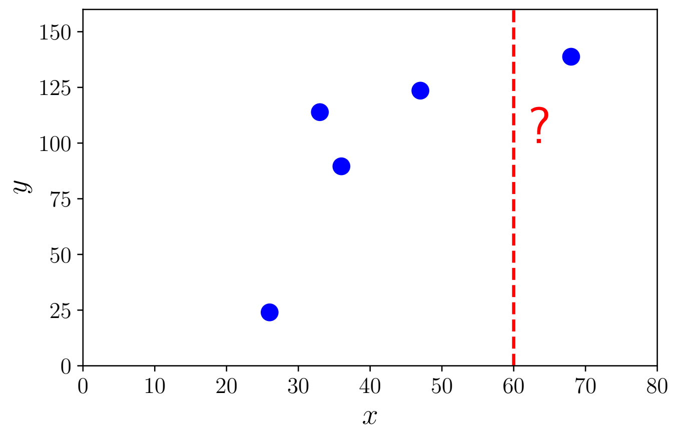

Figure 8.1 Toy data for linear regression. Training data in (x_n, y_n) pairs from the rightmost two columns of Table 8.2. We are interested in the salary of a person aged sixty ($x = 60$) illustrated as a vertical dashed red line, which is not part of the training data.

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

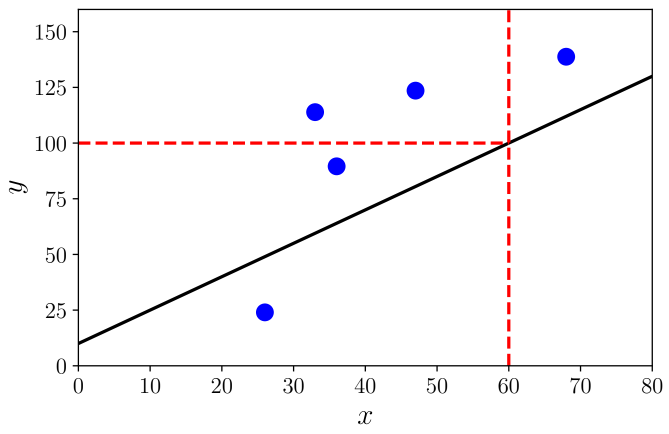

Figure 8.2 Example function (black solid diagonal line) and its prediction at $x = 60$, i.e., $f(60) = 100$.

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

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

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

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

$\theta \in \mathbb{R}^D$
parameters

for $\underline{\text{unknown } \boldsymbol{\theta} \text{ and } \theta_0}$. 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)

1) Data $\longrightarrow$ Tabular , $x_n \in \mathbb{R}^D$, $n = 1..N$

2) Model

3) Learning

2) We define a model depending on a set of parameters

3) Learn the parameters of the model from the data

Aim : to find a GOOD MODEL REPRESENTING THE DATA

What is a good model?

The model generalise well on unseen data

Well $\rightarrow$ We need to define (the) metrics and the metrics some are fine.

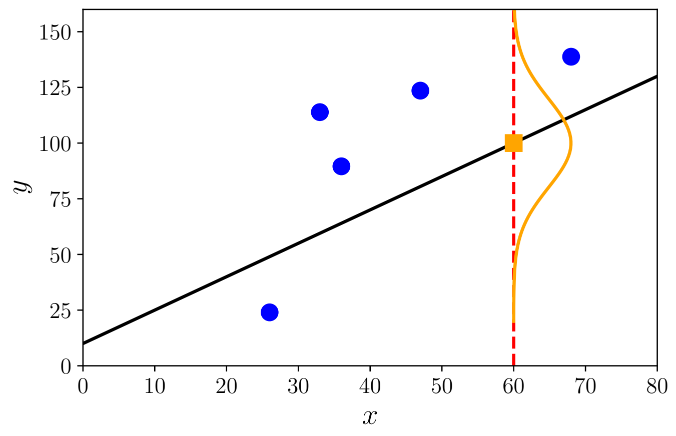

Figure 8.3 Example function (black solid diagonal line) and its predictive uncertainty at $x = 60$ (drawn as a Gaussian).

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

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.

loss function

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

One assumption that is commonly made in machine learning is that the set of examples $(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_N, y_N)$ 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* $\{(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_N, y_N)\}$, we introduce the notation of an example matrix $\boldsymbol{X} := [\boldsymbol{x}_1, \dots, \boldsymbol{x}_N]^\top \in \mathbb{R}^{N \times D}$ and a label vector $\boldsymbol{y} := [y_1, \dots, y_N]^\top \in \mathbb{R}^N$. Using this matrix notation the average loss is given by

independent and identically distributed

training set

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

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

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).

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.

overfitting

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

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

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

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

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

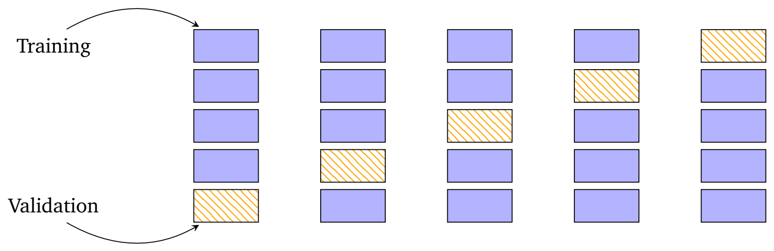

Figure 8.4 K-fold cross-validation. The dataset is divided into $K = 5$ chunks, $K - 1$ of which serve as the training set (blue) and one as the validation set (orange hatch).

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 non-deterministic. 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

Probabilistic model

The data are represented by samples from a multivariate random variable $\underline{x} \sim \boxed{p(\underline{x} | \underline{\theta})}$ with distribution probability depending on parameters θ.

The model is $p(\underline{x}|\underline{\theta})$ and we must learn (compute) $\underline{\theta}$ well representing the data $\underline{x}$.

Maximum Likelihood Estimation (MLE)

1) Define the likelihood function

$$P(\theta) = \prod_{n=1}^{N} p(x_n | \underline{\theta})$$

2) compute

$$\theta^* = \underset{\underline{\theta} \in R^D}{\operatorname{argmax}} \ P(\underline{\theta})$$

for computational reasons

$$\boxed{\theta^* = \underset{\theta \in R^D}{\operatorname{argmin}} \ \underbrace{-\log P(\theta)}_{\mathcal{L}(\theta)}}$$

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

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 $p(\boldsymbol{x} \,|\, \boldsymbol{\theta})$ is modeling for a fixed value of $\boldsymbol{\theta}$. It is a distribution that models the uncertainty of the data. In other words, once we have chosen the type of function we want as a predictor, the likelihood provides the probability of observing data $\boldsymbol{x}$.

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, $\varepsilon_n \sim \mathcal{N}(0, \sigma^2)$. 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 $(\boldsymbol{x}_n, y_n)$,

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

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 of 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

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

where $p(y_n\,|\,\boldsymbol{x}_n,\boldsymbol{\theta})$ 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 $\log(ab) = \log(a) + \log(b)$

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

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. ◇

ASSUME

$$p(y_n\,|\,x_n,\theta) = \mathcal{N}(y_n\,|\,x_n^\top\theta,\sigma^2)$$

Example 8.5
Continuing on our example of Gaussian likelihoods (8.15), the negative log-likelihood can be rewritten as

Known

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

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

$$\underbrace{y_n} - \theta^T x_n$$

$$p(y|x,\theta) = \prod_{n=1}^{N} p(y_n|x_n,\theta)$$

$$-\log p(y|x,\theta) = \log \prod_{n=1}^{N} p(y_n|x_n,\theta)$$

$$= \boxed{-\sum_{n=1}^{N} \log p(y_n|x_n,\theta)} = \mathcal{L}(\theta)$$

$$\log A\cdot B = \log A + \log B$$

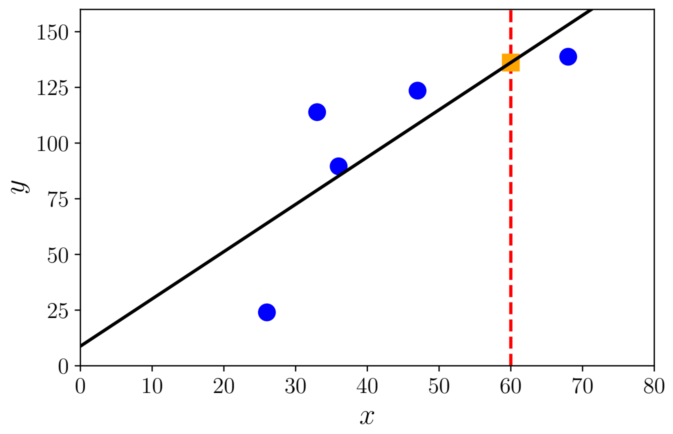

Figure 8.5 For the given data, the maximum likelihood estimate of the parameters results in the black diagonal line. The orange square shows the value of the maximum likelihood prediction at $x = 60$.

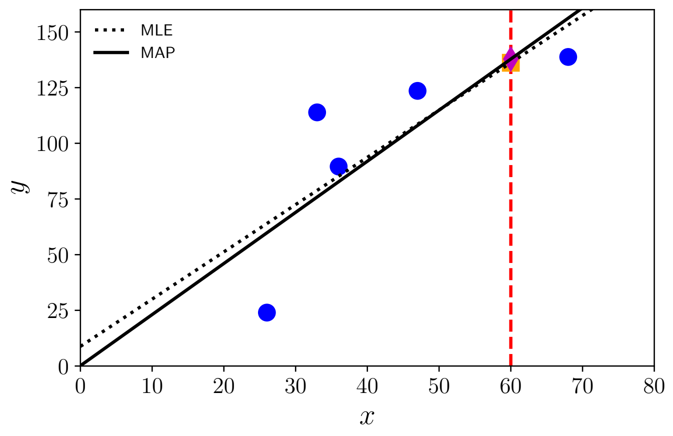

Figure 8.6 Comparing the predictions with the maximum likelihood estimate and the MAP estimate at $x = 60$. The prior biases the slope to be less steep and the intercept to be closer to zero. In this example, the bias that moves the intercept closer to zero actually increases the slope.

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., $p(\boldsymbol{\theta}) = \mathcal{N}\big(\boldsymbol{0}, \boldsymbol{\Sigma}\big)$, 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

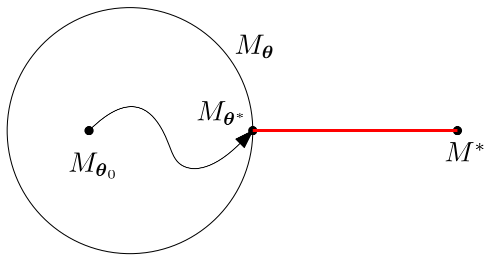
$M_{\boldsymbol{\theta}}$
$M_{\boldsymbol{\theta}^*}$
$M_{\boldsymbol{\theta}_0}$
M^*

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*.

$\diamondsuit$

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

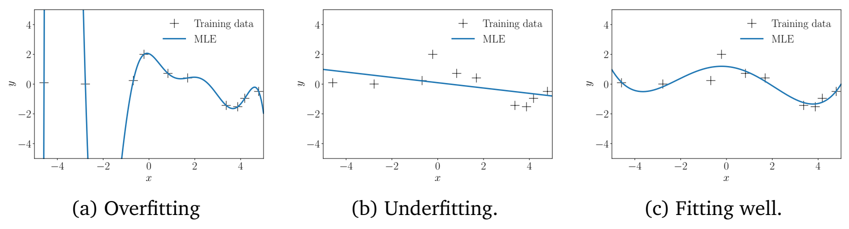

Figure 8.8 Fitting (by maximum likelihood) of different model classes to a regression dataset.

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. STOP HERE

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