4667

Part II

Central Machine Learning Problems

4672

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, 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.

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?". That is, we are interested to find models that perform well on future data. 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.

model

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 between different models. This chapter also covers a few necessary bits and pieces of mathematical and statistical language that are commonly

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. We will introduce the framework of empirical risk minimization in Section 8.1, the principle of maximum likelihood in Section 8.2, and the idea of probabilistic models in Section 8.3. We briefly outline a graphical language for specifying probabilistic models in Section 8.4 and finally discuss model selection in Section 8.5. The rest of this section expands upon the three main components of machine learning: data, models and learning.

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, which in recent years 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, 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

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.

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 infomative for a machine learning task, and 2. we may wish to anonymize the data to help protect the privacy of the employees.

features
attributes
covariates

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 n^{th} 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 (for example in Chapter 10) the it is more convenient to represent examples as column vectors.

example
data point
feature

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 rightmost two 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 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 ex-

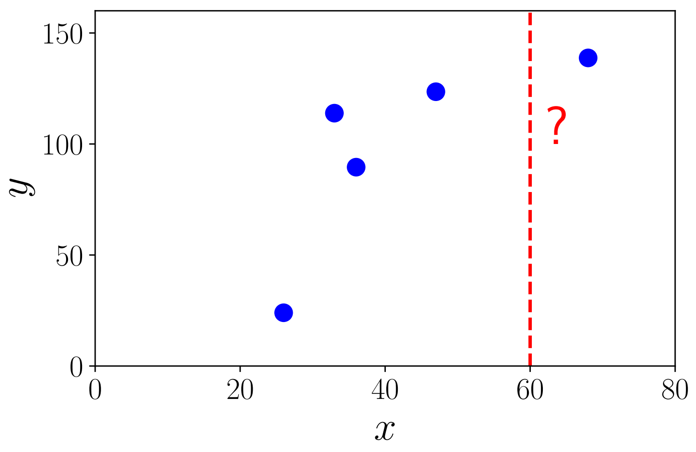


Figure 8.1 Toy data for linear regression. Training data in (x_n, y_n) pairs from the rightmost two columns of Table 8.2. We are interested in the salary of a person aged 60 ($x = 60$) illustrated as a vertical dashed red line, which is not part of the training data.

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

feature map

kernel

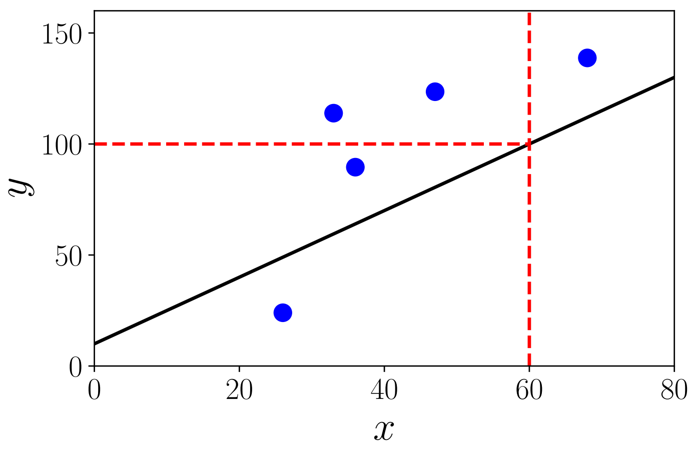


Figure 8.2 Example function (black solid diagonal line) and its prediction at $x = 60$, i.e., $f(60) = 100$.

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

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

This restriction means that the contents of Chapter 2 and 3 suffice for precisely stating the notion of a predictor for the non-probabilistic (in contrast to the probabilistic view described next) 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.

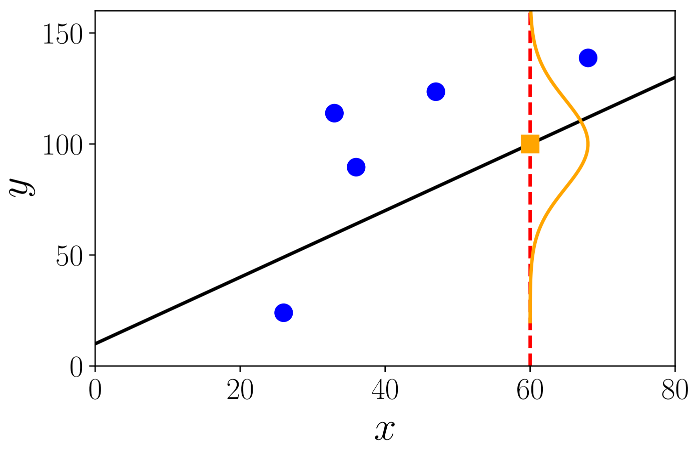


Figure 8.3 Example function (black solid diagonal line) and its predictive uncertainty at $x = 60$ (drawn as a Gaussian).

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 8.3, and introduce a graphical language for describing probabilistic models in a compact way in Section 8.4.

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 8.3) 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 8.1. 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 8.2). We can additionally model the uncertainty of parameters using a probabilisitic model, which we will look at in more detail in Section 8.3.

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 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 behaviour of our predictor on future unseen data using *cross validation* (Section 8.1.4). 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 8.1.3) or by adding a prior (Section 8.2.2). In philosophy, this is considered to be neither induction nor deduction, but

cross validation

abduction

is called *abduction*. According to the Stanford Encyclopedia of Philosophy, abduction is the process of inference to the best explanation (Douven, 2017).

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 between different models is called *model selection*, which we describe in Section 8.5. For non-probabilistic models, model selection is often done using *nested cross validation*, which is described in Section 8.5.1. 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 8.1), the principle of maximum likelihood (Section 9.2), and probabilistic modeling (Section 8.3).

8.1 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 8.2. 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 8.1.1 What is the set of functions we allow the predictor to take?

Section 8.1.2 How do we measure how well the predictor performs on the training data?

Section 8.1.3 How do we construct predictors from only training data that performs well on unseen test data?

Section 8.1.4 What is the procedure for searching over the space of models?

8.1.1 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 $f(\cdot, \boldsymbol{\theta}) : \mathbb{R}^D \to \mathbb{R}$, parameterized by $\boldsymbol{\theta}$. We hope to be able to find a good parameter $\boldsymbol{\theta}^*$ such that we fit the data well

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

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. This simplifies the definition of the hypothesis class and the loss function. ♢

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 a 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

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

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

8.1.2 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 two values 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 word error is often used to mean loss.

One assumption that is commonly made in machine learning is that the set of examples $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_N, y_N)$ are *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

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 have 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 the 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. ♢

8.1.3 Regularization to Reduce Overfitting

This section describes an addition to empirical risk minimization that allows it to generalize well (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

test set

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

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 subscript $_{\text{train}}$ and $_{\text{test}}$ to denote the training and test set respectively. We will revisit this idea of using a finite dataset to evaluate expected risk in Section 8.1.4.

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.

regularization

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.

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 $\boldsymbol{\theta}$:

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

regularizer

regularization parameter

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).

penalty term

The regularization term is sometimes called the *penalty term*, what 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.

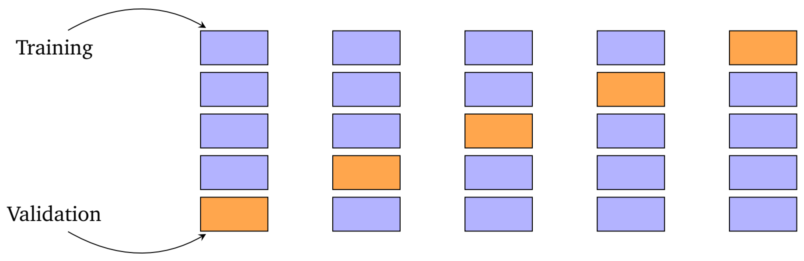


Figure 8.4 K-fold cross validation. The dataset is divided into $K = 5$ chunks, $K - 1$ of which serve as the training set (blue) and one as the validation set (orange).

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 8.2.2. We will see in Chapter 12 that the idea of the regularizer is equivalent to the idea of a large margin.

8.1.4 Cross Validation to Assess the Generalization Performance

We mentioned in the previous section that we measure 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 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 above). 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 training set 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 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.

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 8.2 we will see the effect of using a probability distribution to replace the idea of loss functions and regularization.

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* (Hastie et al., 2001; von Luxburg and Schölkopf, 2011; Vapnik, 1999; Evgeniou et al., 2000). 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 idea 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 tradeoff and feature selection (Bühlmann and Geer, 2011).

An alternative to cross validation is bootstrap and jackknife (Efron and Tibshirani, 1993; Davidson and Hinkley, 1997; Hall, 1992).

Thinking about empirical risk minimization (Section 8.1) as “probability free” is incorrect. There is an underlying unknown probability distribution $p(\boldsymbol{x}, y)$ that governs the data generation, but the approach of empirical risk minimization is agnostic to that choice of distribution. The design of the loss function (Section 8.1.2) does, however, implicitly select for particular conditional probability distributions. 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.

8.2 Parameter Estimation

In Section 8.1 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 8.2.1 we introduce the likelihood, which is analogous to the concept of loss functions (Section 8.1.2) in empirical risk minimization. The concept of priors (Section 8.2.2) is analogous to the concept of regularization (Section 8.1.3).

8.2.1 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 $p(\boldsymbol{x} \mid \boldsymbol{\theta})$ parameterized 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. 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 $(\boldsymbol{x}_1, y_1), \dots, (\boldsymbol{x}_N, y_N)$ 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 expression above 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

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

where $p(y_n \,|\, \boldsymbol{x}_n, \boldsymbol{\theta})$ is a particular distribution (which was Gaussian in the example above (8.15)). The expression "identically distributed" means that each term in the product above 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, and 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. ♢

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

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

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 problem corresponding to maximum likelihood estimation has a closed-form solution. We will see more details on this in Chapter 9. Maximum likelihood estimation may suffer from overfitting, analogous to unregularized empirical risk minimization (Section 9.2.3). 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.

8.2.2 Maximum A Posteriori Estimation

If we have prior knowledge about the distribution of the parameters $\boldsymbol{\theta}$ of our distribution 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 $p(\boldsymbol{\theta} \,|\, \boldsymbol{x})$ (the more specific knowledge) on the parameters $\boldsymbol{\theta}$ from general *prior* statements (prior distribution) $p(\boldsymbol{\theta})$ and

posterior
prior

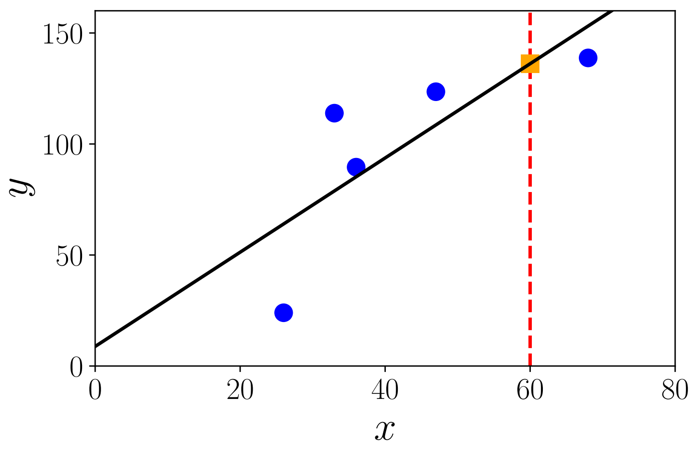


Figure 8.5 For the given data, the maximum likelihood estimate of the parameters results in the black diagonal line. The orange square shows the value of the maximum likelihood prediction at $x = 60$.

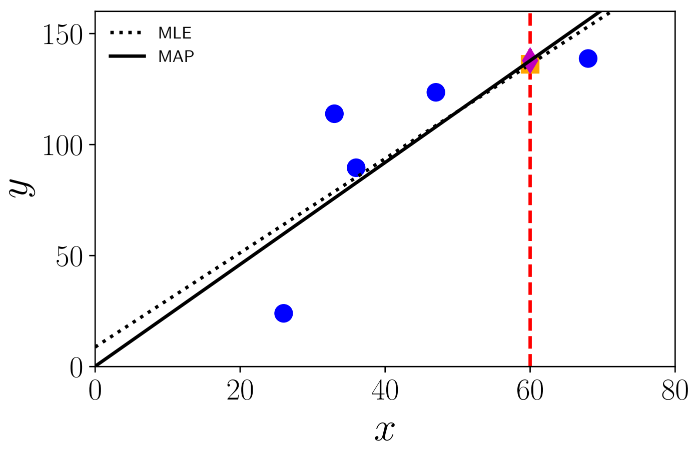


Figure 8.6 Comparing the Maximum Likelihood estimate and the Maximum A Posteriori estimate and their predictions at $x = 60$. The prior biases the slope to be less steep and the intercept to be closer to zero. In this example, the bias that moves the intercept closer to zero actually increases the slope.

likelihood

the function $p(\boldsymbol{x} \mid \boldsymbol{\theta})$ that links the parameters $\boldsymbol{\theta}$ and the observed data $\boldsymbol{x}$ (called the *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 likelihood, and the distribution $p(\boldsymbol{x})$ does not depend on θ. Therefore, we can ignore the value of the denominator and obtain

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

maximum a posteriori estimation

The proportion relation above 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).

An illustration of the effect of adding a zero mean Gaussian prior is shown in Figure 8.6.

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.

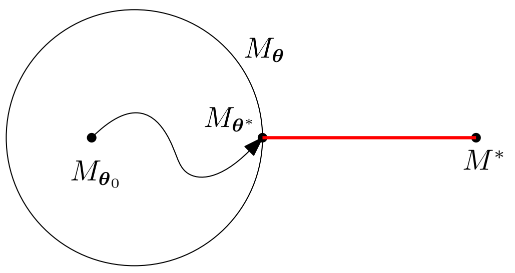
The idea of including prior knowledge about where good parameters lie is widespread in machine learning. An alternative view which we saw in Section 8.1.3 is the idea of regularization, which introduces an additional term that biases the resulting parameters to be close to the origin. Maximum a posteriori 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 $\boldsymbol{\theta}_{\mathrm{ML}}$ possesses the following properties (Lehmann and Casella, 1998; Efron and Hastie, 2016):

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

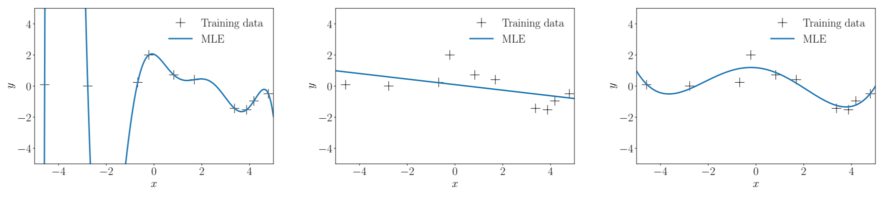
$\diamondsuit$

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 8.3 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.



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



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

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

8.2.3 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 8.2.1) and maximum a posteriori estimation (Section 8.2.2) 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 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 this setting. After the optimization, i.e., when we obtain the best possible parameters $\boldsymbol{\theta}$, we distinguish three different cases: (i) overfitting, (ii) underfitting, (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 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, which 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 8.2.1). We will discuss overfitting in regression more in Section 9.2.2.

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.

underfitting

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 as discussed in Section 8.1.3.

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).

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 8.3. 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.

8.3 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.

8.3.1 Probabilistic Models

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.

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

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 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 8.5) 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.

8.3.2 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 8.2.1, 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 on page 189, 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 here 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 8.2 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 (Kamthe and Deisenroth, 2018; Deisenroth et al., 2015).

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 posterior, 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; Murphy, 2012; Barber, 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 separation above. $\diamondsuit$

8.3.3 Latent Variable Models

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 (Ljung, 1999; Ghahramani and Roweis, 1999) for time-series modeling, and meta learning and task generalization (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.

latent variables

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

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

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 8.2 and 8.3.2. 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 8.2 and 8.3.2, 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

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

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 8.2.2. Moreover, with the likelihood 8.25 Bayesian inference (Section 8.3.2) 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 (Paquet, 2008; Bishop, 2006; Murphy, 2012; Moustaki et al., 2015).

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

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

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 (Murphy, 2012; Bishop, 2006). A quantity that is easier to compute is the posterior distribution on the latent variables, but conditioned on the model parameters, i.e.,

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

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 8.4, 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.

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. The field of *probabilistic programming* is rapidly changing, but several examples of promising languages at the present are:

probabilistic programming

Stan `http://mc-stan.org/`
Edward `http://edwardlib.org/`
PyMC3 `https://docs.pymc.io/`
Pyro `http://pyro.ai/`
Tensorflow Probability `https://github.com/tensorflow/probability`
Infer.NET `http://infernet.azurewebsites.net/`
MXFusion `https://github.com/amzn/MXFusion`

8.4 Directed Graphical Models

This section may be skipped at first reading.

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 8.3, we identified the joint distribution of a probabilistic model as the key quantity of interest because it

directed graphical model

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 models are also known as Bayesian networks.

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.

graphical model

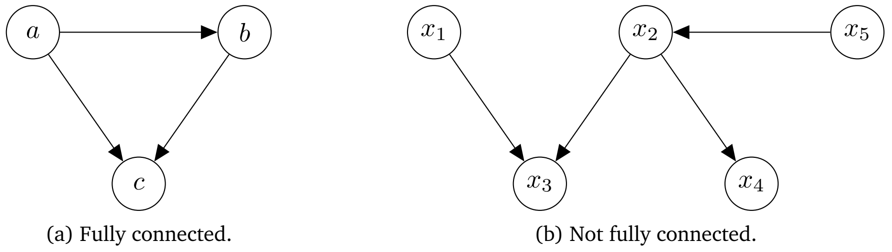
Remark. Not every distribution can be represented in a particular choice of graphical model. A discussion of this can be found in (Bishop, 2006). $\diamondsuit$

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 kind 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.

8.4.1 Graph Semantics

Figure 8.9 Examples of directed graphical models.



$$\begin{tikzcd} a \arrow[rr] \arrow[dr] & & b \arrow[dl] \\ & c & \end{tikzcd} \qquad \begin{tikzcd} x_1 \arrow[dr] & & x_2 \arrow[dl] \arrow[dr] & & x_5 \arrow[ll] \\ & x_3 & & x_4 & \end{tikzcd}$$

directed graphical model/Bayesian network

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

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.

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 on

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 go 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 $p(x_1, \dots, x_5)$ 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

$$p(x_1, x_2, x_3, x_4, x_5) = p(x_1)p(x_5)p(x_2 \mid x_5)p(x_3 \mid x_1, x_2)p(x_4 \mid x_2)\,. \tag{8.30}$$

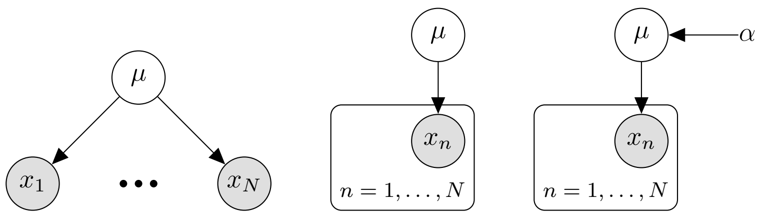


Figure 8.10 Graphical models for a repeated Bernoulli experiment.

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

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

$$p(\boldsymbol{x}) = \prod_{k=1}^{K} p(x_k \,|\, \text{Pa}_k) \tag{8.31}$$

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

$$p(x \,|\, \mu) = \text{Ber}(\mu) \,. \tag{8.32}$$

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

$$p(x_1, \dots, x_N \,|\, \mu) = \prod_{n=1}^{N} p(x_n \,|\, \mu) \,. \tag{8.33}$$

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, $n = 1, \dots, N$ 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 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 hyper-prior on μ. A *hyper-prior* is a second layer of prior distributions on the parameters of the first layer of priors. Figure 8.10(c) places a Beta(α) prior on the latent variable μ. If we treat α as a constant (deterministic parameter), i.e., not a random variable, we omit the circle around it.

plate

hyper-prior

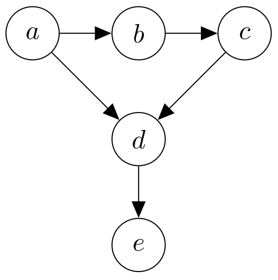


Figure 8.11 D-separation example.

8.4.2 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 non-intersecting 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

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

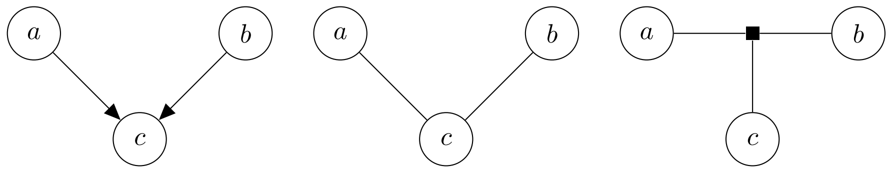
is implied by a given directed acyclic graph. To do so, we consider all possible trails (paths that ignore the direction of the arrows) from any node in $\mathcal{A}$ to any nodes in $\mathcal{B}$. Any such path is said to be blocked if it includes any node such that either

- the arrows on the path meet either head to tail or tail to tail at the node, and the node is in the set $\mathcal{C}$, or
- 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 $\mathcal{A} \perp\!\!\!\perp \mathcal{B} \,|\, \mathcal{C}$.

Example 8.9 (Conditional Independence)
Consider the graphical model in Figure 8.11. By visual inspection we see that

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



(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.

Directed graphical models allow a compact representation of probabilisitic models, and we will see examples of directed graphical models in Chapter 9, 10 and 11. The representation along with the concept of conditional independence, allows us to factorize the respective probabilisitic 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 8.5.

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 Koller and Friedman (2009).

There are three main types of probabilistic graphical models:

directed graphical models
undirected graphical models
factor graphs

- *Directed graphical models* (Bayesian networks), see Figure 8.13(a)
- *Undirected graphical models* (Markov random fields), see Figure 8.13(b)
- *Factor graphs*, see Figure 8.13(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 de-noising, image restoration (Sucar and Gillies, 1994; Shotton et al., 2006; Szeliski et al., 2008; Kittler and Föglein, 1984)) 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 which 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 allow machine learning models to tackle predictions that are structured, for example sequences, trees

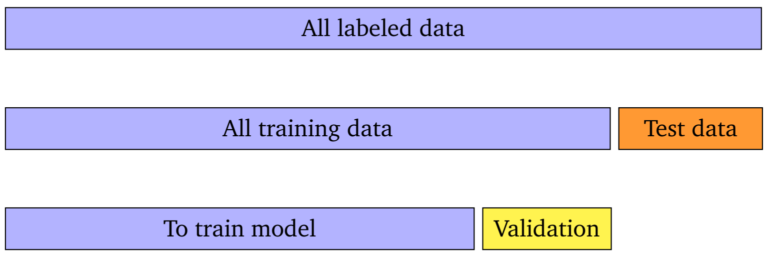


Figure 8.13 Nested cross validation. We perform two levels of K-fold cross validation.

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 its applications to causal inference (Rosenbaum, 2017; Pearl, 2009; Imbens and Rubin, 2015; Peters et al., 2017).

8.5 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.

Note that 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 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 8.2, 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.

8.5.1 Nested Cross Validation

We have already seen an approach (cross validation in Section 8.1.4) that can be used for model selection. Recall that cross validation provides an

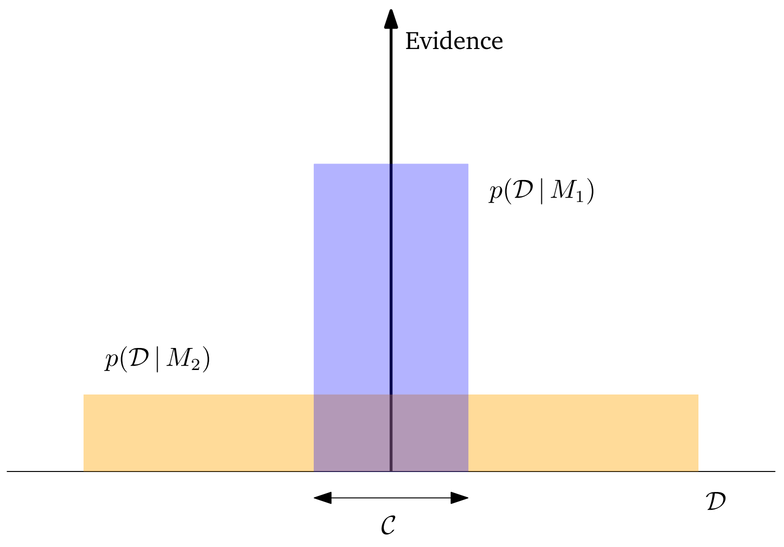


Figure 8.14 Bayesian inference embodies Occam's razor. The horizontal axis describes the space of all possible datasets $\mathcal{D}$. The evidence (vertical axis) evaluates how well a model predicts available data. Since $p(\mathcal{D} \mid M_i)$ needs to integrate to 1, we should choose the model with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

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*.

nested cross validation

test set

validation set

8.5.2 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; MacKay, 1992; Jefferys and Berger, 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 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.

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

evidence

Above, 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.

Let us consider a finite number of models $M = \{M_1, \dots, M_K\}$, 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

Bayesian model selection

generative process

$$M_k \sim p(M) \tag{8.39}$$

$$\boldsymbol{\theta}_k \sim p(\boldsymbol{\theta} \mid M_k) \tag{8.40}$$

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

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

$$p(M_k \mid \mathcal{D}) \propto p(M_k) p(\mathcal{D} \mid M_k) \,. \tag{8.42}$$

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

$$p(\mathcal{D} \mid M_k) = \int p(\mathcal{D} \mid \boldsymbol{\theta}_k) p(\boldsymbol{\theta}_k \mid M_k) d\boldsymbol{\theta}_k \,, \tag{8.43}$$

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 (8.43) is referred to as the *model evidence* or *marginal likelihood*. From the posterior in (8.42), we determine the MAP estimate

model evidence

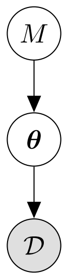
marginal likelihood

$$M^* = \arg\max_{M_k} p(M_k \mid \mathcal{D}) \,. \tag{8.44}$$

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 (8.43).

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). $\diamondsuit$

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 $\mathcal{D}$.



8.5.3 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

$$\underbrace{\frac{p(M_1 \mid \mathcal{D})}{p(M_2 \mid \mathcal{D})}}_{\text{posterior odds}} = \frac{\frac{p(\mathcal{D} \mid M_1)p(M_1)}{p(\mathcal{D})}}{\frac{p(\mathcal{D} \mid M_2)p(M_2)}{p(\mathcal{D})}} = \underbrace{\frac{p(M_1)}{p(M_2)}}_{\text{prior odds}} \underbrace{\frac{p(\mathcal{D} \mid M_1)}{p(\mathcal{D} \mid M_2)}}_{\text{Bayes factor}} . \tag{8.45}$$

posterior odds

prior odds

Bayes factor

The ration of the posteriors is also called the *posterior odds*. The first fraction on the right-hand-side of (8.45), 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.

Jeffreys-Lindley paradox

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. $\diamondsuit$

If we choose a uniform prior over models, the prior odds term in (8.45) is 1, i.e., the posterior odds is the ratio of the marginal likelihoods (Bayes factor)

$$\frac{p(\mathcal{D} \mid M_1)}{p(\mathcal{D} \mid M_2)} . \tag{8.46}$$

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 (8.45) and posterior distributions over models (8.42).

Unfortunately, computing the marginal likelihood requires us to solve an integral (8.43). 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. $\diamondsuit$

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 flavours of learning in Section 8.1, Section 8.2 and Section 8.3 are applied to the four pillars of machine learning (regression, dimensionality reduction, density estimation and classification).

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 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

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 non-parametric models with many parameters, e.g., Gaussian processes.

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

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

$$\log p(\boldsymbol{x} \mid \boldsymbol{\theta}) - M \tag{8.47}$$

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

$$\log p(\boldsymbol{x}) = \log \int p(\boldsymbol{x} \mid \boldsymbol{\theta}) p(\boldsymbol{\theta}) d\boldsymbol{\theta} \approx \log p(\boldsymbol{x} \mid \boldsymbol{\theta}) - \frac{1}{2} M \log N \tag{8.48}$$

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.