

## Chapter 2

# Introduction to supervised learning

### Chapter summary


- Decision theory (loss, risk, optimal predictors): what is the optimal prediction and performance given infinite data and infinite computational resources?
- Statistical learning theory: when is an algorithm “consistent”?
- No free lunch theorems: learning is impossible without making assumptions.

$\mathcal{X}$	input space
$\mathcal{Y}$	output space
$p$	joint distribution on $\mathcal{X} \times \mathcal{Y}$
$(x_1, y_1, \dots, x_n, y_n)$	training data
$f : \mathcal{X} \rightarrow \mathcal{Y}$	prediction function
$\ell(y, z)$	loss function between output $y$ and prediction $z$
$\mathcal{R}(f) = \mathbb{E}[\ell(y, f(x))]$	expected risk of prediction function $f$
$\hat{\mathcal{R}}(f) = \frac{1}{n} \sum_{i=1}^n \ell(y_i, f(x_i))$	empirical risk of prediction function $f$
$f^*(x') = \arg \min_{z \in \mathcal{Y}} \mathbb{E}[\ell(y, z)   x = x']$	Bayes prediction at $x'$
$\mathcal{R}^* = \mathbb{E}_{x' \sim p} \inf_{z \in \mathcal{Y}} \mathbb{E}[\ell(y, z)   x = x']$	Bayes risk

Table 2.1: Summary of notions and notations presented in this chapter and used throughout the book.

## 2.1 From training data to predictions

**Main goal.** Given some observations  $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ ,  $i = 1, \dots, n$ , of inputs/outputs, features/labels, covariates/responses (which are referred to as the training data), the main goal of supervised learning is to predict a new  $y \in \mathcal{Y}$  given a new previously unseen  $x \in \mathcal{X}$ . The unobserved data are usually referred to as the testing data.

 There are few fundamental differences between machine learning and the branch of statistics dealing with regression and its various extensions, particularly when providing theoretical guarantees. The focus on algorithms and computational scalability is arguably stronger within machine learning (but also present in statistics). At the same time, the emphasis on models and their interpretability beyond their predictive performance is more prominent within statistics (but also present in machine learning).

**Examples.** Supervised learning is used in many areas of science, engineering, and industry. There are thus many examples where  $\mathcal{X}$  and  $\mathcal{Y}$  can be very diverse:

- **Inputs  $x \in \mathcal{X}$ :** they can be images, sounds, videos, text, proteins, sequences of DNA bases, web pages, social network activities, sensors from industry, financial time series, etc. The set  $\mathcal{X}$  may thus have a variety of structures that can be leveraged. All learning methods that we present in this textbook will use at one point a vector space representation of inputs, either by building an explicit mapping from  $\mathcal{X}$  to a vector space (such as  $\mathbb{R}^d$ ) or implicitly by using a notion of pairwise dissimilarity or similarity between pairs of inputs. The choice of these representations is highly domain-dependent. However, we note that (a) common topologies are encountered in many diverse areas (such as sequences, two-dimensional or three-dimensional objects), and thus common tools are used, and (b) learning these representations is an active area of research (see discussions in Chapter 7 and Chapter 9).

In this textbook, we will primarily consider that inputs are  $d$ -dimensional vectors, with  $d$  potentially large (up to  $10^6$  or  $10^9$ ).

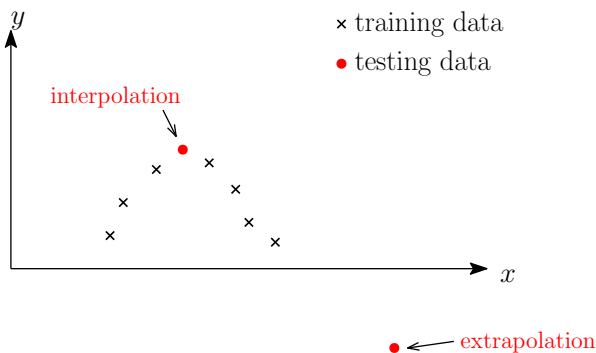
- **Outputs  $y \in \mathcal{Y}$ :** the most classical examples are binary labels  $\mathcal{Y} = \{0, 1\}$  or  $\mathcal{Y} = \{-1, 1\}$ , multicategory classification problems with  $\mathcal{Y} = \{1, \dots, k\}$ , and classical regression with real responses/outputs  $\mathcal{Y} = \mathbb{R}$ . These will be the main examples we treat in most of the book. Note, however, that most of the concepts extend to the more general *structured prediction* set-up, where more general structured outputs (e.g., graph prediction, visual scene analysis, source separation) can be considered (see Chapter 13).

**Why is it difficult?** Supervised learning is difficult (and thus interesting) for a variety of reasons:

- The label  $y$  may not be a deterministic function of  $x$ : given  $x \in \mathcal{X}$ , the outputs are noisy, that is,  $y$  is not a deterministic function of  $x$ . When  $y \in \mathbb{R}$ , we will often make the simplifying “additive noise” assumption that  $y = f(x) + \varepsilon$  with some zero-mean noise  $\varepsilon$ , but in general, we only assume that there is a conditional distribution of  $y$

given  $x$ . This stochasticity is typically due to diverging views between labelers or dependence on random external unobserved quantities (that is,  $y = f(x, z)$ , with  $z$  random and not observed).

- The prediction function  $f$  may be quite complex, highly non-linear when  $\mathcal{X}$  is a vector space, and even hard to define when  $\mathcal{X}$  is not a vector space.
- Only a few  $x$ 's are observed: we thus need interpolation and potentially extrapolation (see below for an illustration for  $\mathcal{X} = \mathcal{Y} = \mathbb{R}$ ), and therefore overfitting (predicting well on the training data but not as well on the testing data) is always a possibility.



Moreover, the training observations may not be uniformly distributed in  $\mathcal{X}$ . In this book, they will be assumed to be random, but some analyses will rely on deterministically located inputs to simplify some theoretical arguments.

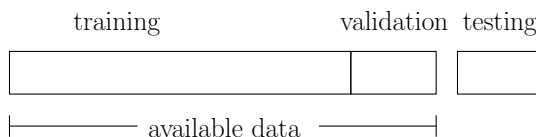
- The input space  $\mathcal{X}$  may be very large, that is, with high dimension when this is a vector space. This leads to both computational issues (scalability) and statistical issues (generalization to unseen data). One usually refers to this problem as the *curse of dimensionality*.
- There may be a weak link between training and testing distributions. In other words, the data at training time can have different characteristics than the data at testing time.
- The criterion for performance is not always well defined.

**Main formalization.** Most modern theoretical analyses of supervised learning rely on a probabilistic formulation, that is, we see  $(x_i, y_i)$  as a realization of random variables. The criterion is to maximize the expectation of some “performance” measure with respect to the distribution of the test data (in this book, *maximizing* the performance will be obtained by *minimizing* a loss function). The main assumption is that the random variables  $(x_i, y_i)$  are independent and identically distributed (i.i.d.) with the same distribution as the testing distribution. In this course, we will ignore the potential mismatch between train and test distributions (although this is an important research topic, as in most applications, training data are not i.i.d. from the same distribution as the test data).

A machine learning algorithm  $\mathcal{A}$  is a function that goes from a dataset, i.e., an element of  $(\mathcal{X} \times \mathcal{Y})^n$ , to a function from  $\mathcal{X}$  to  $\mathcal{Y}$ . In other words, the output of a machine learning algorithm is itself an algorithm!

**Practical performance evaluation.** In practice, we do not have access to the test distribution but samples from it. In most cases, the data given to the machine learning user are split into three parts:

- the *training set*, on which learning models will be estimated,
- the *validation set*, to estimate hyperparameters (all learning techniques have some) to optimize the performance measure,
- the *testing set*, to evaluate the performance of the final chosen model.



In theory, the test set can only be used once! In practice, this is unfortunately only sometimes the case. If the test data are seen multiple times, the estimation of the performance on unseen data is overestimated.

Cross-validation is often preferred to use a maximal amount of training data and reduce the variability of the validation procedure: the available data are divided in  $k$  folds (typically  $k = 5$  or  $10$ ), and all models are estimated  $k$  times, each time choosing a different fold as validation data (pink data below), and averaging the  $k$  obtained error measures. Cross-validation can be applied to any learning method, and its detailed theoretical analysis is an active area of research (see, [Arlot and Celisse, 2010](#), and the many references therein).



“Debugging” a machine learning implementation is often an art: on top of commonly found bugs, the learning method may not predict well enough on testing data. This is where theory can be useful to understand when a method is supposed to work or not. This is the primary goal of this book.

**Model selection.** Most machine learning models have hyper-parameters (e.g., regularization weight, size of the model, number of parameters). To estimate them from data, the common practical approach is to use validation approaches like those highlighted above. It is also possible to use penalization techniques based on generalization bounds. These two approaches are analyzed in Section 4.6.


**Random design vs. fixed design.** What we have described is often referred to as the “random design” set-up in statistics, where both  $x$  and  $y$  are assumed random and sampled i.i.d. It is common to simplify the analysis by considering that the input data  $x_1, \dots, x_n$  are deterministic, either because they are actually deterministic (e.g., equally spaced in the input space  $\mathcal{X}$ ), or by conditioning on them if they are actually random. This will be referred to as the “fixed design” setting and studied precisely in the context of least-squares regression in Chapter 3.


In the context of fixed design analysis, the error is evaluated “within-sample” (that is, for the same input points  $x_1, \dots, x_n$ , but over new associated outputs). This explicitly removes the difficulty of extrapolating to new inputs, hence a simplification in the mathematical analysis.

## 2.2 Decision theory

**Main question.** In this section, we tackle the following question: What is the optimal performance, regardless of the finiteness of the training data? In other words, what should be done if we have a perfect knowledge of the underlying probability distribution of the data? We will thus introduce the concept of *loss function*, *risk*, and “*Bayes*” *predictor*.

We consider a fixed (testing) distribution  $p_{(x,y)}$  on  $\mathcal{X} \times \mathcal{Y}$ , with marginal distribution  $p_{(x)}$  on  $\mathcal{X}$ . Note that we make no assumptions at this point on the input space  $\mathcal{X}$ .


 We will almost always use the overloaded notation  $p$ , to denote  $p_{(x,y)}$  and  $p_{(x)}$ , where the context can always make the definition unambiguous. For example, when  $f : \mathcal{X} \rightarrow \mathbb{R}$  and  $g : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R}$ , we have  $\mathbb{E}[f(x)] = \int_{\mathcal{X}} f(x) dp(x)$  and  $\mathbb{E}[g(x, y)] = \int_{\mathcal{X} \times \mathcal{Y}} g(x, y) dp(x, y)$ .

 We ignore on-purpose measurability issues. The interested reader can look at the book by [Christmann and Steinwart \(2008\)](#) for a more formal presentation.

### 2.2.1 Loss functions

We consider a loss function  $\ell : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$  (often  $\mathbb{R}_+$ ), where  $\ell(y, z)$  is the loss of predicting  $z$  while the true label is  $y$ .

 Some authors swap  $y$  and  $z$  in the definition above.

 Some related research communities (e.g., economics) use the concept of “utility”, which is then maximized.

The loss function only concerns the output space  $\mathcal{Y}$  independently of the input space  $\mathcal{X}$ . The main examples are:


- **Binary classification:**  $\mathcal{Y} = \{0, 1\}$  (or often  $\mathcal{Y} = \{-1, 1\}$ , or, less often, when seen as a subcase of the situation below,  $\mathcal{Y} = \{1, 2\}$ ), and  $\ell(y, z) = 1_{y \neq z}$  (“0-1” loss), that is, 0 if  $y$  is equal to  $z$  (no mistake), and 1 otherwise (mistake).



It is very common to mix the two conventions  $\mathcal{Y} = \{0, 1\}$  and  $\mathcal{Y} = \{-1, 1\}$ .

- **Multicategory classification:**  $\mathcal{Y} = \{1, \dots, k\}$ , and  $\ell(y, z) = 1_{y \neq z}$  (“0-1” loss).
- **Regression:**  $\mathcal{Y} = \mathbb{R}$  and  $\ell(y, z) = (y - z)^2$  (square loss). The absolute loss  $\ell(y, z) = |y - z|$  is often used for “robust” estimation (since the penalty for large errors is smaller).
- **Structured prediction:** while this textbook focuses primarily on the examples above, there are many practical problems where  $\mathcal{Y}$  is more complicated, with associated algorithms and theoretical results. For examples, when  $\mathcal{Y} = \{0, 1\}^k$  (leading to multi-label classification), the Hamming loss  $\ell(y, z) = \sum_{j=1}^k 1_{y_j \neq z_j}$  is commonly used; also, ranking problems involve losses on permutations. See Chapter 13.

Throughout the textbook, we will assume that the loss function is given to us. Note that in practice, the final user imposes the loss function, as this is how models will be evaluated. Clearly, a single real number may not be enough to characterize the entire prediction behavior. For example, in binary classification, there are two types of errors, false positives and false negatives, which can be considered simultaneously. Since we now have two performance measures, we typically need a curve to characterize the performance of a prediction function. This is precisely what “receiver operating characteristic” (ROC) curves are achieving (see, e.g., [Bach et al., 2006](#), and references therein). For simplicity, we stick to a single loss function  $\ell$  in this book.

 While the loss function  $\ell$  will be used to define the generalization performance below, for computational reasons, learning algorithms may explicitly minimize a different (but related) loss function, with better computational properties. This loss function used in training is often called a “surrogate”. This will be studied in the context of binary classification in Section 4.1, and more generally for structured prediction in Chapter 13.

## 2.2.2 Risks

Given the loss function  $\ell : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$ , we can define the *expected risk* (also referred to as *generalization performance*, or *testing error*) of a function  $f : \mathcal{X} \rightarrow \mathcal{Y}$ , as the expectation of the loss function between the output  $y$  and the prediction  $f(x)$ .

**Definition 2.1 (Expected risk)** *Given a prediction function  $f : \mathcal{X} \rightarrow \mathcal{Y}$ , a loss function  $\ell : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$ , and a probability distribution  $p$  on  $\mathcal{X} \times \mathcal{Y}$ , the expected risk of  $f$  is defined as:*

$$\mathcal{R}(f) = \mathbb{E}[\ell(y, f(x))] = \int_{\mathcal{X} \times \mathcal{Y}} \ell(y, f(x)) dp(x, y).$$

The risk depends on the distribution  $p$  on  $(x, y)$ . We sometimes use the notation  $\mathcal{R}_p(f)$

to make it explicit. The expected risk is our main performance criterion in this textbook.



Be careful with the randomness, or lack thereof, of  $f$ : when performing learning from data,  $f$  will depend on the random training data and not on the testing data, and thus  $\mathcal{R}(f)$  is typically random because of the dependence on the training data. However, as a function on functions, the expected risk  $\mathcal{R}$  is deterministic.

Note that sometimes, we consider random predictions, that is, for any  $x$ , we output a distribution on  $\mathcal{Y}$ , and then the risk is taken as the expectation over the randomness of the outputs.

Averaging the loss on the training data defines the *empirical risk*, or *training error*.

**Definition 2.2 (Empirical risk)** *Given a prediction function  $f : \mathcal{X} \rightarrow \mathcal{Y}$ , a loss function  $\ell : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}$ , and data  $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ ,  $i = 1, \dots, n$ , the empirical risk of  $f$  is defined as:*

$$\widehat{\mathcal{R}}(f) = \frac{1}{n} \sum_{i=1}^n \ell(y_i, f(x_i)).$$

Note that  $\widehat{\mathcal{R}}$  is a random function on functions (and is often applied to random functions, with dependent randomness as both will depend on the training data).

**Special cases.** For the classical losses defined earlier, the risks have specific formulations:

- **Binary classification:**  $\mathcal{Y} = \{0, 1\}$  (or often  $\mathcal{Y} = \{-1, 1\}$ ), and  $\ell(y, z) = 1_{y \neq z}$  (“0-1” loss). We can express the risk as  $\mathcal{R}(f) = \mathbb{P}(f(x) \neq y)$ . This is simply the probability of making a mistake on the testing data, while the empirical risk is the proportion of mistakes on the training data.
- **Multi-category classification:**  $\mathcal{Y} = \{1, \dots, k\}$ , and  $\ell(y, z) = 1_{y \neq z}$  (“0-1” loss). We can also express the risk as  $\mathcal{R}(f) = \mathbb{P}(f(x) \neq y)$ . This is also the probability of making a mistake.
- **Regression:**  $\mathcal{Y} = \mathbb{R}$  and  $\ell(y, z) = (y - z)^2$  (square loss). The risk is then equal to  $\mathcal{R}(f) = \mathbb{E}[(y - f(x))^2]$ , often referred to as mean squared error.

⚠ In practice, the *accuracy*, which is one minus the error rate, is often reported.

### 2.2.3 Bayes risk and Bayes predictor

Now that we have defined the performance criterion for supervised learning (the expected risk), the main question we tackle here is: what is the best prediction function  $f$  (regardless of the training data)?

Using the conditional expectation and its associated law of total expectation, we have

$$\mathcal{R}(f) = \mathbb{E}[\ell(y, f(x))] = \mathbb{E}[\mathbb{E}[\ell(y, f(x))|x]],$$

which we can rewrite, for a fixed  $x' \in \mathcal{X}$ :

$$\mathcal{R}(f) = \mathbb{E}_{x' \sim p} \left[ \mathbb{E}[\ell(y, f(x')) | x = x'] \right] = \int_{\mathcal{X}} \mathbb{E}[\ell(y, f(x)) | x = x'] dp(x').$$

⚠ To distinguish between the random variable  $x$  and a value it may take, we use the notation  $x'$ .

Given the conditional distribution given any  $x' \in \mathcal{X}$ , that is  $y|x = x'$ , we can define the *conditional risk* for any  $z \in \mathcal{Y}$  (it is a deterministic function):

$$r(z|x') = \mathbb{E}[\ell(y, z) | x = x'],$$

which leads to

$$\mathcal{R}(f) = \int_{\mathcal{X}} r(f(x')|x') dp(x').$$

To find a minimizing function  $f : \mathcal{X} \rightarrow \mathbb{R}$ , let us first assume that the set  $\mathcal{X}$  is finite: in this situation, the risk can be expressed as a sum of functions that depends on a *single* value of  $f$ , that is,  $\mathcal{R}(f) = \sum_{x' \in \mathcal{X}} r(f(x')|x') \mathbb{P}(x = x')$ . Therefore, we can minimize with respect to each  $f(x')$  *independently*. Therefore, a minimizer of  $\mathcal{R}(f)$  can be obtained by considering for any  $x' \in \mathcal{X}$ , the function value  $f(x')$  to be equal to a minimizer  $z \in \mathcal{Y}$  of  $r(z|x') = \mathbb{E}[\ell(y, z) | x = x']$ . This extends beyond finite sets, as shown below.

⚠ Minimizing the expected risk with respect to a function  $f$  in a restricted set does not lead to such decoupling.

**Proposition 2.1 (Bayes predictor and Bayes risk)** *The expected risk is minimized at a Bayes predictor  $f^* : \mathcal{X} \rightarrow \mathcal{Y}$  satisfying for all  $x' \in \mathcal{X}$ ,*

$$f^*(x') \in \arg \min_{z \in \mathcal{Y}} \mathbb{E}[\ell(y, z) | x = x'] = \arg \min_{z \in \mathcal{Y}} r(z|x'). \quad (2.1)$$

*The Bayes risk  $\mathcal{R}^*$  is the risk of all Bayes predictors and is equal to*

$$\mathcal{R}^* = \mathbb{E}_{x' \sim p} \left[ \inf_{z \in \mathcal{Y}} \mathbb{E}[\ell(y, z) | x = x'] \right].$$

**Proof** We have  $\mathcal{R}(f) - \mathcal{R}^* = \mathcal{R}(f) - \mathcal{R}(f^*) = \int_{\mathcal{X}} [r(f(x')|x') - \min_{z \in \mathcal{Y}} r(z|x')] dp(x')$ , which shows the proposition. ■

Note that (a) the Bayes predictor is not always unique, but that all lead to the same Bayes risk (for example, in binary classification when  $\mathbb{P}(y = 1|x) = 1/2$ ), and (b) that the Bayes risk is usually non zero (unless the dependence between  $x$  and  $y$  is deterministic). Given a supervised learning problem, the Bayes risk is the optimal performance; we define the excess risk as the deviation with respect to the optimal risk.

**Definition 2.3 (Excess risk)** *The excess risk of a function  $f : \mathcal{X} \rightarrow \mathcal{Y}$  is equal to  $\mathcal{R}(f) - \mathcal{R}^*$  (it is always non-negative).*

Therefore, machine learning is “trivial”: *given* the distribution  $y|x$  for any  $x$ , the optimal predictor is known and given by Eq. (2.1). The difficulty will be that this distribution is unknown.



**Special cases.** For our usual set of losses, we can compute the Bayes predictors in closed form:

- **Binary classification:** the Bayes predictor for  $\mathcal{Y} = \{0, 1\}$  and  $\ell(y, z) = 1_{y \neq z}$  is such that

$$\begin{aligned} f^*(x') \in \arg \min_{z \in \{0, 1\}} \mathbb{P}(y \neq z | x = x') &= \arg \min_{z \in \{0, 1\}} 1 - \mathbb{P}(y = z | x = x') \\ &= \arg \max_{z \in \{0, 1\}} \mathbb{P}(y = z | x = x'). \end{aligned}$$

The optimal classifier will select the most likely class given  $x'$ . Using the notation  $\eta(x') = \mathbb{P}(y = 1 | x = x')$ , then, if  $\eta(x') > 1/2$ ,  $f^*(x') = 1$ , while if  $\eta(x') < 1/2$ ,  $f^*(x') = 0$ . What happens for  $\eta(x') = 1/2$  is irrelevant.

The Bayes risk is then equal to  $\mathcal{R}^* = \mathbb{E}[\min\{\eta(x), 1 - \eta(x)\}]$ , which in general strictly positive (unless  $\eta(x) \in \{0, 1\}$  almost surely, that is,  $y$  is a deterministic function of  $x$ ).

This extends directly to multiple categories  $\mathcal{Y} = \{1, \dots, k\}$ , for  $k \geq 2$ , where we have  $f^*(x') \in \arg \max_{i \in \{1, \dots, k\}} \mathbb{P}(y = i | x = x')$ .

⚠ These Bayes predictors and risks are only valid for the 0-1 loss. Less symmetric losses are common in applications (e.g., for spam detection) and would lead to different formulas (see exercise below).

- **Regression:** the Bayes predictor for  $\mathcal{Y} = \mathbb{R}$  and  $\ell(y, z) = (y - z)^2$  is such that<sup>1</sup>

$$\begin{aligned} f^*(x') &\in \arg \min_{z \in \mathbb{R}} \mathbb{E}[(y - z)^2 | x = x'] \\ &= \arg \min_{z \in \mathbb{R}} \left\{ \mathbb{E}[(y - \mathbb{E}[y | x = x'])^2 | x = x'] + (z - \mathbb{E}[y | x = x'])^2 \right\}. \end{aligned}$$

This leads to the conditional expectation  $f^*(x') = \mathbb{E}[y | x = x']$ .

**Exercise 2.1** We consider binary classification with  $\mathcal{Y} = \{-1, 1\}$  with the loss function  $\ell(-1, -1) = \ell(1, 1) = 0$  and  $\ell(-1, 1) = c_- > 0$  (cost of a false positive),  $\ell(1, -1) = c_+ > 0$  (cost of a false negative). Compute a Bayes predictor at  $x$  as a function of  $\mathbb{E}[y | x]$ .

**Exercise 2.2** What is a Bayes predictor for regression with the absolute loss defined as  $\ell(y, z) = |y - z|$ ?

**Exercise 2.3** What is a Bayes predictor for regression with the “ $\varepsilon$ -insentive” loss defined as  $\ell(y, z) = \max\{0, |y - z| - \varepsilon\}$ ?

**Exercise 2.4** (inverting predictions) We consider the binary classification problem with  $\mathcal{Y} = \{-1, 1\}$  and the 0-1 loss. Relate the risk of a prediction  $f$  and its opposite  $-f$ .

---

<sup>1</sup>We use the law of total variance:  $\mathbb{E}[(y - a)^2] = \text{var}(y) + (\mathbb{E}[y] - a)^2$  for any random variable  $y$  and constant  $a \in \mathbb{R}$ , which can be shown by expanding the square.

**Exercise 2.5** (“chance” predictions) *We consider binary classification problems with the 0-1 loss, what is the risk of a random prediction rule where we predict the two classes with equal probabilities independently of the input  $x$ ? Same question with multiple categories.*

**Exercise 2.6** (♦) *We consider a random prediction rule where we predict from the probability distribution of  $y$  given  $x'$ . When is this achieving the Bayes risk?*

## 2.3 Learning from data

The decision theory framework outlined in the previous section gives a test performance criterion and optimal predictors, but it depends on the full knowledge of the test distribution  $p$ . We now briefly review how we can obtain good prediction functions from training data, that is, data sampled i.i.d. from the same distribution.

Two main classes of prediction algorithms will be studied in this textbook:

- (1) Local averaging (Chapter 6).
- (2) Empirical risk minimization (Chapters 3, 4, 7, 8, 9, 11, 12, 13).

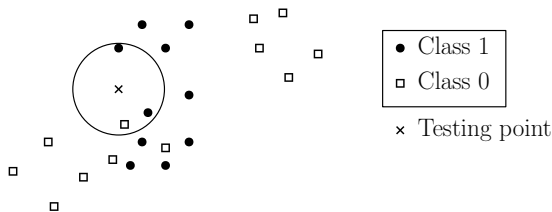
Note that there are prediction algorithms that do not fit precisely into one of these two categories, such as boosting or ensemble classifiers (see Chapter 10). Moreover, some situations do not fit the classical i.i.d. framework, such as in online learning (see Chapter 11). We consider probabilistic methods in Chapter 14, which rely on a different principle.

### 2.3.1 Local averaging

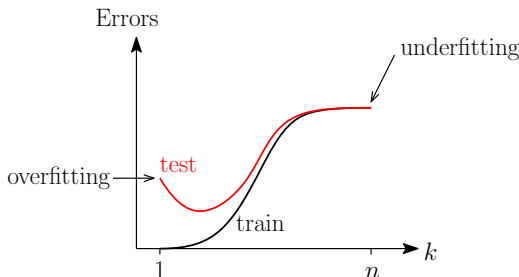
The goal here is to try to approximate/emulate the Bayes predictor, e.g.,  $f^*(x') = \mathbb{E}(y|x = x')$  for least-squares regression, from empirical data. This is often done by explicit/implicit estimation of the conditional distribution by *local averaging* ( $k$ -nearest neighbors, which is used as the primary example for this chapter, Nadaraya Watson, or decision trees). We briefly outline here the main properties for one instance of these algorithms; see Chapter 6 for details.

**$k$ -nearest-neighbor classification.** Given  $n$  observations  $(x_1, y_1), \dots, (x_n, y_n)$  where  $\mathcal{X}$  is a metric space and  $\mathcal{Y} \in \{0, 1\}$ , a new point  $x^{\text{test}}$  is classified by a majority vote among the  $k$ -nearest neighbors of  $x^{\text{test}}$ .

Below, we consider the 3-nearest-neighbor classifier on a particular testing point (which will be predicted as 1).



- Pros: (a) no optimization or training, (b) often easy to implement, (c) can get very good performance in low dimensions (in particular for non-linear dependences between  $x$  and  $y$ ).
- Cons: (a) slow at query time: must pass through all training data at each testing point (there are algorithmic tools to reduce complexity, see Chapter 6), (b) bad for high-dimensional data (because of the curse of dimensionality, more on this in Chapter 6), (c) the choice of local distance function is crucial, (d) the choice of “width” hyperparameters (or  $k$ ) has to be performed.
- Plot of training errors and testing errors as functions of  $k$  for a typical problem. When  $k$  is too large, there is *underfitting* (the learned function is too close to a constant, which is too simple), while for  $k$  too small, there is *overfitting* (there is a strong discrepancy between the testing and training errors).



**Exercise 2.7** How would the curve move when  $n$  increases (assuming the same balance between classes)?

### 2.3.2 Empirical risk minimization

Consider a parameterized family of prediction functions, often referred to as *models*,  $f_\theta : \mathcal{X} \rightarrow \mathcal{Y}$  for  $\theta \in \Theta$  (typically a subset of a vector space), and minimize the empirical risk with respect to  $\theta \in \Theta$ :

$$\hat{\mathcal{R}}(f_\theta) = \frac{1}{n} \sum_{i=1}^n \ell(y_i, f_\theta(x_i)).$$

This defines an estimator  $\hat{\theta} \in \arg \min_{\theta \in \Theta} \hat{\mathcal{R}}(f_\theta)$ , and thus a function  $f_{\hat{\theta}} : \mathcal{X} \rightarrow \mathcal{Y}$ .

The most classic example is linear least-squares regression (studied at length in Chapter 3), where we minimize  $\frac{1}{n} \sum_{i=1}^n (y_i - \theta^\top \varphi(x_i))^2$ , where  $f$  is linear in some feature vector

$\varphi(x) \in \mathbb{R}^d$  (there is no need for  $\mathcal{X}$  to be a vector space). The vector  $\varphi(x)$  can be quite large (or even implicit, like in kernel methods, see Chapter 7). Other examples include neural networks (Chapter 9).

- Pros: (a) can be relatively easy to optimize (e.g., least-squares with simple derivation and numerical algebra, see Chapter 3), many algorithms available (primarily based on gradient descent, see Chapter 5), (b) can be applied in any dimension (if a suitable feature vector is available).
- Cons: (a) can be relatively hard to optimize when the optimization formulation is not convex (e.g., neural networks), (b) need a suitable feature vector for linear methods, (c) the dependence on parameters can be complex (e.g., neural networks), (d) need some capacity control to avoid overfitting, (e) how to parameterize functions with values in  $\{0, 1\}$  (see Chapter 4 for the use of convex surrogates)?

**Risk decomposition.** The material in this section will be studied further in more detail in Chapter 4.

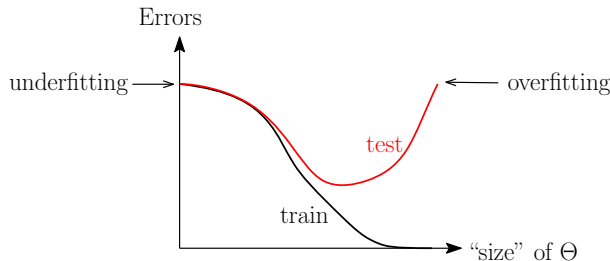
- Risk decomposition in estimation error + approximation error: given any  $\hat{\theta} \in \Theta$ , we can write the excess risk of  $f_{\hat{\theta}}$  as:

$$\begin{aligned} \mathcal{R}(f_{\hat{\theta}}) - \mathcal{R}^* &= \left\{ \mathcal{R}(f_{\hat{\theta}}) - \inf_{\theta' \in \Theta} \mathcal{R}(f_{\theta'}) \right\} + \left\{ \inf_{\theta' \in \Theta} \mathcal{R}(f_{\theta'}) - \mathcal{R}^* \right\} \\ &= \text{estimation error} \quad + \quad \text{approximation error.} \end{aligned}$$

The approximation error  $\inf_{\theta' \in \Theta} \mathcal{R}(f_{\theta'}) - \mathcal{R}^*$  is always non-negative, does not depend on the chosen  $f_{\hat{\theta}}$  and depends only on the class of functions parameterized by  $\theta \in \Theta$ . It is thus always a deterministic quantity, which characterizes the modeling assumptions made by the chosen class of functions. When  $\Theta$  grows, the approximation error goes down to zero if arbitrary functions can be approximated arbitrarily well by the functions  $f_{\theta}$ . It is also independent of  $n$ .

The estimation error  $\left\{ \mathcal{R}(f_{\hat{\theta}}) - \inf_{\theta' \in \Theta} \mathcal{R}(f_{\theta'}) \right\}$  is also always non-negative and is typically random because the function  $f_{\hat{\theta}}$  is random. It typically decreases in  $n$  and increases when  $\Theta$  grows.

Overall, the typical error curves look like this:



- Typically, we will see in later chapters that the estimation error is often decomposed

as follows, for  $\theta'$  a minimizer on  $\Theta$  of the expected risk  $\mathcal{R}(f_{\theta'})$ :

$$\begin{aligned}\mathcal{R}(f_{\hat{\theta}}) - \mathcal{R}(f_{\theta'}) &= \{\mathcal{R}(f_{\hat{\theta}}) - \hat{\mathcal{R}}(f_{\hat{\theta}})\} + \{\hat{\mathcal{R}}(f_{\hat{\theta}}) - \hat{\mathcal{R}}(f_{\theta'})\} + \{\hat{\mathcal{R}}(f_{\theta'}) - \mathcal{R}(f_{\theta'})\} \\ &\leq 2 \sup_{\theta \in \Theta} |\hat{\mathcal{R}}(f_{\theta}) - \mathcal{R}(f_{\theta})| + \text{empirical optimization error},\end{aligned}$$

where the “empirical optimization error” is  $\sup_{\theta \in \Theta} \{\hat{\mathcal{R}}(f_{\hat{\theta}}) - \hat{\mathcal{R}}(f_{\theta})\}$  (it is equal to zero for the exact empirical risk minimizer, but in practice when using optimization algorithms from Chapter 5, it is not). The uniform deviation  $\sup_{\theta \in \Theta} |\hat{\mathcal{R}}(f_{\theta}) - \mathcal{R}(f_{\theta})|$  grows with the “size” of  $\Theta$ , and usually decays with  $n$ . See more details in Chapter 4.

**Capacity control.** To avoid overfitting, we need to make sure that the set of allowed functions is not too large by typically reducing the number of parameters or by restricting the norm of predictors (thus by lowering the “size” of  $\Theta$ ): this typically leads to constrained optimization, and allows for risk decompositions as done above.

Capacity control can also be done by regularization, that is, by minimizing

$$\hat{\mathcal{R}}(f_{\theta}) + \lambda \Omega(\theta) = \frac{1}{n} \sum_{i=1}^n \ell(y_i, f_{\theta}(x_i)) + \lambda \Omega(\theta),$$

where  $\Omega(\theta)$  controls the complexity of  $f_{\theta}$ . The main example is ridge regression:

$$\min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n (y_i - \theta^{\top} \varphi(x_i))^2 + \lambda \|\theta\|_2^2.$$

This is often easier for optimization but harder to analyze (see Chapter 4 and Chapter 5).



There is a difference between parameters (e.g.,  $\theta$ ) learned on the training data and hyperparameters (e.g.,  $\lambda$ ) estimated on the validation data.

### Examples of approximations by polynomials in one-dimensional regression.

We consider  $(x, y) \in \mathbb{R} \times \mathbb{R}$ , with prediction functions which are polynomials of order  $k$ , from  $k = 0$  (constant functions) to  $k = 14$ . For each  $k$ , the model has  $k + 1$  parameters. The training error (using square loss) is minimized with  $n = 20$  observations. The data were generated with inputs uniformly distributed on  $[-1, 1]$  and outputs as the quadratic function  $f(x) = x^2 - \frac{1}{2}$  of the inputs plus some independent additive noise (Gaussian with standard deviation  $1/4$ ). As shown in Figure 2.1 and Figure 2.2, the training error monotonically decreases in  $k$  while the testing error goes down and then up.

## 2.4 Statistical learning theory

The goal of learning theory is to provide some guarantees of performance on unseen data. A common assumption is that the data  $\mathcal{D}_n(p) = \{(x_1, y_1), \dots, (x_n, y_n)\}$  is obtained as

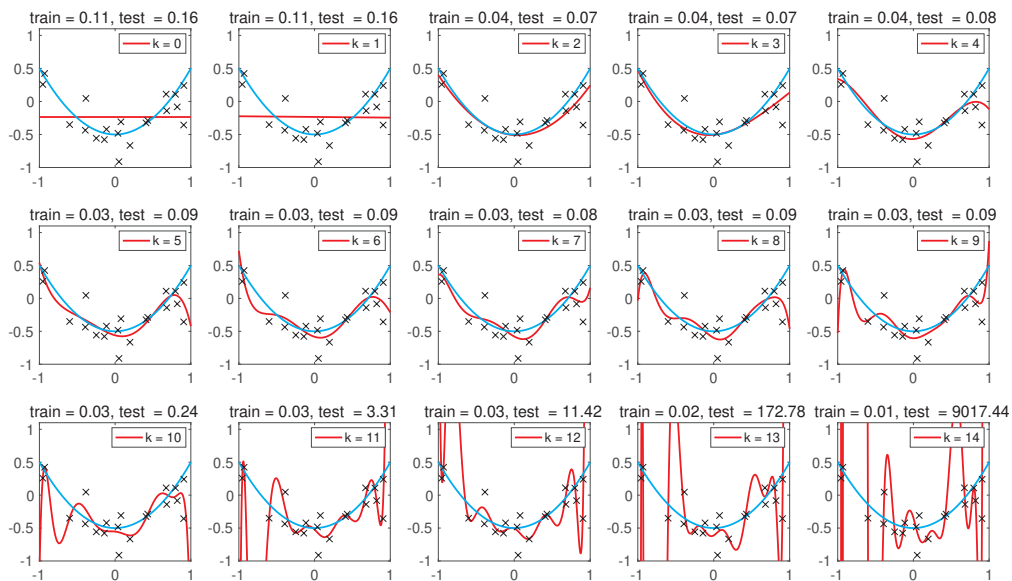


Figure 2.1: Polynomial regression with increasing orders  $k$ . Plots of estimated functions, with training and testing errors. The Bayes prediction function  $f^*(x) = \mathbb{E}[y|x]$  is plotted in blue.

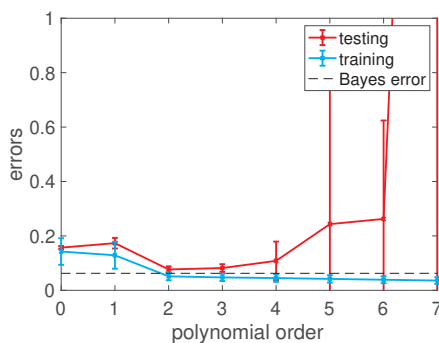


Figure 2.2: Polynomial regression with increasing orders. Plots of training and testing errors with error bars (computed as standard deviations obtained from 32 replications), together with the Bayes error.

independent and identically distributed (i.i.d.) observations from some unknown distribution  $p$  from a family  $\mathcal{P}$ .

An algorithm  $\mathcal{A}$  is a mapping from  $\mathcal{D}_n(p)$  (for any  $n$ ) to a function from  $\mathcal{X}$  to  $\mathcal{Y}$ . The expected risk depends on the probability distribution  $p \in \mathcal{P}$ , as  $\mathcal{R}_p(f)$ . The goal is to find  $\mathcal{A}$  such that the (expected) risk

$$\mathcal{R}_p(\mathcal{A}(\mathcal{D}_n(p))) - \mathcal{R}_p^*$$

is small, where  $\mathcal{R}_p^*$  is the Bayes risk (which depends on the joint distribution  $p$ ), assuming  $\mathcal{D}_n(p)$  is sampled from  $p$ , but without knowing which  $p \in \mathcal{P}$  is considered. Moreover, the risk is random because  $\mathcal{D}_n(p)$  is random.

### 2.4.1 Measures of performance

There are several ways of dealing with randomness to obtain a criterion.

- *Expected error*: we measure performance as

$$\mathbb{E}[\mathcal{R}_p(\mathcal{A}(\mathcal{D}_n(p)))],$$

where the expectation is with respect to the training data. An algorithm  $\mathcal{A}$  is called *consistent in expectation* for the distribution  $p$ , if

$$\mathbb{E}[\mathcal{R}_p(\mathcal{A}(\mathcal{D}_n(p)))] - \mathcal{R}_p^*$$

goes to zero when  $n$  tends to infinity. In this course, we will primarily use this notion of consistency.


- “*Probably approximately correct*” (PAC) learning: for a given  $\delta \in (0, 1)$  and  $\varepsilon > 0$ :

$$\mathbb{P}(\mathcal{R}_p(\mathcal{A}(\mathcal{D}_n(p))) - \mathcal{R}_p^* \leq \varepsilon) \geq 1 - \delta.$$

The crux is to find  $\varepsilon$ , which is as small as possible (typically as a function of  $\delta$ ). The notion of PAC consistency corresponds, for any  $\varepsilon > 0$ , to have such an inequality for each  $n$  and a sequence  $\delta_n$  that tends to zero.

### 2.4.2 Notions of consistency over classes of problems

An algorithm is called *universally consistent* (in expectation) if for all probability distributions  $p = p_{(x,y)}$  on  $(x, y)$  the algorithm  $\mathcal{A}$  is consistent in expectation for the distribution  $p$ .

 Be careful with the order of quantifiers: convergence speed will depend on  $p$ . See the no-free lunch theorem section below to highlight that having a uniform rate over all distributions is hopeless.

Most often, we want to study uniform consistency within a class  $\mathcal{P}$  of distributions satisfying some regularity properties (e.g., the inputs live in a compact space, or the

dependence between  $y$  and  $x$  is at most of some complexity). We thus aim at finding an algorithm  $\mathcal{A}$  such that

$$\sup_{p \in \mathcal{P}} \mathbb{E}[\mathcal{R}_p(\mathcal{A}(\mathcal{D}_n(p)))] - \mathcal{R}_p^*$$

is as small as possible. The so-called “minimax risk” is equal to

$$\inf_{\mathcal{A}} \sup_{p \in \mathcal{P}} \mathbb{E}[\mathcal{R}_p(\mathcal{A}(\mathcal{D}_n(p)))] - \mathcal{R}_p^*.$$

This is typically a function of the sample size  $n$  and properties of  $\mathcal{X}$ ,  $\mathcal{Y}$  and the allowed set of problems  $\mathcal{P}$  (e.g., dimension of  $\mathcal{X}$ , number of parameters). To compute estimates of the minimax risk, several techniques exist:

- Upper-bounding the optimal performance: one given algorithm with a convergence proof provides an upper bound. This is the main focus of this book.
- Lower-bounding the optimal performance: in some setups, it is possible to show that the infimum over all algorithms is greater than a certain quantity. See Chapter 15 for a description of techniques to obtain such lower bounds. Machine learners are happy when upper-bounds and lower-bounds match (up to constant factors).

**Non-asymptotic vs. asymptotic analysis.** The analysis can be “non-asymptotic”, with an upper bound with explicit dependence on all quantities; the bound is then valid for all  $n$ , even if sometimes vacuous (e.g., a bound greater than 1 for a loss uniformly bounded by 1).

The analysis can also be “asymptotic”, where, for example,  $n$  goes to infinity and limits are taken (alternatively, several quantities can be made to grow simultaneously).



What (arguably) matters most here is the dependence of these rates on the problem, not the choice of “in expectation” vs. “in high probability”, or “asymptotic” vs. “non-asymptotic”, as long as the problem parameters explicitly appear.

## 2.5 No free lunch theorems (♦)

Although it may be tempting to define the optimal learning algorithm that works optimally for all distributions, this is impossible. In other words, learning is only possible with assumptions. See Devroye et al. (1996, Chapter 7) for more details.

The following theorem shows that for any algorithm, for a fixed  $n$ , there is a data distribution that makes the algorithm useless (with a risk that is the same as the chance level).

**Proposition 2.2 (no free lunch - fixed  $n$ )** *Consider the binary classification with 0-1 loss, with  $\mathcal{X}$  infinite. Let  $\mathcal{P}$  denote the set of all probability distributions on  $\mathcal{X} \times \{0, 1\}$ .*



For any  $n > 0$  and any learning algorithm  $\mathcal{A}$ ,

$$\sup_{p \in \mathcal{P}} \mathbb{E}[\mathcal{R}_p(\mathcal{A}(\mathcal{D}_n(p)))] - \mathcal{R}_p^* \geq 1/2.$$

**Proof (◆◆)** Let  $k$  be a positive integer. Without loss of generality, we can assume that  $\mathbb{N} \subset \mathcal{X}$ . The main ideas of the proof are (a) to construct a probability distribution supported on  $k$  elements in  $\mathbb{N}$ , where  $k$  is large compared to  $n$  (which is fixed), and to show that the knowledge of  $n$  labels does not imply doing well on all  $k$  elements, and (b) to choose parameters of this distribution (the vector  $r$  below) by comparing to a performance obtained by random parameters.

Given  $r \in \{0, 1\}^k$ , we define the joint distribution  $p$  on  $(x, y)$  such that we have  $\mathbb{P}(x = j, y = r_j) = 1/k$  for  $j \in \{1, \dots, k\}$ ; that is, for  $x$ , we choose one of the first  $k$  elements uniformly at random, and then  $y$  is selected deterministically as  $y = r_x$ . Thus, the Bayes risk is zero (because there is a deterministic relationship):  $\mathcal{R}_p^* = 0$ .

Denoting  $\hat{f}_{\mathcal{D}_n} = \mathcal{A}(\mathcal{D}_n(p))$  the classifier, and  $S(r) = \mathbb{E}[\mathcal{R}_p(\hat{f}_{\mathcal{D}_n})]$  the expectation of the expected risk, we want to maximize  $S(r)$  with respect to  $r \in \{0, 1\}^k$ ; the maximum is greater than the expectation of  $S(r)$  for any probability distribution  $q$  on  $r$ , in particular the uniform distribution (each  $r_j$  being an independent unbiased Bernoulli variable). Then

$$\begin{aligned} \max_{r \in \{0, 1\}^k} S(r) &\geq \mathbb{E}_{r \sim q} S(r) \\ &= \mathbb{P}(\hat{f}_{\mathcal{D}_n}(x) \neq y) = \mathbb{P}(\hat{f}_{\mathcal{D}_n}(x) \neq r_x), \end{aligned}$$

because  $x$  is almost surely in  $\{1, \dots, k\}$  and  $y = r_x$  almost surely. Note that we take expectations and probabilities with respect to  $x_1, \dots, x_n, x$ , and  $r$  (all being independent of each other).

Then, we get, using that  $\mathcal{D}_n(p) = \{x_1, r_{x_1}, \dots, x_n, r_{x_n}\}$ :

$$\begin{aligned} \mathbb{E}_{r \sim q} S(r) &= \mathbb{E} \left[ \mathbb{P}(\hat{f}_{\mathcal{D}_n}(x) \neq r_x | x_1, \dots, x_n, r_{x_1}, \dots, r_{x_n}) \right] \text{ by the law of total expectation,} \\ &\geq \mathbb{E} \left[ \mathbb{P}(\hat{f}_{\mathcal{D}_n}(x) \neq r_x \ \& \ x \notin \{x_1, \dots, x_n\} | x_1, \dots, x_n, r_{x_1}, \dots, r_{x_n}) \right] \\ &\hspace{15em} \text{by monotonicity of probabilities,} \\ &= \mathbb{E} \left[ \frac{1}{2} \mathbb{P}(x \notin \{x_1, \dots, x_n\} | x_1, \dots, x_n, r_{x_1}, \dots, r_{x_n}) \right], \end{aligned}$$

because  $\mathbb{P}(\hat{f}_{\mathcal{D}_n}(x) \neq r_x | x \notin \{x_1, \dots, x_n\}, x_1, \dots, x_n, r_{x_1}, \dots, r_{x_n}) = 1/2$  (the label  $x = r_x$  has the same probability of being 0 or 1, given that it was not observed). Thus,

$$\mathbb{E}_{r \sim q} S(r) \geq \frac{1}{2} \mathbb{P}(x \notin \{x_1, \dots, x_n\}) = \frac{1}{2} \mathbb{E} \left[ \prod_{i=1}^n \mathbb{P}(x_i \neq x | x) \right] = \frac{1}{2} (1 - 1/k)^n.$$

Given  $n$ , we can let  $k$  tend to infinity to conclude. ■

A caveat is that the hard distribution may depend on  $n$  (from the proof, it takes  $k$  values, with  $k$  tending to infinity fast enough compared with  $n$ ). The following theorem is given without proof and is much “stronger” (Devroye et al., 1996, Theorem 7.2), as it more convincingly shows that learning can be arbitrarily slow without assumption (note that the earlier one is not a corollary of the later one).

**Proposition 2.3 (no free lunch - sequence of errors)** *Consider a binary classification problem with the 0-1 loss, with  $\mathcal{X}$  infinite. Let  $\mathcal{P}$  denote the set of all probability distributions on  $\mathcal{X} \times \{0, 1\}$ . For any decreasing sequence  $a_n$  tending to zero and such that  $a_1 \leq 1/16$ , for any learning algorithm  $\mathcal{A}$ , there exists  $p \in \mathcal{P}$ , such that for all  $n \geq 1$ :*

$$\mathbb{E}[\mathcal{R}_p(\mathcal{A}(\mathcal{D}_n(p)))] - \mathcal{R}_p^* \geq a_n.$$

## 2.6 Quest for adaptivity

As seen in the previous section, no method can be universal and achieve a good convergence rate on all problems. However, such negative results consider classes of problems that are arbitrarily large. In this textbook, we will consider reduced sets of learning problems by considering  $\mathcal{X} = \mathbb{R}^d$  and putting restrictions on the target function  $f^*$  based on smoothness and/or dependence on an unknown low-dimensional projection. That is, the most general set of functions will be the set of Lipschitz-continuous functions, for which the optimal rate will be essentially proportional to  $O(n^{-1/d})$ , typical of the curse of dimensionality. No method can beat this, not  $k$ -nearest-neighbors, not kernel methods, not even neural networks.

When the target function is smoother, that is, with all derivatives up to order  $m$  bounded, then we will see that kernel methods (Chapter 7) and neural networks (Chapter 9), with the proper choice of the regularization parameter, will lead to the optimal rate of  $O(n^{-m/d})$ .

When the target function moreover depends only on a  $k$ -dimensional linear projection, neural networks (if the optimization problem is solved correctly) will have the extra ability to lead to rates of the form  $O(n^{-m/k})$  instead of  $O(n^{-m/d})$ , which is not the case for kernel methods (see Chapter 9)

Note that another form of adaptivity, which is often considered, is in situations where the input data lie on a submanifold of  $\mathbb{R}^d$  (e.g., an affine subspace), where for most methods presented in this textbook, adaptivity is obtained. In the convergence rate,  $d$  can be replaced by the dimension of the subspace (or submanifold) where the data live. See studies by Kpotufe (2011) for  $k$ -nearest neighbors, and Hamm and Steinwart (2021) for kernel methods.

See more details in <https://francisbach.com/quest-for-adaptivity/> as well as Chapter 7 and Chapter 9 for detailed results.

## 2.7 Beyond supervised learning

This textbook focuses primarily on the traditional supervised learning paradigm, with independent and identically distributed data and where the training and testing distributions match. Many applications require extensions to this basic framework, which also lead to many interesting theoretical developments that are out of scope. We present briefly some of these extensions below with references for further reading.

**Unsupervised learning.** While in supervised learning, both inputs and outputs (e.g., labels) are observed, and the main goal is to model how the output depends on the input, in unsupervised learning, only inputs are given. The goal is then to “find some structure” within the data, for example, an affine subspace around which the data live (for principal component analysis), the separation of the data in several groups (for clustering), or the identification of an explicit latent variable model (such as with matrix factorization). The new representation of the data is typically either used for visualization (then, with two or three dimensions), or for reducing dimension before applying a supervised learning algorithm.

While supervised learning relied on an explicit decision-theoretic framework, it is not always clear how to characterize performance and perform evaluation; each method typically has an ad-hoc empirical criterion, such as reconstruction of the data, full or partial (like in self-supervised learning), log-likelihood when probabilistic models are used (see Chapter 14), in particular graphical models (Bishop, 2006; Murphy, 2012). Often, intermediate representations are used for subsequent processing (see, e.g., Goodfellow et al., 2016).

Theory can be applied to sampling behavior and recovery of specific structures when assumed (e.g., for clustering or dimension reduction), with a variety of theoretical results in manifold learning, matrix factorization methods such as K-means, principal component analysis or sparse dictionary learning (Mairal et al., 2014), outlier/novelty detection, or independent component analysis (Hyvärinen et al., 2001).

**Semi-supervised learning.** This is the intermediate situation between supervised and unsupervised, with some labeled (typically few) examples and some unlabeled (typically many) examples. Several frameworks exist based on various assumptions (Chapelle et al., 2010; Van Engelen and Hoos, 2020).

**Active learning.** This is a similar setting as semi-supervised learning, but the user can choose which unlabelled point to label to maximize performance once new labels are obtained. The selection of samples to label is often done by computing some form of uncertainty estimation on the unlabelled data points (see, e.g. Settles, 2009).

**Online learning.** Mostly in a supervised setting, this framework allows us to go beyond the training/testing splits, where data are acquired, and predictions are made on the fly,

with a criterion that takes into account the sequential nature of learning. See [Cesa-Bianchi and Lugosi \(2006\)](#); [Hazan \(2022\)](#) and Chapter 11.

**Reinforcement learning.** On top of the sequential nature of learning already present in online learning, predictions may influence the future sampling distributions, for example, in situations where some agents interact with an environment ([Sutton and Barto, 2018](#)), with algorithms relying on similar concepts than optimal control ([Liberzon, 2011](#)).

## 2.8 Summary - book outline

Now that the main concepts are introduced, we can give an outline of the book chapters, which we have separated into three parts.

**Part I: Preliminaries.** The first part contains this introductory chapter and Chapter 3 on linear least-squares regression. We start with such a chapter as it allows for the introduction of the main concepts of the book, such as underfitting, overfitting, regularization, using simple linear algebra, and without the need for more advanced analytic or probabilistic tools.

**Part II: Generalization bounds for learning algorithms.** The second part is dedicated to the core concepts in learning theory and should be studied sequentially.

- **Empirical risk minimization:** Chapter 4 is dedicated to methods based on the minimization of the potentially regularized or constrained regularized risk, with the introduction of the key concept of Rademacher complexity that analyzes estimation errors efficiently. Convex surrogates for binary classification are also introduced to allow the use of only real-valued prediction functions.
- **Optimization:** Chapter 5 shows how gradient-based techniques can be used to minimize approximately the empirical risk and, through stochastic gradient descent, obtain generalization bounds for finitely-parameterized linear models (which are linear in their parameters) leading to convex objective functions.
- **Local averaging methods:** Chapter 6 is the first chapter dealing with so-called “non-parametric” methods that can potentially adapt to complex prediction functions. This class of methods explicitly builds a prediction function mimicking the Bayes predictor (without any optimization algorithm), such as  $k$ -nearest-neighbor methods. These methods are classically subject to the curse of dimensionality.
- **Kernel methods:** Chapter 7 presents the most general class of linear models that can be infinite-dimensional and adapt to complex prediction functions. They are made computationally feasible using the “kernel trick”, and they still rely on convex optimization, so they lead to strong theoretical guarantees, particularly by being adaptive to the smoothness of the target prediction function.
- **Sparse methods:** While the previous chapter focused on Euclidean or Hilbertian regularization techniques for linear models, Chapter 8 considers regularization

by sparsity-inducing penalties such as the  $\ell_1$ -norm or the  $\ell_0$ -penalty, leading to the high-dimensional phenomenon that learning is possible even with potentially exponentially many irrelevant variables.

- **Neural networks:** Chapter 9 presents a class of prediction functions that are not linearly parameterized, therefore leading to non-convex optimization problems, where obtaining a global optimum is not certain. The chapter studies approximation and estimation errors, showing the adaptivity of neural networks to smoothness and linear latent variables (in particular for non-linear variable selection).

**Part III: Special topics.** The third part presents a series of special topic chapters that can be read in essentially any order.

- **Ensemble learning:** Chapter 10 presents a class of techniques aiming at combining several predictors obtained from the same model class but learned on slightly modified datasets. This can be done in parallel, such as in bagging techniques, or sequentially, like boosting methods.
- **From online learning to bandits:** Chapter 11 consider sequential decision problems within the regret framework, focusing first on online convex optimization, then on zeroth order optimization (without access to gradients), and then multi-armed bandits.
- **Over-parameterized models:** Chapter 12 presents a series of results related to models with a large number of parameters (enough to fit the training data perfectly) and trained with gradient descent. We present the implicit bias of gradient descent in linear models towards minimum Euclidean norm solutions and then the double descent phenomenon, before looking at implicit biases and global convergence for non-convex optimization problems.
- **Structured prediction:** Chapter 13 goes beyond the traditional regression and binary classification frameworks by first considering multi-category classification and then the general framework of structured prediction, where output spaces can be arbitrarily complex.
- **Probabilistic methods:** Chapter 14 presents a collection of results related to probabilistic modeling, highlighting that probabilistic interpretations can sometimes be misleading but also naturally lead to model selection frameworks through Bayesian inference and PAC-Bayes analysis.
- **Lower bounds on performance:** While most of the book is dedicated to obtaining upper bounds on the generalization or optimization performance of our algorithms, Chapter 15 considers lower-bounds on such performance, showing how many algorithms presented in this book are, in fact, “optimal” for a specific class of learning or optimization problems.

