Gabriel Stoltz

An Introduction to Machine Learning

May 6, 2026

IP Paris – M1 Applied Mathematics and Statistics

Preface

Machine learning is not a spectators' sport

Bibliographical guide

The main reference for this course is the introductory book [41] by Kevin Murphy, which focuses on algorithms and numerical methods. It covers a wide range of methods and techniques, and can be found online for free at the url

`https://probml.github.io/pml-book/book1.html`

Another important reference, for the more theoretical parts of the course, is the book on learning theory from Francis Bach [4], which can be found at the page

`https://www.di.ens.fr/%7Efbach/ltfp_book.pdf`

Other nice references mentioned in the course include [].

S. Shalev-Shwartz and S. Ben-David, Understanding Machine Learning M. Mohri, A. Rostamizadeh and A. Talwalkar, Foundations of Machine Learning D. Barber, Bayesian Reasoning and Machine Learning C. Bishop, Pattern Recognition and Machine Learning E. Alpaydin, Introduction to Machine Learning P. Mehta, M. Bukov, C.-H. Wang, A.G.R. Day, C. Richardson, C.K. Fisher, D.J. Schwab, A high-bias, low-variance introduction to Machine Learning for physicists, Physics Reports 810, 1-124 (2019) [arXiv reference]

Exercises

More difficult questions are indicated by this symbol in the margin. These questions would typically not be asked at the exam.

Some practical comments

This set of lecture notes is the first typed version I have on this material. It therefore certainly contains various typos. Do not hesitate to point me out anything you spot!

Paris, 18 July 2023

Gabriel Stoltz

Contents

1

A first experience in machine learning

Not everything can be learned. The question which underlies machine learning at large is: *What can be learned, and under which conditions?* This chapter gives an introduction to machine learning, by first discussing in Section 1.1 what machine learning is about through some overview of the field. We next turn in Section 1.2 to supervised learning, which is the type of learning we will mostly consider in these lecture notes. We conclude this introductory chapter with our first examples of machine learning methods, based on local averaging, in particular K-nearest neighbors (see Section 1.3).

1.1 What is machine learning?

We give here an overview of machine learning based on [41, Chapter 1], [3, Chapter 1], [40, Chapter 1], [6, Chapter 13] and [50, Chapter 1].

1.1.1 The (big) data paradigm

Although machine learning as a scientific field has been present since decades, it gained a lot of popularity in the past decade, thanks to a new paradigm where

- data can be abundant and easily obtained, and therefore comes first in many settings (before the system is even being modeled);
- efficient computing devices are available.

Examples of situations where data is easily obtained include: internet trafic (creation of data when visiting webpages), music or movie ratings (when liking songs on some apps, writing a review for a movie, ...), customer behaviors (recording information on products bought online), etc.

Gathering data is not a goal in itself. What one ultimately aims at is to predict new events or behaviors based on the gathered information. For instance, a supermarket chain may want to be able to say which customer is likely to buy which product, when, how much, ... while the customer may want to find products that fit his/her needs. Another example is spam classification: the user may want to tailor the classifier so that it better sorts the incoming emails. A last example is character recognition in images, for instance for automatic recognition of amounts when depositing bank checks. In all these situations, one may lack a good knowledge of the underlying process, and be unable to model it convincingly. The idea is then to make up for this by relying on accumulated experience in the form of data. This can be coined as "re-cognizing" events or data, relying on the definition of "cognize" as "perceive, become aware of".

1.1.2 Definitions

Machine learning is a subfield of artificial intelligence,[1] and has strong links with data mining, data analysis and data visualization. It can be defined in various ways, including

- automatically detecting meaningful patterns in data;
- transforming past information/experience (data) into predictions as accurate as possible;
- improving the performance at certain tasks when having more experience.

All these statements explicitly or implicitly suggest that machine learning is about designing accurate and efficient prediction *algorithms* (to be emphasized as a clear list of instructions/procedures to follow in order to obtain a result). The so-obtained algorithm should be able to adapt to modifications in behaviors. Illustrative examples to this end are spam classification and fraud detection.

1.1.2.1 Machine learning tasks

The application domains of machine learning are ubiquitous:

- science (processing of measurement data in astronomy, physics; DNA analysis in biology, ...)
- medicine (for instance medical diagnosis)
- image recognition/segmentation (for medical applications, social media, etc)
- text analysis/classification
- speech processing/natural language processing (for instance translations)
- finance/banking (credit applications, fraud detection, prediction of stock market prices, ...)
- playing games (chess, go)

These application domains require machine learning tools for various tasks:

[1] There are many definitions of artificial intelligence (AI)... We use here elements from the Wikipedia page of the topic. In these notes, we consider AI as a scientific field studying and developing "the intelligence of machines or software, as opposed to the intelligence of humans or other animals [...] The various subfields of IA research are centered around particular goals and the use of particular tools. The traditional goals of AI research include reasoning, knowledge representation, planning, learning, natural language processing, perception, and support for robotics. General intelligence (the ability to complete any task performable by a human) is among the field's long-term goals."

- classification (many possible applications: credit scoring, digit classification, face recognition and biometrics, ...), see for instance Chapters 3 and 6;
- regression, see for instance Chapter 2;
- ranking (which corresponds to ordering items given a criterion, e.g. webpages when performing a query on google);
- clustering (for example to find communities in social networks), see Chapter 9;
- dimensionality reduction (e.g. for image compression), see Chapter 5 and Section 8.3;
- data generation (for instance with tools such as ChatGPT, Code-Llama, Dall-E, ...).

Exercise 1.1. *Find other examples of machine learning tasks.*

1.1.2.2 Relationship with other scientific fields

Machine learning is a proteiform scientific field, at the crossroad of various other fields, in particular:

- *probability and statistics.* Distinctive features of machine learning are (i) the fact that the distribution of the data is unknown (distribution free setting), so that theoretical results are established over (large) classes of distributions; (ii) non asymptotic results are of prime interest, as the number of data points may not be large in comparison with the number of degrees of freedom/unknowns of the machine learning model; (iii) the key output of machine learning algorithms are predictions on unseen data, so that metrics of success are based on this criterion, and not on the quality of the estimation of the parameters of the model as in statistics. Somehow, machine learning cares less about checking statistical hypotheses than finding causes for the observed data. It also has a more computer science oriented mindset than traditional statistics.
- *optimization theory*, as many models of machine learning require some form of parameter optimization with respect to some loss/cost function. This optimization is performed in (very) high dimension, often for non convex targets.
- *computer science*, as the algorithmic part of machine learning is pivotal. A particular attention is paid to equilibrating the prediction performance and the computational cost of the methods which are being implemented.
- *artificial intelligence*: machine learning can be considered as a specific component, which allows to transform "experience" (in the form of data) into "actions" (based on the predictions provided by the algorithm). There is however no attempt to emulate or reproduce human intelligence in machine learning.

1.1.3 Types of learning

In these lecture notes, we will mostly perform *supervised learning*, although *unsupervised learning* will also be considered to some extent. This will be done in a *statistical framework*, using *batch learning* with a *passive teacher*. In order to make sense of these sentences, let us comment on the various alternative options:

- *Supervised learning* is described more precisely in Section 1.2. In short, this means that data points come with labels, as opposed to *unsupervised learning* where data is unlabeled. Sometimes, a semi-supervised setting is considered, where only a fraction of the data points are labeled. A typical example of supervised learning is spam detection, for which the training database needs to include examples of spams and legitimate emails in order to make predictions on new incoming emails. An example of unsupervised learning for emails would be anomaly detection, *i.e.* marking a new incoming email as deviating from the usual email content. Supervised learning is intuitively more efficient, as some extra information is available compared to unsupervised learning. However, attaching a label to the data requires some (possibly costly and time consuming) preprocessing of the data – think for instance about medical diagnosis, where a medical doctor needs to go over various images and pinpoint whether there

is a pathology or not. This motivates working with unlabeled data, or only with a fraction of labeled data.
- *Active learning* corresponds to settings where the algorithm can request new data to be generated in some regions of data space; whereas, in *passive learning*, the data is given to the algorithm without the possibility to give some feedback on the next data location.
- *Online learning* arises in situations where data continuously flows in, while *batch learning* corresponds to settings where the data is handed out at the beginning of the procedure. One example of online learning is finance, where stockbrokers aim at predicting stock market prices in real time for their daily decisions.
- *Passive* vs. *adversarial training* depends on the helpfulness of the teacher. A passive teacher is encountered when the data is random, which corresponds to the framework of statistical learning. This should be contrasted with the adversarial scenario, where data is generated in order to trick the learner into making mistakes. This is useful for algorithms targeted at fraud detection for instance, or more generally to obtain guarantees in the worst case scenario (in opposition to typical scenarios, as in the statistical learning framework).

Unsupervised learning. From a mathematical viewpoint, and anticipating to some extent on the presentation of supervised learning in Section 1.2, unsupervised learning aims at fitting an unconditional model $p_{\text{data}}(x)$ which can generate new data x, as opposed to supervised learning which considers data points x and labels y, and fits a conditional model $p_{\text{data}}(x|y)$ in order to make predictions. In essence, unsupervised learning aims at finding compact descriptions of data. The main interest of this learning framework is that it avoids to collect labeled data. It can also be considered in situations where the labeling is ambiguous (as for text or image classifications when some categories are close to each other). More generally, it allows to find patterns or "explanations" in high dimensional data instead of focusing on low dimensional inputs only.

One difficulty with unsupervised learning is the absence of ground truth provided by labels. Evaluating the quality of the model therefore requires some work. Possible approaches to this end are (i) to check the likelihood of the generated data (using some test data and density estimations); (ii) to use the learned unsupervised (reduced) representation of the data for classification/regression tasks and hope to be more efficient than with the full data; (iii) sometimes, a qualitative gain in the understanding of the model ("interpretability") can be considered as a good enough motivation for unsupervised learning.

Reinforcement learning. Another learning framework, which we will not touch upon, is provided by reinforcement learning, where an agent learns to interact with an environment, through some policy (list of actions), which is updated using some reward function. In this framework, data is also unlabeled, but the output of the algorithm is a balance between exploration of new data (which can be seen as taking actions in a dynamic environment) and exploitation of the data collected until now (to update the parameters of the method in order to make better predictions, the metric for the quality of the prediction being some reward function). A prominent example is provided by learning to play chess or go. In essence, reinforcement learning amounts to "learning with a critic" (with some occasional thumbs up or down) as opposed to "learning with a teacher".

1.1.4 Learning stages

We briefly discuss in this section the overall workflow associated with machine learning techniques. Only the first step is made precise below, the other ones being discussed in the following chapters:

(1) Preparation of the data: The first step is to collect data/examples, to preprocess them by curating the dataset, and next apply some featurization procedure and label the data points for supervised learning. The dataset can then be decomposed into three subsets: a training dataset, a validation dataset (which will be used to find the best parameters of the model) and a test set. The latter set is never used to find the parameters of the model; its sole purpose is to assess the quality of the prediction.
(2) Choose a loss function, which measures the performance of the prediction.

(3) Choose the hypothesis set (i.e. the class of models which is considered) and the hyperparameters for the algorithm.
(4) Find the parameters of the model leading to the best prediction by (i) for a given value of the hyperparameters, finding the parameters of the model giving the smallest loss for the training dataset; (ii) selecting the hyperparameters to obtain the smallest loss on the validation set; (iii) assessing the quality of the so-obtained prediction on the test set.

A focus on data preparation. The nature and quality of the data is key to the success of the learning process. Although this is of paramount importance in practice and requires quite some care, we consider in these lecture notes that the data has been correctly prepared, and therefore focus on the algorithmic and theoretical aspects of learning. In order to test methods, there are nowadays various benchmark datasets, of increasing complexity. For instance, for image classification, one can start with the rather simple MNIST dataset for which the task is to classify handwritten digits, and then progressively complexify the classification task by considering fashion-MNIST (recognizing clothing items), the various CIFAR datasets, the ImageNet dataset, etc. Some websites such as Kaggle gather standard datasets (see also the references on the course webpage).

After collecting raw data, the first task is to remove duplicates, and check for missing data. Various rules exist in the latter case to make up for missing fields (mean value imputation, use of generative methods to fill in fields, etc). Categorical data, such as colors (for a wine, an item of clothing, an animal, etc), genders, etc., require some care in the way they are handled, as algorithms and computers work with numbers. The customary way of proceeding is to use one-hot encoding to transform a categorical variable with K possible values into a vector $\{0,1\}^K$ with only one non-zero entry; more precisely, the categorical component x_i is transformed into the vector $(\mathbf{1}_{x_i=1}, \dots, \mathbf{1}_{x_i=K})$ (the interest of this transformation will become more clear in contexts such as the one of Section 3.2.4 on multiclass logistic regression). This is one instance of featurization, the process of transforming the raw data into a more relevant input. This featurization step can be much more involved for certain applications. For instance, in order to predict forces in atomistic models of materials, the raw input, which is the atomistic configuration of a system, is featurized using various functions computing radial and angular moments of the distribution of neighboring atoms around each other atom. Another example is provided by data inputs which are not of the same sizes, such as sequences of words or chunks of temporal series. Let us emphasize here that coming up with (or learning) a good representation of the data is still an active research field in various application domains of machine learning. In any case, some exploratory data analysis needs to be conducted in order to understand the salient properties of the dataset; and possibly remove some attributes/features (for instance through pairwise scatter plots to identify redundant or irrelevant features).

In the remainder of these lecture notes, we always consider that the data is organized in the following design matrix, where each row represents one of the n data point $x_i = (x_{i,1}, \dots, x_{i,d}) \in \mathbb{R}^{1\times d}$, and each of the d columns represents a feature (also called attribute):

$$X = \begin{pmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,d} \\ x_{2,1} & x_{2,2} & \dots & x_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{n,1} & x_{n,2} & \dots & x_{n,d} \end{pmatrix} \in \mathbb{R}^{n\times d}.$$

Normalization. In many methods, it is convenient to have data features of similar sizes, and in fact of order 1. This can be achieved through linear transformations. The three main options to this end are

- *standardization:* this corresponding to a normalization of the data component by component, *i.e.* centering the data in each column and rescaling it so that the empirical variance is 1. More precisely, this amounts to setting $\widetilde{x}_{i,k} = (x_{i,k} - b_k)/a_k$ (alternatively, $x_{i,k} = a_k\widetilde{x}_{i,k} + b_k$) with a_k, b_k chosen such that

$$\sum_{i=1}^{n} \widetilde{x}_{i,k} = 0, \qquad \frac{1}{n}\sum_{i=1}^{n} \widetilde{x}_{i,k}^2 = 1. \tag{1.1}$$

The values a_k, b_k can in fact be analytically computed, see Exercise 1.2 below.

- *whitening:* this corresponds to a global normalization of the data, *i.e.* applying an affine transformation so that the data is centered and with a unit empirical covariance. The empirical covariance of the data inputs is

$$\Sigma_n = \frac{1}{n}\sum_{i=1}^{n} (x_i - \overline{x}_n)^\top (x_i - \overline{x}_n) \in \mathbb{R}^{d\times d}. \tag{1.2}$$

Note that the matrix Σ_n is positive symmetric. Assuming moreover that it is definite, one can consider $\Sigma_n^{-1/2}$ by spectral calculus, and introduce

$$\widetilde{x}_i = (x_i - \overline{x}_n)\, \Sigma_n^{-1/2}. \tag{1.3}$$

A simple computation shows that the transformed data set $\{\widetilde{x}_1, \dots, \widetilde{x}_n\}$ is by construction centered and with unit covariance; see Exercise 1.3. The original data inputs can be recovered by undoing the transformation as $x_i = \overline{x}_n + \widetilde{x}_i \Sigma_n^{1/2}$.

- *scaling:* this corresponds to a linear transformation mapping the data to the interval $[0, 1]$, with minimal value 0 and maximal value 1 in each column. More precisely,

$$\widetilde{x}_{i,k} = \frac{x_{i,k} - m_k}{M_k - m_k}, \qquad m_k = \min_{1\leqslant j\leqslant n} x_{j,k}, \quad M_k = \max_{1\leqslant j\leqslant n} x_{j,k}.$$

In all cases, when the machine learning method is trained on the transformed data set $\{\widetilde{x}_1, \dots, \widetilde{x}_n\}$, some preprocessing is necessary to make predictions for a new (test) data point x', as one first needs to transform it the same way the data points were transformed. When normalization is considered, this means that the prediction is performed on $\widetilde{x}'$ whose components are $\widetilde{x}'_k = (x' - b_k)/a_k$; when whitening is applied, prediction is performed on $\widetilde{x}' = \Sigma_n^{-1/2}(x' - \overline{x}_n)$; for scaling, prediction is done on $\widetilde{x}' = (x' - m_k)/(M_k - m_k)$. Let us emphasize that, in all cases, the parameters of the transformation are those computed on the training data set.

Exercise 1.2 (Standardization). *Prove that the values a_k, b_k which allow to satisfy* (1.1) *are the empirical standard deviation and the empirical average:*

$$a_k = \sqrt{\frac{1}{n}\sum_{i=1}^{n} (x_{i,k} - \overline{x}_n)^2}, \qquad b_k = \overline{x}_{n,k} = \frac{1}{n}\sum_{i=1}^{n} x_{i,k}.$$

Exercise 1.3 (Whitening). *Prove that the transformed data points* (1.3) *are such that*

$$\frac{1}{n}\sum_{i=1}^{n} \widetilde{x}_i = 0 \in \mathbb{R}^d, \qquad \frac{1}{n}\sum_{i=1}^{n} \widetilde{x}_i^\top \widetilde{x}_i = \mathrm{Id}_d.$$

1.2 Supervised learning

We make precise the mathematical framework behind the subclass of machine learning problems corresponding to supervised learning, as this will be the core topic considered in these lectures. The presentation is based on [4, Sections 2.1 and 2.2] and [40, Section 2.4.2].

1.2.1 Mathematical framework

Given a dataset of n elements $\mathcal{D} = \{(x_1, y_1), \dots, (x_n, y_n)\} \subset \mathcal{X} \times \mathcal{Y}$ (couples of inputs/outputs or features/labels), the aim of supervised learning is to predict the output or label $y \in \mathcal{Y}$ on an unseen data point $x \in \mathcal{X}$ ("test data"). Equivalently, this amounts to learning a mapping $\mathcal{X} \to \mathcal{Y}$.

In many practical cases, $\mathcal{X}$ is (isomorphic to) $\mathbb{R}^d$, either explicitly or implicitly (for instance when using kernels, see Section 5.5.2 and Chapter 6). Typical examples include physical measurements of various quantities at different locations (e.g. pressure, temperature, wind speed, humidity, etc. at various places in France for weather forecasting), or pixel values for images.

The label or output space $\mathcal{Y}$ depends on the type of problem which is considered. For classification (see Section 1.2.1.1) it is a discrete space: $\mathcal{Y} = \{0, 1\}$ or $\{-1, 1\}$ for binary classification, and $\mathcal{Y} = \{1, \dots, K\}$ when there are K classes. For regression problems, $\mathcal{Y} = \mathbb{R}$ or a higher dimensional space $\mathbb{R}^K$.

The measure of performance, *i.e.* the quality of the mapping $f : \mathcal{X} \to \mathcal{Y}$ which is learnt, depends on the task (classification vs. regression). Let us however warn the reader that the criterion for performance is in practice not always defined as precisely as in these lecture notes. Moreover, it can be the case that the output y cannot be a deterministic function of the input x, for instance when there is some measurement noise, or when there are extra features which are unobserved. A typical example would be the determination of the gender of a person depending on the height, weight and shoe size. In any case, the function f can be quite complicated, and nonlinear.

There are two (related) key challenges, which we will constantly encounter, when trying to learn the function $f : \mathcal{X} \to \mathcal{Y}$:

- the number of data points n which are observed can be rather small in view of the complexity of the function to learn. This requires to carefully assess the interpolation and extrapolation capabilities of the mapping f. Typically, interpolation is much easier than extrapolation: loosely speaking, prediction at new inputs in between data points is more reliable than predictions in regions not covered by data points.
- the input space can be of very large dimension d (the so-called "curse of dimensionality"). This raises scalability issues for the algorithms, and also limits the capacity of the model to make correct predictions on new data – the so-called generalization ability.

Formalizing the learning framework. We consider the paradigm of statistical learning, where elements in the dataset are assumed to be identically and independently distributed (i.i.d.) according to some unknown probability measure $p_{\text{data}}(dx\, dy)$. Expectations with respect to the latter probability measure are denoted by $\mathbb{E}$. The mapping $\mathcal{X} \to \mathcal{Y}$ to be learned is parametrized by $\theta \in \Theta$, so that the model is sought in the parametric class of functions $\{f_\theta, \theta \in \Theta\}$. The ideal learning problem can then be formulated as the following minimization of the expected risk:

$$\min_{\theta \in \Theta} \mathcal{R}(\theta), \qquad \mathcal{R}(\theta) = \mathbb{E}\left[\ell\left(y, f_\theta(x)\right)\right], \tag{1.4}$$

for some loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ (typically having nonnegative values). In practice, only a finite number of data points are available through a dataset $\mathcal{D} = \{(x_1, y_1), \dots, (x_n, y_n)\} \subset (\mathcal{X} \times \mathcal{Y})^n$. In this context, a learning algorithm is formally an application which associates a function $f \in \mathscr{F}(\mathcal{X}, \mathcal{Y})$ (the set of measurable functions from $\mathcal{X}$ to $\mathcal{Y}$) to a dataset $\mathcal{D}$ of size n. This is often done in practice by relying on the so-called empirical risk minimization (see Definition 1.3 for a more formal presentation), where the expectation in (1.4) is approximated by a finite sum. More precisely, the algorithm returns $f_{\widehat{\theta}_n}$ with

$$\widehat{\theta}_n \in \operatorname*{argmin}_{\theta \in \Theta} \widehat{\mathcal{R}}_n(\theta), \qquad \widehat{\mathcal{R}}_n(\theta) = \frac{1}{n} \sum_{i=1}^{n} \ell\left(y_i, f_\theta(x_i)\right).$$

Exercise 1.4. *For θ fixed, give sufficient conditions on ℓ and p_{data} ensuring that $\widehat{\mathcal{R}}_n(\theta)$ converges to $\mathcal{R}(\theta)$ as $n \to +\infty$, and make precise in which sense convergence holds.*

Let us however already emphasize that the goal of machine learning is not to minimize the loss on the training data, but on future data not yet seen, to guarantee some good prediction ability. This generalization performance can be discussed within the framework of decision theory, see Section 1.2.2 below and Chapter 10.

1.2.1.1 Classification

In classification problems, the aim is to predict a label $y \in \mathcal{Y}$ in a discrete set $\mathcal{Y}$. For multiclass classification, $\mathcal{Y} = \{1, \dots, K\}$, with K the number of classes considered in the classification problem. For binary classification, $\mathcal{Y} = \{0, 1\}$ for certain algorithms (for instance logistic regression, see Chapter 3), while $\mathcal{Y} = \{-1, 1\}$ for others (for instance support vector machines, see Chapter 6). One example besides the MNIST dataset, and which we will consider for the hands-on, is the Iris data set (see for instance [41, Section 1.2.1.1]), for which there are $d = 4$ features and $K = 3$ classes.

There are various choices for the loss functions. A natural one is $\ell(z, y) = \mathbf{1}_{y \neq z}$, for which the associated risk function $\mathcal{R}(\theta) = \mathbb{P}(y \neq f_\theta(x))$ in (1.4) is called the misclassification error. However, as discussed in Section 3.1, the corresponding risk function is difficult to optimize. The use of gradient-like optimization algorithms, as those presented in Chapter 4, suggests to resort to smooth surrogate loss functions. Appropriate choices for these surrogate functions are discussed in Section 3.1.

Let us conclude this section by presenting a typical way of predicting classes in multiclass classification, relying on the softmax function, which takes as argument $(a_1, \dots, a_K) \in \mathbb{R}^K$ and returns

$$S_K(a_1, \dots, a_K) = \left(\frac{\mathrm{e}^{a_1}}{\sum_{k=1}^K \mathrm{e}^{a_k}}, \frac{\mathrm{e}^{a_2}}{\sum_{k=1}^K \mathrm{e}^{a_k}}, \dots, \frac{\mathrm{e}^{a_K}}{\sum_{k=1}^K \mathrm{e}^{a_k}} \right) \in [0,1]^K.$$

Note that the denominators in the arguments of S_K ensure that the components of S_K sum to 1, so that S_K returns a discrete probability. The choice of an exponential function is convenient to transform real numbers into positive ones.

The prediction is based on some function $f_\theta : \mathcal{X} \to \mathbb{R}^K$, so that the vector $S_K(f_\theta(x))$ gives, for a new input $x \in \mathcal{X}$, the probabilities of the label y to be in one of the classes; more precisely, the probability that $y = k$ is the k-th component of $S_K(f_\theta(x))$, denoted by $S_K(f_\theta(x))_k$. A natural classification rule is therefore

$$y \in \operatorname*{argmax}_{1 \leqslant k \leqslant K} \{S_K(f_\theta(x))_k\}.$$

In fact, the choice $f_\theta(x) = W^\top x + b$, with $\theta = (W, b)$ where $W \in \mathbb{R}^{d \times K}$ and $b \in \mathbb{R}^K$ (we interpret here x as a column vector), corresponds to multinomial logistic regression (see Section 3.2.4). The part W of the parameter θ is called weight, while the part b is called bias (let us however emphasize that this has nothing to do with the notion of bias for estimators, as encountered in statistics).

Remark 1.1. *For $K = 2$ classes (binary classification), a simple computation shows that*

$$S_2(a_1, a_2) = \left(\frac{\mathrm{e}^{a_1}}{\mathrm{e}^{a_1} + \mathrm{e}^{a_2}}, 1 - \frac{\mathrm{e}^{a_1}}{\mathrm{e}^{a_1} + \mathrm{e}^{a_2}} \right) = (\sigma(a_1 - a_2), 1 - \sigma(a_1 - a_2)),$$

where we introduced the sigmoid function

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

This shows that it suffices to learn $a_1 - a_2$, and not a_1, a_2 separately. The model can then be parametrized using only $\sigma(f_\theta(x))$ with $f_\theta(x) = W^\top x + b \in \mathbb{R}$ where $W \in \mathbb{R}^K$ and $b \in \mathbb{R}$.

1.2.1.2 Regression

Regression problems correspond to situations where the space $\mathcal{Y}$ is continuous, typically $\mathcal{Y} = \mathbb{R}^K$. The most famous choice of loss function is the square loss $\ell(y, z) = \|y - z\|_2^2$ (with $\|\cdot\|_2$ the standard Euclidean norm on $\mathbb{R}^K$), for which the associated empirical risk minimization reads

$$\widehat{\mathcal{R}}_n(\theta) = \frac{1}{n} \sum_{i=1}^n \|y_i - f_\theta(x_i)\|_2^2. \tag{1.5}$$

Other loss functions can be considered, for instance the mean absolute loss (with $y = (y_1, \dots, y_K)$)

$$\ell(y, z) = \|y - z\|_1 = \sum_{k=1}^{K} |y_k - z_k|,$$

or generalizations of the latter loss. The interest of these functions is that large deviations are less penalized, which is important to limit the impact of outliers in the dataset and make regression more robust.

Exercise 1.5. *Prove that* (1.5) *is proportional to the log-likelihood of the sample $\mathcal{D}$ in a statistical model where y is distributed according to a Gaussian distribution centered on $f_\theta(x)$ with given variance $\sigma^2 > 0$.*

Typical instances of models used in regression are:

- linear regression $f_\theta(x) = b + \sum_{j=1}^{d} w_j x_j$ for $x = (x_1, \dots, x_d) \in \mathbb{R}^d$, with $\theta = (b, w_1, \dots, w_d) \in \mathbb{R}^{d+1}$;
- polynomial regression, where some data point $x \in \mathbb{R}$ is first featurized as $\phi(x) = (1, x, \dots, x^p)$ (with p the degree of the polynomial), and then a prediction is made as $f_\theta(x) = \theta^\top \phi(x)$ for some vector $\theta \in \mathbb{R}^{p+1}$. Note that this regression is still linear in the parameter θ to estimate;
- regression using (deep) neural networks, presented in Chapter 8, and various other models generically relies on the prediction function $f_\theta(x, \theta) = w^\top \phi(x, v)$, where the parameters $\theta = (w, v)$ are decomposed in two parts: a part w which is used to linearly combine the output of a (nonlinear) featurization function, and a part v which characterizes the featurization function $\phi(x, v)$. For neural networks for instance, this featurization function is of the form

  $$\phi(x, v) = \rho_L \left(\rho_{L-1} \left(\rho_{L-2}(\dots \rho_1(x, W_1, b_1)), W_{L-1}, b_{L-1} \right), W_L, b_L \right),$$

  where L is the number of layers, and $v = (W_1, b_1, \dots, W_L, b_L)$ gathers the parameters to be learnt at each layer.

1.2.2 Decision theory and statistical learning

Decision theory (see Section 1.2.2.1) is a mathematical framework which allows to construct the best predictor when the distribution of the data p_{data} is known. However, as mentioned earlier, this distribution is typically not known in practice. One should therefore see the predictor provided by decision theory as some ideal choice which provides some baseline lower bound to the loss/risk which can be actually achieved. Statistical learning (see Section 1.2.2.2) aims at obtaining quantitative bounds/guarantees on how much the model which is learnt departs from the optimal choice provided by decision theory.

1.2.2.1 Decision theory

Definition 1.1 (Expected risk). *Given a measurable function $f : \mathcal{X} \to \mathcal{Y}$, a loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ and a probability distribution p_{data} on $\mathcal{X} \times \mathcal{Y}$, the expected risk of a prediction function is*

$$\mathcal{R}(f) = \mathbb{E}\left[\ell(Y, f(X))\right] = \int_{\mathcal{X} \times \mathcal{Y}} \ell(y, f(x))\, p_{\text{data}}(dx\, dy). \tag{1.6}$$

Note that, in this definition, we distinguished the random variables $(X, Y) \sim p_{\text{data}}$, and their realizations $(x, y) \in \mathcal{X} \times \mathcal{Y}$. In the remainder of these lectures notes (and as already done in (1.4) for instance), we will often denote the random variables with small letters. It should be clear from the context whether (x, y) denotes a random variable or one of its realizations.

Example 1.1. *For classification, the loss function $\ell(y,z) = \mathbf{1}_{y\neq z}$ leads to the expected risk $\mathcal{R}(f) = \mathbb{P}(f(x) \neq y)$, which is the error rate; alternatively, $\mathcal{R}(f) = 1 - \mathbb{P}(f(x) = y)$, where $\mathbb{P}(f(x) = y)$ is the* accuracy.

Example 1.2. *For regression, a common choice is $\ell(y,z) = \|y - z\|_2^2$, in which case the expected risk is the mean square error between the random variables $f(X)$ and Y.*

Remark 1.2. *For some methods, the prediction provided by the function f is itself random, and depends on extra random variables (think of a binary classification example where one would guess the label at random). In this case, there is no simple outcome $f(x)$, but rather a distribution of possible values for $f(x)$. The definition* (1.6) *then still makes sense, upon further taking the expectation over the extra randomness.*

In order to find the function $f^\star$ which minimizes the expected risk, we condition the values of the expected risk on the realizations of the input[2] (distinguishing here for clarity the random variable X and its realizations):

$$\mathcal{R}(f) = \mathbb{E}\left[\mathbb{E}\left(\ell(Y, f(X)) \,\middle|\, X\right)\right] = \mathbb{E}\left[r(f(X) \,|\, X)\right] = \int_{\mathcal{X}} r\left(f(x') \,|\, x'\right) \, p_{\mathrm{data}}(dx'),$$

where

$$r(z \,|\, x') = \mathbb{E}\left(\ell(Y,z) \,\middle|\, X = x'\right)$$

is the conditional expectation of the loss. This reformulation makes it clear that, for any realization x' of X, the associated prediction $f(x')$ should minimize $r(\cdot \,|\, x')$. This observation is summarized in the following result.

Proposition 1.1. *The expected risk is minimized at a Bayes predictor $f^\star : \mathcal{X} \to \mathcal{Y}$ defined as*[3]

$$f^\star(x') \in \operatorname*{argmin}_{z\in\mathcal{Y}} r(z \,|\, x'). \tag{1.7}$$

The Bayes risk is the risk of all Bayes predictors:

$$\mathcal{R}^\star = \mathbb{E}\left[\ell\left(Y, f^\star(X)\right)\right] = \mathbb{E}\left[\inf_{z\in\mathcal{Y}} \mathbb{E}\left(\ell(Y,z) \,\middle|\, X\right)\right].$$

Let us emphasize that the Bayes predictor defined in (1.7) may not be unique as the set of minimizers for x' given may contain several elements. However, all Bayes predictors lead to the same Bayes risk. The Bayes risk is usually positive, unless the dependence between X and Y is deterministic.

Definition 1.2 (Excess risk). *The excess risk of a measurable function $f : \mathcal{X} \to \mathcal{Y}$ is $\mathcal{R}(f) - \mathcal{R}^\star$.*

Machine learning would be trivial if a Bayes predictor $f^\star$ was known! This would however require the knowledge of the distribution of outcomes y conditionally on the value of the input x, which is typically not the case as the distribution of the data is unknown.

Exercise 1.6. *Compute the Bayes predictor and the Bayes risk for the loss function considered in Examples 1.1 (with $\mathcal{Y} = \{0,1\}$) and 1.2 (for one-dimensional outputs,* i.e. *$\mathcal{Y} = \mathbb{R}$).*

Exercise 1.7. *For binary classification on $\mathcal{Y} = \{-1,1\}$, compute the Bayes predictor and the Bayes risk for the loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ defined by its values $\ell(-1,1) = c_-$ (cost of false positive), $\ell(1,-1) = c_+$ (cost of false negative), and $\ell(-1,-1) = \ell(1,1) = 0$. Interpret the so-obtained Bayes predictor.*

[2] Here and in the sequel, we abuse notation and denote p_{data} and its marginal distributions by the same symbol.

[3] We assume here that there is a well defined minimizer; otherwise the argmin should be replaced by an arginf.

Exercise 1.8. *In a binary classification context on $\mathcal{Y} = \{0,1\}$ with $\ell(0,0) = \ell(1,1) = 0$ and $\ell(1,0) = \ell(0,1)$, compute the so-called "chance level", which is the risk associated with the random prediction function $f(x) = \mathbf{1}_{U \leqslant 1/2}$ with $U \sim \mathcal{U}[0,1]$ a random variable uniformly distributed on $[0,1]$, independent of the sample points.*

Exercise 1.9 (Random predictions). *We consider a binary classification problem with $\mathcal{Y} = \{0,1\}$, and two random prediction functions: $g(x)$, which returns 1 with probability $\eta(x) = \mathbb{P}(y = 1|x)$, and 0 with probability $1-\eta(x)$; and $h(x)$, which returns 0 and 1 with probability $1/2$ for each of them. Recall that, for a given function f, the risk is defined as*

$$\mathcal{R}(f) = \mathbb{E}\left[\mathbf{1}_{y \neq f(x)}\right].$$

(a) Show that $\mathbb{P}(y \neq g(x)|x) = 2\eta(x)(1-\eta(x))$.
(b) Compute the excess risk for g.
(c) When is g achieving the Bayes risk?
(d) Compute the excess risk for h. When is h achieving the Bayes risk?

Exercise 1.10 (Bayes predictor for robust regression). *Robust regression amounts to considering the loss function $\ell(y,z) = |y-z|$ for regression problems with $\mathcal{Y} = \mathbb{R}$, instead of the square loss $(y-z)^2$. The interest of the loss function based on absolute values and not their square is that outliers have a smaller impact on the prediction.*

(a) Consider a random variable Z which admits a positive continuous density $\rho(z)$ with respect to the Lebesgue measure, and is integrable. Prove that

$$m = \underset{c \in \mathbb{R}}{argmin}\, \mathbb{E}|Z-c|$$

is the median of Z, namely the (here unique) value such that $\mathbb{P}(Z \leqslant m) = \mathbb{P}(Z \geqslant m)$.
(b) Assume for simplicity that the (unknown) distribution $p_{\mathrm{data}}(dx\,dy)$ of the data points has a positive continuous density with respect to the Lebesgue measure $dx\,dy$. Compute the Bayes predictor and the associated Bayes risk.
(c) We no longer assume that the (unknown) distribution $p_{\mathrm{data}}(dx\,dy)$ of the data points has a positive continuous density with respect to the Lebesgue measure $dx\,dy$. For a general random variable, the median is defined as a value $m \in \mathbb{R}$ such that

$$\mathbb{P}(Z \leqslant m) \geqslant \frac{1}{2}, \qquad \mathbb{P}(Z \geqslant m) \geqslant \frac{1}{2}.$$

Note that the inequalities can be strict (in particular when there is a Dirac mass at m). Compute the Bayes predictor and the associated Bayes risk.

1.2.2.2 Statistical learning

Decision theory tells us what to do if the distribution p_{data} of the data is known, in which case optimal performance can be achieved. Now, in practice, this distribution is unknown. Statistical learning is a framework to obtain bounds or guarantees quantifying how much the learned model departs from the (unknown) optimal choice. A key concept in this endeavor is empirical risk.

Definition 1.3 (Empirical risk). *Given a measurable function $f : \mathcal{X} \to \mathcal{Y}$, a loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ and a data set $\{(x_1,y_1),\dots,(x_n,y_n)\}$, the empirical risk of a prediction function is*

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

The empirical risk can in principle be successfully minimized: upon considering a sufficiently flexible class of predictor functions, one can achieve zero training loss. A successful empirical risk minimization does however not imply successful predictions, as the prediction function may fit too closely the training data and learn irrelevant features. This is the celebrated issue of *overfitting*, which we will come back to in Chapter 2. The discrepancy between the expected risk and the empirical risk is measured by the generalization gap

$$\mathcal{R}(f) - \widehat{\mathcal{R}}_n(f).$$

The population risk $\mathcal{R}(f)$ is approximated in practice by the test risk, which corresponds to some empirical risk based on data points which have not been seen during training, and can therefore be considered as new data points sampled from p_{data}.

Let us make precise the various steps involved in training a model and then assessing its quality. Given a dataset $\mathcal{D}$, we first separate it into a training set $\mathcal{D}_{\text{train}}$ and a test set $\mathcal{D}_{\text{test}}$. A typical ratio is to keep 80% of the data points in the training set, and 20% in the test set. This allows to balance the need to have many data points in order for the training procedure to see as many instances as possible, while still keeping enough data points in order to have a sufficiently reliable estimation of the expected risk. The next step is to find a predictor function $\widehat{f}_n$ which minimizes the empirical risk over the data set $\mathcal{D}_{\text{train}}$, among a prescribed class $\mathcal{F}$ of functions $\mathcal{X} \to \mathcal{Y}$:

$$\widehat{f}_n \in \operatorname*{argmin}_{f \in \mathcal{F}} \left\{ \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x_i, y_i) \in \mathcal{D}_{\text{train}}} \ell(y_i, f(x_i)) \right\}.$$

The final step is to approximate the expected risk as

$$\frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{(x_j, y_j) \in \mathcal{D}_{\text{test}}} \ell\left(y_j, \widehat{f}_n(x_j)\right).$$

In fact, we will see later on that one needs to further divide the training set in an actual training set, and a validation set, used to fix the hyperparameters of the method. The important point here is that the test set is never used to determine the predictor function $\widehat{f}_n$ which minimizes the empirical risk. It is only used at the very end, to assess the quality of the prediction.

1.2.2.3 Learning from data

Let us conclude this introduction to machine learning by mentioning three important ways to obtain a (good) prediction from the data at hand rather than from the full knowledge of (the unknown distribution) p_{data}:

- in local averaging methods, one relies on some interpolation procedure with the data at hand to provide predictions for new data points. One famous example is the K-nearest neighbor method, presented in Section 1.3;
- the most common situation we will consider is to obtain prediction functions from the minimization of the empirical risk;
- we will only briefly consider other ways to find predictors, in particular using the boosting paradigm (see Section 7.3). One could also rely on probabilistic methods, which we however do not consider in these lecture notes.

1.3 Local averaging methods

The material in this section is based on [41, Chapter 16] and [4, Chapter 6], as well as [6, Chapter 14] and [50, Chapter 19]. The main idea behind local averaging methods is that "things that look alike must be alike", *i.e.* close-by points should have similar labels. A successful prediction relies in this context on the smoothness of the data, in the sense that the labels/values y should depend smoothly on the inputs x (if the labels/values y were truly random, there would be no way to learn meaningful patterns in the data and generalize this knowledge to new data points...).

1.3.1 Presentation of the K-nearest neighbor algorithm

Nearest neighbor methods are based on a majority vote among the closest neighbors for classification, and some form of averaging over the closest neighbors for regression. They provide a useful starting method in machine learning since they are easy to code and to understand. In essence, they amount to memorizing a training set, and predicting the label of a new instance x' based on the labels of the closest points x_i in the training set.

To make this precise, we first need to choose some metric $\mathtt{d} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$ to measure distances between data points. This is a key element in the method. A typical choice is the Euclidean distance, which however behaves badly in high dimensions (where all points are far away from each other). A good practice to alleviate these issues is to perform some rescaling of the data (see for instance [25, Section 13.3]; this is motivated by the fact that the data can be in different units or heterogeneous in its scale), for instance through standardization or whitening (recall the discussion at the end of Section 1.1.4), or to rely on the Mahalanobis distance

$$\mathtt{d}(x, x') = \sqrt{(x - x')^\top \Sigma_n^{-1} (x - x')},$$

where the covariance Σ_n of the data inputs is defined in (1.2) (in fact this corresponds to considering a simple Euclidean distance on the whitened data). Note that the distance function actually used can be considered as a hyperparameter of the method, to be set using some (cross) validation procedure.

Additionally, it may be beneficial to first perform some dimensionality reduction (using for instance principal component analysis, see Chapter 5, or its nonlinear generalization based on autoencoders, see Section 8.3) – both in order to reduce the computational cost of the method, and to possibly improve predictions since only the large scale features of the data are retained. This heuristic consideration is backed up by the theoretical estimates discussed around (1.18).

Algorithm 1.1. Fix a number of neighbors $K \geqslant 1$ and a training data set $\{(x_1, y_1), \dots, (x_n, y_n)\} \subset \mathcal{X} \times \mathcal{Y}$. For a new input $x' \in \mathcal{X}$, sort the data points by increasing distance[4]

$$\mathtt{d}\left(x_{i_1(x')}, x'\right) \leqslant \mathtt{d}\left(x_{i_2(x')}, x'\right) \leqslant \dots \leqslant \mathtt{d}\left(x_{i_n(x')}, x'\right), \tag{1.8}$$

where $\{i_1(x'), \dots, i_n(x')\} = \{1, \dots, n\}$. The prediction is then performed as

- (classification) y' is the majority of labels among the labels of the K-closest neighbors $\{y_{i_1(x')}, \dots, y_{i_K(x')}\}$;
- (regression) y' is the average of the values for the K-closest neighbors: $y' = \dfrac{1}{K} \displaystyle\sum_{k=1}^{K} y_{i_k(x')}$.

The value $K = 1$ corresponds to predictions based on a Voronoi tesselation of the input space. This means in practice that a new input inherits the label or value of the closest input in the data set. Intuitively, this seems however not very robust a choice, as a small variation in the inputs can lead to abrupt changes in the predictions. As we will see in Section 1.3.3.3, the variance in the prediction is indeed large when K is small. On the other hand, when K is large, decision boundaries separating various values for the predictions will be smoother, and the predictors more robust. However, the predictions may be quite biased when K is large. Overall, this calls for some optimal choice of K based on a bias/variance trade-off. We will see both theoretical guidelines which give firm foundations to this choice (see Section 1.3.3), as well as a practical manner of fixing the value of K based on a key procedure in machine learning, cross validation (see Section 1.3.2).

Let us finally discuss the algorithmic cost of finding the closest neighbor. A priori, this scales as $\mathrm{O}(nd)$ (when computing the distance to all elements in the data set), but the cost can be reduced by clever algorithmic strategies based on a tree search (see the discussions in [41, Section 16.1.3] and [50, Section 19.3]). This also suggests that it makes sense to sparsify to some extent the training set and keep only the most relevant points.

[4] with some rule to break ties, for instance $i_k(x') \leqslant i_{k+1}(x')$ (*i.e.* always choose the smallest index), or choosing at random among equidistant points.

1.3.2 Choosing K by (cross) validation

We give here a brief introduction to (cross) validation, referring to Section 10.2 for complements. We distinguish two cases, depending on whether the training data set is large or not.

1.3.2.1 Large training data sets

When enough data is available, the training data set $\mathcal{D}$ is decomposed into an actual training data set $\mathcal{D}_{\text{train}}$ and a validation set $\mathcal{D}_{\text{val}}$. The validation set should be large enough so that the population risk can be well approximated by computing the empirical risk over it; while most of the data should still be used for training. A common rule of thumb is to devote 20% of the initial data $\mathcal{D}$ for validation, and keep 80% for training. The assignement to $\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{val}}$ is done at random, typically by shuffling the data.

Once these sets are obtained, the validation procedure amounts to iterating over the possible values of the hyperparameters of the method, here the number of neighbors $K \geqslant 1$, training the model for these hyperparameters (which need not be done for K-nearest neighbors as no training is required in the method), and then computing the empirical risk on the validation set:

$$\widehat{\mathcal{R}}_n^{\text{val}}(K) = \frac{1}{|\mathcal{D}_{\text{val}}|} \sum_{(x_j, y_j) \in \mathcal{D}_{\text{val}}} \ell\left(y_j, \widehat{y}_j\right),$$

where $n = |\mathcal{D}_{\text{train}}|$ is here the size of the actual training set $\mathcal{D}_{\text{train}}$, and $\widehat{y}_j$ is the prediction for the input x_j of the method trained on $\mathcal{D}_{\text{train}}$. One then chooses the value of K for which $\widehat{\mathcal{R}}_n^{\text{val}}(K)$ is minimal.

Let us emphasize once again that the procedure crucially relies on $\mathcal{D}_{\text{train}} \cap \mathcal{D}_{\text{val}} = \emptyset$, *i.e.* no validation data is used in the training procedure. The statistical consistency of validation is discussed in Section 10.2.

1.3.2.2 Small training data sets

When the data set $\mathcal{D}$ is small, there are two obstructions to the validation procedure described in Section 1.3.2.1: first, the validation set would be small, and so the approximation of the population risk by the empirical risk computed on the validation set would be unreliable; second, one would prefer not to loose data on validation, and use as much data as possible for training. Cross-validation is a procedure to address these two issues.

Concretely, the data set $\mathcal{D}$ is first shuffled, then decomposed into L folds of same sizes $|\mathcal{D}|/L$ (assuming that this number is an integer). Among these folds, one is kept for validation, while the remaining $L-1$ ones are used for training. The empirical risk is next computed with the data of the fold which was chosen for validation. This is then repeated for all possible L choices for the validation set, the approximation of the expected risk being obtained as the average of the empirical risks computed for each validation fold. More precisely, denoting by $\mathcal{D}_k$ for $1 \leqslant k \leqslant L$ the various folds and by $n = |\mathcal{D}|$ the size of the initial data set, the expected risk is approximated as

$$\widehat{\mathcal{R}}_{n,L}^{\text{CV}} = \frac{1}{L} \sum_{k=1}^{L} \widehat{\mathcal{R}}\left(\mathcal{D} \setminus \mathcal{D}_k, \mathcal{D}_k\right), \tag{1.9}$$

where

$$\widehat{\mathcal{R}}\left(\mathcal{D}_{\text{train}}, \mathcal{D}_{\text{val}}\right) = \frac{1}{|\mathcal{D}_{\text{val}}|} \sum_{(x_j, y_j) \in \mathcal{D}_{\text{val}}} \ell\left(y_j, \widehat{y}_j^{\mathcal{D}_{\text{train}}}\right),$$

with $\widehat{y}_j^{\mathcal{D}_{\text{train}}}$ the prediction for the input x_j and the method trained on $\mathcal{D}_{\text{train}}$.

Usual values for L are in the range $5 - 10$. Another option is to consider $L = n$, which corresponds to the so-called "leave-one-out" cross-validation procedure. Similarly to the validation procedure described in Section 1.3.2.1, the cross-validation risk (1.9) is computed for various values of the hyperparameters of the method (here, the number K of nearest neighbors), choosing in the end the hyperparameter leading to the smallest cross-validation risk.

1.3.3 Theoretical analysis of the K-nearest neighbor algorithm

Recall that the aim of machine learning, as discussed in Section 1.2.2, is to estimate the Bayes predictor $f^\star$, knowing only training data points $\mathcal{D}_{\text{train}} = \{(x_i, y_i)_{1 \leqslant i \leqslant n}\}$, in order to minimize the excess risk $\mathcal{R}(f) - \mathcal{R}^\star = \mathbb{E}[\ell(Y, f(X))] - \mathcal{R}^\star$. We first describe in Section 1.3.3.1 the general framework allowing to study the theoretical properties of a broader class of estimators known as local methods; then perform a consistency analysis for this class of estimators in Section 1.3.3.2, finally specifying to K-nearest neighbors in Section 1.3.3.3.

1.3.3.1 Local methods

Local methods, such as K-nearest neighbors, approximate $f^\star$ without any form of optimization. Conceptually, this is done by approximating the conditional distribution $p_{\text{data}}(dy \,|\, x)$ of labels y given inputs x by some distribution $\widehat{p}(dy \,|\, x)$, so that the minimization problem (1.7)

$$f^\star(x) \in \operatorname*{argmin}_{z \in \mathcal{Y}} \int_{\mathcal{Y}} \ell(y, z)\, p_{\text{data}}(dy \,|\, x)$$

is approximated by

$$\widehat{f}(x) \in \operatorname*{argmin}_{z \in \mathcal{Y}} \int_{\mathcal{Y}} \ell(y, z)\, \widehat{p}(dy \,|\, x).$$

This corresponds to some form of plug-in estimator, as encountered in Statistics. Let us illustrate this general strategy for two prominent examples:

- for multi-class classification over C classes, with the 0-1 loss $\ell(y, z) = \mathbf{1}_{y \neq z}$,

$$\widehat{f}(x) \in \operatorname*{argmin}_{z \in \{1, \dots, C\}} \left\{ \sum_{k=1}^{C} \mathbf{1}_{z \neq k}\, \widehat{p}(y = k \,|\, x) \right\} = \operatorname*{argmin}_{z \in \{1, \dots, C\}} \left\{ 1 - \sum_{k=1}^{C} \mathbf{1}_{z = k}\, \widehat{p}(y = k \,|\, x) \right\},$$

  so that

$$\widehat{f}(x) \in \operatorname*{argmax}_{k \in \{1, \dots, C\}} \widehat{p}(y = k \,|\, x); \tag{1.10}$$

- for regression with the square loss, the predictor is the conditional expectation associated with $\widehat{p}$:

$$\widehat{f}(x) = \int_{\mathcal{Y}} y\, \widehat{p}(dy \,|\, x). \tag{1.11}$$

We further consider here linear estimators, for which the approximation to the conditional distribution $p_{\text{data}}(dy \,|\, x)$ is a convex combination of Dirac masses at y_i, namely

$$\widehat{p}(dy \,|\, x) = \sum_{i=1}^{n} \widehat{w}_i(x) \delta_{y_i}(dy),$$

where the weights depend only on x and $x_1, \dots, x_n$ (and not on the labels) and are such that

$$\forall x \in \mathcal{X}, \qquad \forall i \in \{1, \dots, n\}, \quad \widehat{w}_i(x) \geqslant 0, \qquad \sum_{i=1}^{n} \widehat{w}_i(x) = 1.$$

In addition to K-nearest neighbors, partition estimators and kernel density estimators (known as Nadaraya–Watson estimators) also belong to this class; see [4, Section 6.2]. In this setting, the estimators (1.10) and (1.11) respectively read

$$\widehat{f}(x) \in \operatorname*{argmax}_{k \in \{1, \dots, C\}} \left\{ \sum_{i=1}^{n} \widehat{w}_i(x) \mathbf{1}_{y_i = k} \right\},$$

and

$$\widehat{f}(x) = \sum_{i=1}^{n} \widehat{w}_i(x) y_i.$$

The K-nearest neighbors methods corresponds to the choice

$$\widehat{w}_j(x) = \begin{cases} \dfrac{1}{K} \text{ if } j \in \{i_1(x), \dots, i_K(x)\}, \\ 0 \text{ otherwise.} \end{cases} \tag{1.12}$$

1.3.3.2 Consistency analysis for local methods

We discuss the consistency of local methods for regression; see the introduction of [4, Section 6.3] for comments and references on the classification case. Consistency is understood here as the excess risk converging to 0 in some sense as the number of training data points goes to infinity. This notion of consistency (rather than, say, quantifying the convergence of predictors to the Bayes predictor) is adapted to machine learning problems as the aim is to devise good predictors, as measured by the fact that the prediction error is small.

We make the following assumptions, to simplify the presentation.

Assumption 1.1. *The noise is bounded: There exists $\sigma \in \mathbb{R}_+$ such that $|Y - \mathbb{E}(Y \mid X)| \leqslant \sigma$ almost surely.*

Assumption 1.2. *The Bayes predictor $f^\star(x) = \mathbb{E}[Y \mid X = x]$ is B-Lipschitz for some continuous distance function $\mathtt{d} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$:*

$$\forall (x_1, x_2) \in \mathcal{X}^2, \qquad |f^\star(x_1) - f^\star(x_2)| \leqslant B\mathtt{d}(x_1, x_2).$$

When these assumptions hold, one can derive the following bound on the excess risk – and in fact the average of the excess risk with respect to realizations of the training data. Let us emphasize that we consider here only the average excess risk for simplicity. In some frameworks, such as the probably approximately correct (PAC) setting (see Chapter 10) one may prefer bounds which hold stronger than in average, but this requires more work/assumptions on the distribution of the data.

Proposition 1.2. *When Assumptions 1.1 and 1.2 hold,*

$$\mathbb{E}_{\mathcal{D}}\left[\mathcal{R}(\widehat{f})\right] - \mathcal{R}^\star \leqslant \sigma^2 \sum_{i=1}^{n} \int_{\mathcal{X}} \mathbb{E}_{\mathcal{D}}\left(\widehat{w}_i(x')^2\right) p_{\mathrm{data}}(dx') + B^2 \int_{\mathcal{X}} \mathbb{E}_{\mathcal{D}}\left[\sum_{i=1}^{n} \widehat{w}_i(x') \mathtt{d}(x_i, x')^2\right] p_{\mathrm{data}}(dx'), \tag{1.13}$$

where $\mathbb{E}_{\mathcal{D}}$ is an expectation with respect to the training set.

The first term on the previous right hand side, which arises from the inherent label noise (and would therefore be present even for the optimal prediction function $f^\star$) involves the factor

$$\mathbb{E}_{\mathcal{D}}\left(\widehat{w}_i(x')^2\right) = \mathbb{E}_{\mathcal{D}}\left[\left(\widehat{w}_i(x') - \frac{1}{n}\right)^2\right] + \frac{2}{n} - \frac{1}{n^2}$$

which measures the deviation to uniform weights. Note indeed that uniform weights minimize the latter quantity since, by a discrete Cauchy–Schwarz inequality,

$$n\left(\sum_{i=1}^{n} \widehat{w}_i(x')^2\right)^2 \geqslant \left(\sum_{i=1}^{n} \widehat{w}_i(x')\right)^2 = 1,$$

with equality if and only if $\widehat{w}_i(x') = 1/n$ for all $1 \leqslant i \leqslant n$. The expectation of the squares of the weights is therefore related to overfitting issues, as the variance of the weights is largest when all

the mass is put in one weight, for a data point which changes from a realization of the data set to another. The second term on the right hand side of (1.13), which depends on the regularity of the optimal prediction function $f^\star$, measures some bias in the prediction. It is smaller when $f^\star$ varies less, *i.e.* B is small. It is related to underfitting as, in the worst case scenario, all weights are the same and nothing is learned.

A careful inspection of the proof shows that it would be sufficient instead of Assumption 1.1 to assume that $\mathbb{E}\left[|Y - \mathbb{E}(Y \,|\, X)|^2\right] \leqslant \sigma^2$. Weaker conditions than Assumption 1.2 are discussed in [4, Section 6.4] (based on the notion of universal consistency).

Exercise 1.11. *The aim of this exercise is to prove Proposition 1.2. The first step is to decompose the prediction error as the sum of two contributions, one depending on the realization of the labels y_i (which therefore measures some variance; recalling the abuse of notation where y_i is a random variable) and one deterministic term for $x_1, \dots, x_n$ given (akin to some bias).*

(a) Show that

$$\widehat{f}(x) - f^\star(x) = \sum_{i=1}^n \widehat{w}_i(x)\,[y_i - \mathbb{E}(y_i \,|\, x_i)] + \sum_{i=1}^n \widehat{w}_i(x)\,[f^\star(x_i) - f^\star(x)]\,.$$

(b) Deduce that the square error, conditionally on the inputs $x_1, \dots, x_n$, can be bounded as

$$\mathbb{E}\left[\left|\widehat{f}(x) - f^\star(x)\right|^2 \middle| x_1, \dots, x_n\right] \leqslant \sigma^2 \sum_{i=1}^n \widehat{w}_i(x)^2 + B^2 \sum_{i=1}^n \widehat{w}_i(x)\mathtt{d}(x_i, x)^2.$$

(c) Prove that (1.13) *holds.*

It suffices at this stage to ensure that the two terms on the right hand side of (1.13) converge to 0 as $n \to +\infty$. Ideally, one would also like to make precise the convergence rate here, and to find guidelines to choose the hyperparameters of the method. This is where one needs to specify the analysis to the algorithm under consideration.

1.3.3.3 Consistency analysis for K-nearest neighbors

We make precise in this section how the two terms on the right hand side of (1.13) depend on various parameters, in particular the number of training points n, the number of neighbors K and the dimension d of the inputs.

The first remark is that the variance term is particularly simple for K-nearest neighbors. Indeed, in view of (1.12),

$$\forall x' \in \mathcal{X}, \qquad \sum_{i=1}^n \widehat{w}_i(x')^2 = \frac{1}{K}.$$

The first term on the right hand side of (1.13) can therefore be bounded by σ^2/K. This already shows that one needs to let the number of neighbors K go to infinity when $n \to +\infty$ for the expected excess risk to converge to 0.

Now, in order to equilibrate between the two contributions in (1.13), we need to more carefully quantify the squared bias. This requires bounds on the distances to the nearest neighbors; in fact, in view of (1.8) and (1.12),

$$\sum_{i=1}^n \widehat{w}_i(x')\mathtt{d}(x_i, x')^2 \leqslant \mathtt{d}\left(x_{i_K(x')}, x'\right)^2, \tag{1.14}$$

so controlling the distance to the K-th neighbor is key. We need at this stage to introduce another assumption.

Assumption 1.3. *The distribution $p_{\mathrm{data}}(dx')$ of the inputs has a compact support $\mathcal{X} \subset \mathbb{R}^d$.*

The fact that an additional assumption is needed is related to the so-called *no free lunch theorem*, which roughly says that there is no algorithm that can learn a fully arbitrary distribution (see Section 10.3 for further discussions). In the remainder, we denote by

$$\mathrm{diam}(\mathcal{X}) = \max_{(x,x')\in\mathcal{X}^2} \mathtt{d}(x,x') < +\infty$$

the diameter of the compact set $\mathcal{X}$ (which is closed and bounded), and choose $\mathtt{d}(x,x') = \|x-x'\|_\infty$ with

$$\|\xi\|_\infty = \max_{1\leqslant i\leqslant d} |\xi_i|$$

the ℓ^∞ norm on $\mathbb{R}^d$.

Lemma 1.1. *Suppose that Assumption 1.3 holds true, and that $d \geqslant 2$. Consider random variables $(X_1,\dots,X_n,X_{n+1})$ independently and identically distributed according to p. Then, the expected squared distance between X_{n+1} and its first nearest neighbor among $\{X_1,\dots,X_n\}$ in ℓ^∞ norm is upper bounded by $4\,\mathrm{diam}(\mathcal{X})^2 n^{-2/d}$.*

Note that the scaling of the distance with respect to the dimension is not so good, as a very large number of points $n \sim \varepsilon^{-d}$ is needed in order for the expected squared distance to be of order ε^2.

Proof. Denote by $X_{(i)}$ a nearest-neighbor of X_i among $\{X_1,\dots,X_{n+1}\}\setminus\{X_i\}$. Note that $R_i^2 = \|X_i - X_{(i)}\|_\infty^2$ has the same distribution for all $1\leqslant i\leqslant n+1$, so that

$$\mathbb{E}(R_{n+1}^2) = \frac{1}{n+1}\sum_{i=1}^{n+1}\mathbb{E}(R_i^2). \tag{1.15}$$

Moreover, $R_i \leqslant \mathrm{diam}(\mathcal{X})$. Now, the sets $B_i = \{x\in\mathbb{R}^d \,|\, \|x-X_i\|_\infty < R_i/2\}$ are disjoint for $i\neq j$ because $\|X_i - X_j\|_\infty \geqslant \max(R_i,R_j)$. Up to a translation, we can assume that $\mathcal{X}\subset[0,\mathrm{diam}(\mathcal{X})]^d$, so that

$$\bigcup_{i=1}^n B_i \subset \left[-\frac{\mathrm{diam}(\mathcal{X})}{2}, \frac{3\mathrm{diam}(\mathcal{X})}{2}\right]^d.$$

The union of the sets $B_1,\dots,B_{n+1}$ has therefore a volume smaller than $2^d\mathrm{diam}(\mathcal{X})^d$. As a consequence,

$$\sum_{i=1}^{n+1} R_i^d \leqslant 2^d\mathrm{diam}(\mathcal{X})^d,$$

and, by convexity (since $d/2\geqslant 1$),

$$\left(\frac{1}{n+1}\sum_{i=1}^{n+1} R_i^2\right)^{d/2} \leqslant \frac{1}{n+1}\sum_{i=1}^{n+1} R_i^d \leqslant \frac{2^d\mathrm{diam}(\mathcal{X})^d}{n+1}. \tag{1.16}$$

Finally,

$$\frac{1}{n+1}\sum_{i=1}^{n+1} R_i^2 \leqslant \frac{4\,\mathrm{diam}(\mathcal{X})^2}{(n+1)^{2/d}}.$$

The desired result follows by combining the latter bound with (1.15). □

The expected square distance between a new data point and its K-th nearest neighbor among n independent data points can then be bounded as follows.

Lemma 1.2. *Suppose that Assumption 1.3 holds true, and that $d\geqslant 2$. Consider random variables $(X_1,\dots,X_n,X_{n+1})$ independently and identically distributed according to p_{data}. Then,*

$$\mathbb{E}\left(\left\|X_{n+1} - X_{i_K(X_{n+1})}\right\|_\infty^2\right) \leqslant 8\,\mathrm{diam}(\mathcal{X})^2\left(\frac{2K}{n}\right)^{2/d}.$$

Proof. Assume without loss of generality that $2K \leqslant n$ (otherwise the bound is trivial). We randomly split the set $X_1, \dots, X_n$ into $2K$ sets of sizes approximately $n/(2K)$ (some sets have sizes $\lfloor n/(2K) \rfloor$, others have sizes $\lceil n/(2K) \rceil$), and denote by $X_{(K)}^j$ a 1-nearest neighbor of X_{n+1} in the j-th set. Then, $\left\|X_{n+1} - X_{i_K(X_{n+1})}\right\|_\infty$ is smaller than the K-th term among $\left\|X_{n+1} - X_{(K)}^j\right\|_\infty$ for $1 \leqslant j \leqslant 2K$ since the minimum to find the K-th nearest neighbor is taken over a smaller set of elements $X_{(K)}^1, \dots, X_{(K)}^{2K}$ instead of $X_1, \dots, X_n$.

Now, for a nonnegative sequence of elements $a_1, \dots, a_{2K}$, the k-th smallest term among them is upper bounded by $(a_1 + \cdots + a_{2K})/(2K - k)$ as there are $2K - k$ terms larger than the k-th smallest term in the sum $a_1 + \cdots + a_{2K}$. Applying the latter inequality for $a_j = \left\|X_{n+1} - X_{(K)}^j\right\|_\infty^2$ and $k = K$ gives

$$\left\|X_{n+1} - X_{i_K(X_{n+1})}\right\|_\infty^2 \leqslant \frac{1}{K} \sum_{j=1}^{2K} \left\|X_{n+1} - X_{(K)}^j\right\|_\infty^2 .$$

We next use Lemma 1.1 to upper bound the expectation of $\left\|X_{n+1} - X_{(K)}^j\right\|_\infty^2$ by $4 \operatorname{diam}(\mathcal{X})^2 (n/2K)^{-2/d}$ (in fact, we use (1.16) and replace n by $\lfloor n/(2K) \rfloor$ to obtain this bound). This gives

$$\mathbb{E}\left(\left\|X_{n+1} - X_{i_K(X_{n+1})}\right\|_\infty^2\right) \leqslant \frac{1}{K} \sum_{j=1}^{2K} \mathbb{E}\left(\left\|X_{n+1} - X_{(K)}^j\right\|_\infty^2\right) \leqslant \frac{8 \operatorname{diam}(\mathcal{X})^2}{(n/2K)^{2/d}},$$

which is indeed the desired bound. □

The combination of (1.13), (1.14) and Lemma 1.2 finally leads to the following upper bound for the expected excess risk of prediction for the K-th nearest neighbor method:

$$0 \leqslant \mathbb{E}_{\mathcal{D}}\left[\mathcal{R}(\widehat{f})\right] - \mathcal{R}^\star \leqslant \frac{\sigma^2}{K} + 8B^2 \operatorname{diam}(\mathcal{X})^2 \left(\frac{2K}{n}\right)^{2/d}. \tag{1.17}$$

Note that the last term has a bad scaling with respect to the dimension because f is only assumed to be Lipschitz (and not more regular than this). It is possible at this stage to find the value of K which minimizes the right hand side of the previous inequality. This gives some theoretical guidance on the choice of K.

Exercise 1.12. *Determine the optimal choice of K for the upper bound as a function of n and the various parameters of the model (σ, $\operatorname{diam}(\mathcal{X})$ and B), and interpret the various scalings. Write also the associated upper bound on the expected excess risk.*

The previous exercise shows that

$$0 \leqslant \mathbb{E}_{\mathcal{D}}\left[\mathcal{R}(\widehat{f})\right] - \mathcal{R}^\star \leqslant \frac{C}{n^{2/(d+2)}}.$$

In view of the first equality in (1.13), and when d is large, this can be rewritten as

$$\sqrt{\mathbb{E}_{\mathcal{D}}\left[\left\|\widehat{f} - f^\star\right\|_{L^2(p_{\mathrm{data}})}^2\right]} = \mathrm{O}\left(n^{-1/d}\right). \tag{1.18}$$

The estimate on the right hand side is typical of the so-called "curse of dimensionality". To give an intuition of the scalings encountered in this context, consider the situation when we have n points in dimension d, regularly spaced on a grid. The distance between neighboring points is of order $n^{-1/d}$ in each direction (as there are $n = m^d$ points overall when there are m points in each direction; the points being at a distance $1/m$ in each direction). When $f^\star(x')$ is approximated by the value $f^\star(x_i)$ at the closest grid point x_i, the error is of order

$$|f^\star(x') - f^\star(x_i)| \sim B\|x' - x_i\| \sim n^{-1/d},$$

which is consistent with (1.18).

2

Least square regression

We consider in this chapter a simple (the simplest?) model to predict real valued outputs in a regression context, namely methods based on least square regression with linear models. In contrast to other scientific fields, we focus here on the prediction capabilities of the model rather than on its statistical properties.

Various lessons and features of least square models will be encountered for more complex models, and therefore serve as basic examples to build up an intuition of statistical learning. In particular we will see how to derive estimates which allow to

- capture a bias-variance tradeoff;
- make precise how the generalization performance degrades with the dimension when no regularization is considered, and how it can remain stable when regularization is introduced;
- generalize the approach to more complex models with nonlinear features (kernel methods and neural networks).

Overall, the results obtained in this chapter are simple to derive with rather basic tools of linear algebra, and no optimization algorithm is needed to find predictors since these predictors are given by closed form or analytic expressions in many cases (except for Section 2.4).

We start by presenting the general framework of linear least square regression in Section 2.1, and next introduce the ordinary least square method in Section 2.2. We then discuss two ways to prevent overfitting in least square regression, by adding either a penalization on the Euclidean norm of the parameter (ridge regression; see Section 2.3) or on its ℓ^1 norm (the so-called LASSO method; see Section 2.4).

2.1 General framework of linear least square regression

We provide in this section a general framework for linear least square regression problems, in the case of one dimensional outputs $\mathcal{Y} = \mathbb{R}$ for simplicity. We follow the presentation in [4, Section 3.2]. The aim of least square regression is to approximate the Bayes predictor $f^\star(x') = \mathbb{E}(y \,|\, x = x')$ associated with the loss function[1]

$$\mathcal{R}(f) = \mathbb{E}\left(|y - f(x)|^2\right).$$

Recall that the Bayes predictor is derived in Exercise 1.2. We consider to this end a parametrized family of predictors $f_\theta : \mathcal{X} \to \mathcal{Y}$ for parameters $\theta \in \Theta$. We also replace the population loss by an empirical risk. Denoting by $\mathcal{D} = \{(x_i, y_i)_{1 \leqslant i \leqslant n}\}$ the training set, the minimization problem to consider is therefore

$$\inf_{\theta \in \Theta} \widehat{\mathcal{R}}_n(f_\theta), \qquad \widehat{\mathcal{R}}_n(f) = \frac{1}{n} \sum_{i=1}^n \left(y_i - f_\theta(x_i)\right)^2. \tag{2.1}$$

We denote by $\widehat{\theta} \in \Theta$ a minimizer of the latter problem, assuming that such a minimizer indeed exists.

Until now we have described a rather general least square regression problem, which is relevant for many models including neural networks (see Chapter 8). Linear least square regression corresponds to considering functions f_θ linear in θ; but possibly nonlinear in x. Generically, these functions are written as

$$f_\theta(x) = \theta^\top \varphi(x) \in \mathbb{R}, \qquad \theta \in \Theta = \mathbb{R}^d, \tag{2.2}$$

where $\varphi : \mathcal{X} \to \mathbb{R}^d$ is a featurization function. Before reformulating the regression problem in a more abstract manner, let us first give three paradigmatic examples:

- *simple linear regression* corresponds to the case when $f_\theta(x) = \theta_0 + \theta_1 x$ with $x \in \mathbb{R}$ and $\theta = (\theta_0, \theta_1) \in \mathbb{R}^2$. Therefore, $\varphi(x) = (1, x) \in \mathbb{R}^2$;
- *multiple linear regression* extends the previous example to the situation when $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$, by considering $\varphi(x) = (1, x) \in \mathbb{R}^{d+1}$ and $\theta \in \mathbb{R}^{d+1}$, so that

$$f_\theta(x) = \theta_0 + \theta_1 x_1 + \cdots + \theta_d x_d \in \mathbb{R};$$

- *polynomial regression* is described, for one-dimensional inputs $x \in \mathbb{R}$, by the featurization function $\varphi(x) = (1, x, x^2, \ldots, x^k)$. Therefore, $\theta \in \mathbb{R}^{k+1}$ and

$$f_\theta(x) = \theta_0 + \theta_1 x_1 + \cdots + \theta_k x^k \in \mathbb{R};$$

- *kernel regression* will be mentioned later on in Section 6.4.

Let us conclude the presentation of the general framework by reformulating (2.1) for the choice (2.2) as (with some abuse of notation on $\widehat{\mathcal{R}}_n$)

$$\inf_{\theta \in \mathbb{R}^d} \widehat{\mathcal{R}}_n(\theta), \qquad \widehat{\mathcal{R}}_n(\theta) = \frac{1}{n} \|Y - X\theta\|_2^2, \tag{2.3}$$

where $\|\cdot\|_2$ is the standard Euclidean norm on $\mathbb{R}^n$, and

$$Y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix} \in \mathbb{R}^{n \times 1}, \qquad X = \begin{pmatrix} \text{—} \; \varphi(x_1) \; \text{—} \\ \vdots \\ \text{—} \; \varphi(x_n) \; \text{—} \end{pmatrix} \in \mathbb{R}^{n \times d}. \tag{2.4}$$

Note that $X\theta \in \mathbb{R}^n$ has the same dimension as Y.

[1] Note that we use here the abuse of notation where the random variables x, y are not denoted by capital letters. We keep the notation X for the matrix gathering the data.

Exercise 2.1 (Regression with weight functions). *Fix n inputs $x_1, \dots, x_n \in \mathcal{X}$, and assume that the associated labels in $\mathcal{Y} = \mathbb{R}$ satisfy*

$$y_i = f^*(x_i) + \xi_i,$$

where $(\xi_i)_{1 \leqslant i \leqslant n}$ are independent and identically distributed one dimensional random variables of mean 0 and variance σ^2, and $f^* : \mathcal{X} \to \mathcal{Y}$ is a given function. Consider the prediction function

$$\widehat{f}(x) = \sum_{i=1}^{n} w_i(x) y_i,$$

where $w_1, \dots, w_n$ are deterministic weight functions. Prove that

$$\mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n}\left(\widehat{f}(x_i) - f^*(x_i)\right)^2\right] = \frac{1}{n}\|WF^* - F^*\|_2^2 + \frac{\sigma^2}{n}\mathrm{Tr}\left(W^\top W\right),$$

where the expectation is over $(\xi_1, \dots, \xi_n)$, F^* is the column vectors whose i-th entry is $f^*(x_i)$, and $W \in \mathbb{R}^{n \times n}$ is a matrix to make precise.

2.2 Ordinary least square regression

Ordinary least square regression is the simplest regression model, but it is already quite useful, and its analysis provides a lot of insight. The presentation in this section is based on [4, Sections 3.3 to 3.5] (see also Section 3.8 for more advanced topics not covered in the lectures) and [41, Section 11.2].

2.2.1 Derivation of the least square estimator

We make the following key assumption throughout all this section (but not for Sections 2.3 and 2.4).

Assumption 2.1. *The dimension of the features d is smaller than the number of data points n, and the matrix X defined in* (2.4) *has full rank d.*

For polynomial regression, this assumption is satisfied as soon as all the inputs $(x_i)_{1 \leqslant i \leqslant n} \subset \mathbb{R}$ are distinct since X is a Vandermonde matrix.

Exercise 2.2. *Prove that the minimization problem* (2.3) *is well posed (*i.e. *there exists a unique minimizer), and find the expression of this global minimum.*

The above exercise shows that the unique minimizer of (2.3) is

$$\widehat{\theta} = \frac{1}{n}\widehat{\Sigma}^{-1} X^\top Y, \tag{2.5}$$

with

$$\widehat{\Sigma} = \frac{1}{n} X^\top X \in \mathbb{R}^{d \times d} \tag{2.6}$$

the uncentered covariance of the input data. The normalization factor $1/n$ ensures that all entries of the matrix $\widehat{\Sigma}$ are of order 1 uniformly in n; and similarly for entries of $X^\top Y / n$.

2.2.2 Interpretations of the least square estimator

We provide in this section various reformulations of the least square regression problem, which allow to make a connection with results seen in Statistics.

Simple regression. For simple regression, for which $x_i \in \mathbb{R}$, the expression of $\widehat{\theta}$ can be made even more explicit.

Exercise 2.3. *Prove that*

$$\widehat{\theta}_0 = \overline{y} - \widehat{\theta}_1 \overline{x}, \qquad \widehat{\theta}_1 = \frac{\mathrm{Cov}\left(\{(x_i)_{1\leqslant i\leqslant n}\}, \{(y_i)_{1\leqslant i\leqslant n}\}\right)}{\mathrm{Var}\{(x_i)_{1\leqslant i\leqslant n}\}},$$

where the empirical means are

$$\overline{x} = \frac{1}{n}\sum_{i=1}^{n} x_i, \qquad \overline{y} = \frac{1}{n}\sum_{i=1}^{n} y_i,$$

while the empirical covariance and variance are defined as

$$\mathrm{Cov}\left(\{(a_k)_{1\leqslant k\leqslant K}\}, \{(b_k)_{1\leqslant k\leqslant K}\}\right) = \frac{1}{K}\sum_{k=1}^{K} a_k b_k - \left(\frac{1}{K}\sum_{k=1}^{K} a_k\right)\left(\frac{1}{K}\sum_{k=1}^{K} b_k\right),$$

and $\mathrm{Var}\{(a_k)_{1\leqslant k\leqslant K}\} = \mathrm{Cov}(\{(a_k)_{1\leqslant k\leqslant K}\}, \{(a_k)_{1\leqslant k\leqslant K}\})$.

Maximum likelihood estimation in the Gaussian model. In order to make a connection with Statistics, we consider a situation where it is assumed that the outputs y_i are related to the inputs x_i as

$$y_i = \theta^\top \varphi(x_i) + \varepsilon_i,$$

where $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$ are independent Gaussian random variables (supposed to be one-dimensional for simplicity; but the analysis in this section can be straightforwardly extended to the case when the outputs are multidimensional). In this context, the likelihood of a realization is

$$\prod_{i=1}^{n} \sqrt{\frac{1}{2\pi\sigma^2}} \exp\left(-\frac{\varepsilon_i^2}{2\sigma^2}\right).$$

Now, $\varepsilon_i^2 = (y_i - \theta^\top \varphi(x_i))^2$, so that the likelihood is proportional to

$$\exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}\left|y_i - \theta^\top \varphi(x_i)\right|^2\right) = \exp\left(-\frac{n}{2\sigma^2}\widehat{\mathcal{R}}_n(\theta)\right).$$

Finding the least square estimator is therefore equivalent in this context to finding the maximum likelihood estimator for the Gaussian model at hand.

Orthogonal projection. We recall here an interpretation of the vector of predictions $\widehat{Y} = X\widehat{\theta}$ in terms of an orthogonal projection of Y.

Proposition 2.1. *The vector of predictions* $\widehat{Y} = X(X^\top X)^{-1}X^\top Y$ *is the orthogonal projection of* $Y \in \mathbb{R}^n$ *onto* $\mathrm{Ran}(X) = \mathrm{Span}(X_1, \ldots, X_d) \subset \mathbb{R}^n$*, the column space of* X*.*

Proof. Introduce the matrix $P = X(X^\top X)^{-1}X^\top$. Note that P is a projector since $P^2 = P$. Moreover, $P(Xa) = Xa$ for any $a \in \mathbb{R}^d$, so that $Pu = u$ for any $u \in \mathrm{Ran}(X)$. Finally,[2] given the expression of P, it is clear that $Pu' = 0$ for any $u' \in \mathrm{Ker}(X^\top) = \mathrm{Ran}(X)^\perp$. This shows that P is the orthogonal projector onto $\mathrm{Ran}(X)$. □

[2] Alternatively, one could note that $(Y - \widehat{Y})^\top Xa = 0$ for any $a \in \mathbb{R}^d$, which implies that $(\mathrm{Id} - P)Y$ is orthogonal to $\mathrm{Ran}(X)$.

Prediction errors. We characterize here the quality of the prediction on the training data set, using the coefficient of determination. More precisely, when $\mathbf{1}_n \in \mathrm{Ran}(X)$ (which is the case for instance when the first component of $\varphi(x)$ is 1),

$$\widehat{\mathcal{R}}_n(\widehat{\theta}) = \mathrm{Var}\{(y_i)_{1\leqslant i\leqslant n}\}(1-r^2), \qquad r^2 = \frac{\mathrm{Var}\{(\widehat{y}_i)_{1\leqslant i\leqslant n}\}}{\mathrm{Var}\{(y_i)_{1\leqslant i\leqslant n}\}} = \frac{\left\|\widehat{Y} - \overline{\widehat{y}}\mathbf{1}_n\right\|_2^2}{\|Y-\overline{y}\mathbf{1}_n\|_2^2}.$$

To prove this equality, we note first that Y and $\widehat{Y}$ have the same means since $Y-\widehat{Y}$ is orthogonal to $\mathbf{1}_n \in \mathrm{Ran}(X)$. We can then decompose $Y-\overline{y}\mathbf{1}_n$ as the sum of $Y-\widehat{Y}$ and $\widehat{Y}-\overline{y}\mathbf{1}_n$ and use Pythagoras' theorem to write

$$\widehat{\mathcal{R}}_n(\widehat{\theta}) = \frac{1}{n}\left\|Y-\widehat{Y}\right\|_2^2 = \frac{1}{n}\left(\|Y-\overline{y}\mathbf{1}_n\|_2^2 - \left\|\widehat{Y}-\overline{y}\mathbf{1}_n\right\|_2^2\right) = \mathrm{Var}\{(y_i)_{1\leqslant i\leqslant n}\} - \mathrm{Var}\{(\widehat{y}_i)_{1\leqslant i\leqslant n}\}.$$

The prediction error on the training data set is smaller when r is closer to 1. In fact, since the triangle $\overline{\widehat{y}}\mathbf{1}_n$, Y, $\widehat{Y}$ is rectangle in $\widehat{Y}$, the ratio of lengths defining r also corresponds to the cosine of some angle, and eventually corresponds to some normalized covariance of the data:

$$r = \frac{\left\|\widehat{Y}-\overline{\widehat{y}}\mathbf{1}_n\right\|_2}{\|Y-\overline{y}\mathbf{1}_n\|_2} = \frac{\left\langle \widehat{Y}-\overline{\widehat{y}}\mathbf{1}_n, Y-\overline{y}\mathbf{1}_n\right\rangle}{\left\|\widehat{Y}-\overline{\widehat{y}}\mathbf{1}_n\right\|_2 \|Y-\overline{y}\mathbf{1}_n\|_2} = \frac{\mathrm{Cov}\left(\widehat{Y},Y\right)}{\sqrt{\mathrm{Var}(Y)\mathrm{Var}(\widehat{Y})}},$$

with some abuse of notation for the arguments of Var and Cov.

2.2.3 Numerical solution of the ordinary least square problem

We discuss in this section how to solve (2.3) in practice. A first remark is that a good practice is to standardize the data (recall Section 1.1.4) by replacing $x_{n,j}$ with

$$x'_{n,j} = \frac{x_{n,j}-\overline{x}_j}{\overline{\sigma}_j}, \qquad \overline{\sigma}_j = \sqrt{\mathrm{Var}\{(x_{i,j})_{1\leqslant i\leqslant n}\}}.$$

This allows to replace X with a matrix X' where each column has mean 0 and empirical variance 1. There are then two main approaches to solve (2.3):

- rely on gradient methods to perform a direct minimization of the loss function; see Chapter 4. This may be the most efficient approach when d is large, although usually plain gradient methods are not used to find solutions, but rather conjugate gradient methods or quasi-Newton methods, typical choices considered for computational linear algebra (see [41, Section 11.2.2.3] and references therein).
- solve the linear system
$$X^\top X\widehat{\theta} = X^\top Y \tag{2.7}$$
in view of the expression of the global minimizer $\widehat{\theta}$ given by (2.5).

We discuss here the second option. The first remark is that it is not a good idea to invert the matrix $X^\top X$ as this may be numerically unstable and has a large computational cost $\mathrm{O}(d^3)$ for the inversion (and $\mathrm{O}(n^2d^2)$ for constructing the matrix). Solving the linear system (2.7) is computationally much cheaper and also more stable. There are two main options to do so (see [41, Section 11.2.2.3]):

- use a singular value decomposition to write
$$X = USV^\top, \tag{2.8}$$
where $U \in \mathbb{R}^{n\times n}$ with $U^\top U = \mathrm{Id}_n$, $S \in \mathbb{R}^{n\times d}$ contains the $r = \min(n,d)$ singular values $\lambda_1,\dots,\lambda_d$ on its main diagonal and entries 0 otherwise, and $V \in \mathbb{R}^{d\times d}$ with $VV^\top = V^\top V = \mathrm{Id}_d$. The cost of performing this SVD is $\mathrm{O}(nd\times\min(n,d))$.

In order to solve $X^\top X\widehat{\theta} = X^\top Y$, we note that $X^\top X = V(S^\top S)V^\top$, where $S^\top S \in \mathbb{R}^{d\times d}$ is a diagonal matrix with entries $\lambda_1^2, \dots, \lambda_d^2$. The solution $\widehat{\theta}$ is found by first performing the SVD of X, then computing $\widehat{\theta} = V(S^\top S)^{-1}V^\top X^\top Y$.
See also Remark 2.1 below for a discussion on how to compute predictions when $n \leqslant d$ (which is not the situation we consider here).

- a second option is to perform a QR decomposition, where one writes $X = QR$ with $Q = [q_1|\dots|q_d] \in \mathbb{R}^{n\times d}$ and $q_i \cdot q_j = \delta_{ij}$, and $R \in \mathbb{R}^{d\times d}$ an upper triangular matrix. This option can be interesting when $n \gg d$. The QR decomposition is obtained by a pivoting strategy where one writes $X_1 = r_{11}q_1$ (which determines q_1) then $X_2 = r_{12}q_1 + r_{22}q_2$ (which determines q_2), etc. The system to solve (2.7) can be rewritten as $R^\top Q^\top QR\widehat{\theta} = R^\top R\widehat{\theta} = X^\top Y = R^\top Q^\top Y$, so that it suffices to solve the simple triangular system $R\widehat{\theta} = Q^\top Y$ (which has a cost $\mathrm{O}(d^2)$ once the QR decomposition is obtained; the cost of the QR decomposition being $\mathrm{O}(d^3)$, which can be cheaper than the cost of SVD when $n \gg d$).

In `scikit-learn`, the method `sklean.linear_model.fit` calls the routine `scipy.linalg.lstsq`, which itself calls by default the LAPACK routine DGELSD, which relies on SVD.

Remark 2.1 (Computing predictions for $n \leqslant d$). *When $n \leqslant d$ (with X of full rank n in this case), one should rewrite the problem to solve as $XX^\top X\widehat{\theta} = XX^\top Y$ and use that $XX^\top = U(SS^\top)U^\top \in \mathbb{R}^{n\times n}$, where $SS^\top \in \mathbb{R}^{n\times n}$ is a diagonal matrix with entries $\lambda_1^2, \dots, \lambda_n^2$. This is sufficient to compute the prediction $\widehat{Y} = X\widehat{\theta}$.*

2.2.4 Statistical analysis

We discuss in this section how to prove guarantees on the performance of predictions based on ordinary least squares. Two settings can be considered:

- *random design*: both the inputs x_i and the outputs y_i are random. This is the classical setting of machine learning where one wants to generalize predictions to unseen data;
- *fixed design*: the input data $(x_1, \dots, x_n)$ is fixed (and therefore not random), and we consider the prediction error only at these points. In this case, the associated population risk is

$$\mathcal{R}(\theta) = \mathbb{E}_Y\left[\frac{1}{n}\|Y - X\theta\|_2^2\right],$$

where the expectation is over the realizations y_i for each input x_i. This can be seen as the so-called "in-sample prediction error", and corresponds to learning the vector $X\theta_\star$ of best predictions instead of a function $\mathcal{X} \to \mathbb{R}$. In this setting, no generalization to unseen inputs is attempted.

We consider here the fixed design setting. The main interest of this setting is that it is mathematically simpler to analyze. We will only hint at generalizations to the random design setting (see [4, Section 3.8] for further elements).

In view of the no free lunch theorem (see Chapter 10), we need assumptions on how the data is generated in order to provide guarantees. Throughout this section, we make the following assumption, which corresponds to the situation of a so-called well specified model:

$$\exists \theta_\star \in \mathbb{R}^d \text{ such that } y_i = \theta_\star^\top \varphi(x_i) + \varepsilon_i \text{ with } \varepsilon_i \text{ iid, } \mathbb{E}(\varepsilon_i) = 0,\ \mathbb{E}(\varepsilon_i^2) = \sigma^2. \tag{2.9}$$

Some conditions on the noise could weakened; in particular, it would be sufficient to assume that the ε_i are independent but not necessarily identically distributed, with $\mathbb{E}(\varepsilon_i^2) \leqslant \sigma^2$.

We show below that the minimal value of the excess risk for this model is 0 as the Bayes predictor belongs to the class of functions upon which the loss is minimized.

Exercise 2.4. *Give the expression of the Bayes predictor.*

The fact that the model is well specified allows to obtain sharper bounds of order $\mathrm{O}(1/n)$ instead of bounds $\mathrm{O}(n^{-1/2})$ obtained for the general analysis of models in supervised learning (see Chapter 10.1). To write such bounds, we consider the risk averaged over the realizations of the data set, *i.e.* in expectation with respect to realizations of the outputs for fixed inputs, somewhat similarly to the situation considered for KNNs where the risk was averaged over realizations of the data set. This leads to the notion of "average excess risk", where "average" refers to realizations of the training data set, while the notion of excess risk builds upon the expected risk, which refers to realizations of test data.

The first result is a decomposition of the risk into a squared bias and a variance, as made precise in the following exercise.

Exercise 2.5 (Risk decomposition). *Assume that* (2.9) *holds.*

(a) Prove that $\mathcal{R}(\theta) = \sigma^2 + \dfrac{1}{n}(\theta - \theta_\star)^\top X^\top X(\theta - \theta_\star)$.

(b) Deduce that $\theta_\star$ *is the unique global minimum of* $\mathcal{R}$ *and compute the value of the minimal loss* $\mathcal{R}^\star$.

(c) The results of the previous questions imply that

$$\mathbb{E}_Y\left[\mathcal{R}\left(\widehat{\theta}\right)\right] - \mathcal{R}^\star = \mathbb{E}_Y\left[\left\|\widehat{\theta} - \theta_\star\right\|_{\widehat{\Sigma}}^2\right],$$

where $\|a\|_{\widehat{\Sigma}} = \sqrt{a^\top \widehat{\Sigma} a}$ *is the Mahalanobis distance (recall the definition* (2.6) *of* $\widehat{\Sigma}$*). Prove that the following bias-variance decomposition holds for the average excess risk:*

$$\mathbb{E}_Y\left[\mathcal{R}\left(\widehat{\theta}\right)\right] - \mathcal{R}^\star = \left\|\mathbb{E}_Y\left[\widehat{\theta}\right] - \theta_\star\right\|_{\widehat{\Sigma}}^2 + \mathbb{E}_Y\left[\left\|\widehat{\theta} - \mathbb{E}_Y\left(\widehat{\theta}\right)\right\|_{\widehat{\Sigma}}^2\right]. \tag{2.10}$$

The next exercise summarizes some properties of $\widehat{\theta}$.

Exercise 2.6 (Properties of $\widehat{\theta}$). *Assume that* (2.9) *holds.*

(a) Prove that $\widehat{\theta}$ *is an unbiased estimator of* $\theta_\star$, i.e. $\mathbb{E}_Y\left[\widehat{\theta}\right] = \theta_\star$ *(where we recall that the expectation is taken only over the realizations of the outputs).*

(b) Show that the covariance of $\widehat{\theta}$ *is*

$$\mathrm{Cov}_Y\left(\widehat{\theta}\right) = \mathbb{E}_Y\left[\left(\widehat{\theta} - \theta_\star\right)\left(\widehat{\theta} - \theta_\star\right)^\top\right] = \frac{\sigma^2}{n}\widehat{\Sigma}^{-1} \in \mathbb{R}^{d\times d}.$$

We can finally use the information on the estimator $\widehat{\theta}$ to obtain guarantees on the average risk.

Proposition 2.2 (Risk of ordinary least square estimator). *Assume that* (2.9) *holds. Then, the average excess risk of* $\widehat{\theta}$ *is*

$$\mathbb{E}_Y\left[\mathcal{R}\left(\widehat{\theta}\right)\right] - \mathcal{R}^\star = \frac{\sigma^2 d}{n}.$$

The interpretation of this result is the following:

(i) the factor d in the numerator shows that ordinary least squares does not work well in high dimension. Some regularization is needed in order to obtain results which have a better scaling with respect to the dimension (and possibly additional assumptions on the structure of the data distribution);

(ii) the result can be extended to the random design setting by replacing the bound of Proposition 2.2 with

$$\frac{\sigma^2}{n}\mathbb{E}\left[\mathrm{Tr}\left(\Sigma\widehat{\Sigma}^{-1}\right)\right], \qquad \Sigma = \mathbb{E}\left[\varphi(x)\varphi(x)^\top\right], \tag{2.11}$$

see [4, Proposition 3.10] and Exercise 2.7 below. Obtaining estimates on the trace in the expectation however requires some non trivial results on random matrices (in particular concentration inequalities for matrices for good lower bounds on $\widehat{\Sigma}$).

Proof. From (2.10) and Exercise 2.6(a), and since covariance matrices are symmetric,

$$\mathbb{E}_Y\left[\mathcal{R}\left(\widehat{\theta}\right)\right] - \mathcal{R}^\star = \mathbb{E}_Y\left[\left\|\widehat{\theta} - \theta_\star\right\|_{\widehat{\Sigma}}^2\right] = \sum_{k,k'=1}^{d} \widehat{\Sigma}_{k,k'} \mathbb{E}_Y\left(\left[\widehat{\theta} - \theta_\star\right]_k \left[\widehat{\theta} - \theta_\star\right]_{k'}\right)$$
$$= \sum_{k,k'=1}^{d} \widehat{\Sigma}_{k,k'} \left[\mathrm{Cov}_Y(\widehat{\theta})\right]_{k,k'} = \mathrm{Tr}\left(\mathrm{Cov}_Y(\widehat{\theta})\widehat{\Sigma}\right).$$

We next use Exercise 2.6(b) to obtain

$$\mathbb{E}_Y\left[\mathcal{R}\left(\widehat{\theta}\right)\right] - \mathcal{R}^\star = \frac{\sigma^2}{n}\mathrm{Tr}\left(\mathrm{Id}_d\right),$$

which gives the desired equality. □

Exercise 2.7 (Average excess risk in the random design setting). *We consider in this exercise the random design setting,* i.e. *a situation in which the inputs $x_1, \dots, x_n$ in the data set are independently sampled according to some probability measure $p_{\mathrm{data}}(dx)$, with associated labels $y_i = \varphi(x_i)^\top \theta_\star + \varepsilon_i$, where the random variables ε_i are independent from each other and from the inputs $x_1, \dots, x_n$, with $\mathbb{E}(\varepsilon_i) = 0$ and $\mathbb{E}(\varepsilon_i^2) = \sigma^2$.*

(a) Prove that $\mathcal{R}(\theta) - \mathcal{R}^\star = \|\theta - \theta_\star\|_\Sigma^2$ with $\mathcal{R}^\star = \sigma^2$ and Σ defined in (2.11).
(b) Assume that $\widehat{\Sigma}$ is invertible. Show that

$$\mathbb{E}\left[\mathcal{R}(\widehat{\theta})\right] - \mathcal{R}^\star = \frac{\sigma^2}{n}\mathbb{E}\left[\mathrm{Tr}\left(\Sigma\widehat{\Sigma}^{-1}\right)\right],$$

where the expectation is over the realizations of the data set.

Proposition 2.2 gives a result on the average excess risk, which is necessarily nonnegative. It is instructive to compare the estimate to the one obtained for the training risk, as provided by the following exercise.

Exercise 2.8. *Assume that* (2.9) *holds with $\varepsilon \sim \sigma\mathcal{N}(0, \mathrm{Id}_n)$. Prove that the average excess training risk is*

$$\mathbb{E}_Y\left[\widehat{\mathcal{R}}_n\left(\widehat{\theta}\right)\right] - \mathcal{R}^\star = -\frac{\sigma^2 d}{n}.$$

The key comment is that the average excess training risk is negative, so that the average training risk is smaller than the Bayes risk; in fact, there is a sign change on the term of the right hand side compared to the equation in Proposition 2.2. The difference between the risk (as provided by Proposition 2.2) and the training risk gives an indication of the degree of overfitting. The larger the difference between these two quantities is, the larger the average excess risk is. This is akin to some form of generalization gap, but not strictly as we work here in the fixed design setting. The estimates provided here still allow to perform some model selection and detect overfitting.

2.3 Ridge regression

This section presents a very important idea in machine learning, namely that adding a regularization (penalty) term to the empirical risk to be minimized can in fact improve the predictive performance of least square estimators. This may seem counterintuitive at first sight, but crucially relies on the fact that having a small training loss does not mean that the model has a small test loss, due to overfitting. The main motivation for regularization is therefore to avoid overfitting by reducing the effective size of the parameter space. For ridge regression, the extra penalty term is

proportional to the squared Euclidean norm of the parameter. Our presentation is based on [4, Sections 3.6] and [41, Sections 4.5, 5.4 and 11.3].

Let us start by giving an example to motivate the need for regularization. Say that we aim at predicting whether the outcome of a coin tossing is heads or tails; and that out of 3 draws, we get 3 heads. Are we going to predict that all subsequent draws will give heads? Most likely not.

This example illustrates the core issue of the problem: when there are too many parameters in the model to exactly fit the data (including the inherent noise involved in the realizations of the random variables at hand), then one actually fits the empirical distribution of the data, and not the target test distribution. One should therefore prevent a perfect minimization of the training loss, which would put all the mass on the training data, in order to leave some space to cover the (unseen) test data. We next formalize this idea with the notion of capacity control, and then turn to a specific way to obtain such a capacity control, namely ridge penalization.

2.3.1 Capacity control

Capacity control consists in limiting the complexity, or expressivity, of the model by adding a penalization term to the empirical risk function. More precisely, the objective function to minimize becomes, for a parametrized class of functions $\{f_\theta,\ \theta \in \Theta\}$,

$$\widehat{\mathcal{R}}_n(f_\theta) + \lambda \Omega(\theta), \tag{2.12}$$

for some penalization function $\Omega : \Theta \to \mathbb{R}_+$, and a penalization strength $\lambda \geqslant 0$. Paradigmatic examples are

- ridge penalization: $\Omega(\theta) = \|\theta\|_2^2$. This is also called "weight decay", for reasons that will be made precise below (see Exercise 2.10);
- LASSO (the acronym is motivated and explained in Section 2.4): $\Omega(\theta) = \|\theta\|_1$;
- elastic net, which is a combination of the previous two penalizations, and therefore corresponds to adding a penalty term of the form $\lambda_1\|\theta\|_1 + \lambda_2\|\theta\|_2^2$ for $\lambda_1, \lambda_2 \geqslant 0$.

Finding the penalization strength by (cross) validation. The most appropriate value of the penalization parameter λ is found by (cross) validation (see for instance the presentation in [41, Section 5.4.3]). We describe the procedure in the context of a large data set, as in Section 1.3.2.1, the adaptation to smaller data sets with K-fold cross validation being straightforward. More precisely, consider a training set $\mathcal{D}_{\text{train}}$ and a validation set $\mathcal{D}_{\text{val}}$. One first trains the penalized empirical risk (2.12) on the training data $\mathcal{D}_{\text{train}}$ for various values of λ:

$$\widehat{\theta}_\lambda \in \operatorname*{argmin}_{\theta \in \Theta} \left\{ \widehat{\mathcal{R}}_n(f_\theta) + \lambda \Omega(\theta) \right\}.$$

This defines a family of minimizers $f_{\widehat{\theta}_\lambda}$ indexed by λ. The best value of λ is then found as the one minimizing the *unpenalized* loss on the validation data $\mathcal{D}_{\text{val}} = \{(x_m, y_m)_{1 \leqslant m \leqslant N_{\text{val}}}\}$:

$$\lambda^\star \in \operatorname*{argmin}_{\lambda \geqslant 0} \left\{ \frac{1}{N_{\text{val}}} \sum_{m=1}^{N_{\text{val}}} \ell\left(f_{\widehat{\theta}_\lambda}(x_m), y_m \right) \right\}.$$

Let us emphasize here that one should not add a penalization term to the validation loss, as this loss is an approximation of the test loss, which is ultimately the quantity one wants to minimize. The whole point of the (cross) validation procedure is to determine the strength of the penalization to add to the training loss in order to prevent overfitting and obtain better predictions.

Alternative formulation of capacity control. Regularization is often introduced by adding a penalty term as in (2.12). The latter loss function can however be seen as the minimization of the Lagrangian associated with the family of constrained optimization problems (indexed by $D \geqslant 0$):

$$\inf_{\theta \in \Theta} \left\{ \widehat{\mathcal{R}}_n(f_\theta) \;\middle|\; \Omega(\theta) \leqslant D \right\}. \tag{2.13}$$

Note that the unconstrained minimization of $\widehat{\mathcal{R}}_n$ is obtained in the limit $D \to +\infty$; whereas capacity control becomes stronger as $D \to 0$. The parameter D can be seen as the dual parameter of the penalization strength $\lambda \geqslant 0$.

We will most often consider the minimization of the penalized loss (2.12), but sometimes it is useful to consider the dual perspective (2.13) (see for instance the discussion at the end of Section 2.4.1).

2.3.2 Ridge penalization

We discuss in this section least square regression with a ridge penalization, which corresponds to the so-called ridge regression.[3] One crucial interest of ridge regression is that one can clearly study the impact and benefit of regularization, as the minimizer of (2.12) has an analytical expression similar to the one obtained for ordinary least squares in (2.5).

In order to motivate once again the interest of regularization in the context of least square regression, consider the situation when $d/n \to 1$, or simply $n = d$. In this case, (2.5) simplifies as $\widehat{\theta} = X^{-1}Y$ (since $\left(X^\top X\right)^{-1} X^\top X = \mathrm{Id}_n$ and X is invertible, so that $X^{-1} = \left(X^\top X\right)^{-1} X^\top$). The prediction $X\widehat{\theta} = Y$ then gives a perfect fit to the training data. This is not a great news in terms of generalizing to unseen data points as all degrees of freedom in $\widehat{\theta}$ are used to match training data points, so that there is absolutely no flexibility to adjust for test data points. The situation is even worse for $d > n$, as in this case the solutions to the minimization problem (2.3) are not unique since one can add to a minimizer an arbitrary element of the kernel of X. Some of the solutions to the minimization problem can therefore have a very large norm.

Analytical expression of the solution to the ridge regression problem. In this section and the end of this chapter, we no longer suppose that Assumption 2.1 holds, namely that X has full rank d. This allows in particular to consider situations where $d \geqslant n + 1$.

Definition 2.1 (Ridge least square regression). *For $\lambda > 0$,*

$$\widehat{\theta}_\lambda = \operatorname*{argmin}_{\theta \in \mathbb{R}^d} \left\{ \frac{1}{n} \|Y - X\theta\|_2^2 + \lambda \|\theta\|_2^2 \right\}.$$

Exercise 2.9. *Prove that the minimization problem in Definition 2.1 is well posed for $\lambda > 0$ (recall that we no longer assume that X has full rank), and that the minimizer can be explicitly written as*

$$\widehat{\theta}_\lambda = \frac{1}{n} \left(\widehat{\Sigma} + \lambda \mathrm{Id}_d\right)^{-1} X^\top Y, \tag{2.14}$$

where $\widehat{\Sigma}$ is still defined by (2.6).

Exercise 2.10 (Motivating the terminology "weight decay"). *Consider a one dimensional featurization function $\phi : \mathcal{X} \to \mathbb{R}$, and the associated ridge regression problem*

$$\widehat{\mathcal{R}}_{n,\lambda}(\theta) = \frac{1}{n} \sum_{i=1}^n \left(y_i - \theta\phi(x_i)\right)^2 + \lambda\theta^2$$

for $\theta \in \Theta = \mathbb{R}$ and $\lambda > 0$. Give the expression of the optimal parameter $\widehat{\theta}_\lambda$ of Definition 2.1, and motivate the terminology "weight decay".

other motivation (to add): update in SGD

Exercise 2.11. *Show that $\widehat{\theta}_\lambda$ in (2.14) can be equivalently written as*

$$\widehat{\theta}_\lambda = X^\top \left(XX^\top + n\lambda \mathrm{Id}_n\right)^{-1} Y.$$

What could the interest of such a reformulation?

[3] We do not discuss here the origin of the term 'ridge', which has a long history dating back to the end of the 50s.

Numerical solution of the ridge regression problem. Let us first discuss how to adapt the methods described in Section 2.2.3 for ordinary least squares to ridge regression. For gradient methods and the approach based on QR decompositions, we first show how to reduce the minimization problem in Definition 2.1 to a minimization problem for ordinary least squares. We introduce to this end

$$\widetilde{X} = \begin{pmatrix} X \\ \sqrt{n\lambda}\mathrm{Id}_d \end{pmatrix} \in \mathbb{R}^{(n+d)\times d}, \qquad \widetilde{Y} = \begin{pmatrix} Y \\ 0_d \end{pmatrix} \in \mathbb{R}^{(n+d)\times 1}.$$

Then,

$$\frac{1}{n}\|Y - X\theta\|_2^2 + \lambda\|\theta\|_2^2 = \frac{1}{n}\left\|\widetilde{Y} - \widetilde{X}\theta\right\|_2^2.$$

The right hand side of the previous equality is the typical quantity to minimize for ordinary least squares. This formulation shows that the cost of computing $\widehat{\theta}_\lambda$ is $\mathrm{O}((n+d)^3)$ for the approach based on the QR decomposition of $\widetilde{X}$; while it remains of order $\mathrm{O}(n+d)$ per gradient step if one prefers to resort to the (stochastic) gradient approaches described in Chapter 4.

For the approach based on SVD, we follow [25, Section 18.3.5] and write $X = USV^\top$, with the same matrices as in (2.8). When $d > n$, the matrix S has $d-n$ elements 0 at the end of its n lines. One can therefore retain only the first n columns of S, in a matrix denoted by $\widetilde{S}$, and also get rid of the $d-n$ last lines of $V^\top$, *i.e.* keep only the n first columns of V, in a matrix denoted by $\widetilde{V}$. It still holds $X = U\widetilde{S}\widetilde{V}^\top$. We next introduce $R = U\widetilde{S} \in \mathbb{R}^{n\times n}$. Then, $X = R\widetilde{V}^\top$, and simple computations show that

$$\widehat{\theta}_\lambda = V\left(R^\top R + n\lambda\mathrm{Id}_n\right)^{-1} R^\top Y.$$

The parameter therefore corresponds to the one obtained by ridge regression based on the n data points $\{(r_i, y_i)\}_{1\leqslant i\leqslant n}$, with r_i the ith row of R. This amounts to reducing the data matrix from $X \in \mathbb{R}^{n\to d}$ to $R \in \mathbb{R}^{n\times n}$, allowing to reduce the computational cost to $\mathrm{O}(dn^2)$ (the cost of the SVD).

When solutions for several (many) values of λ are needed, in order to perform (cross) validation in particular, it may be beneficial to consider a regularization path, where one starts by solving the problem for the largest value of λ at hand, and then successively solves the optimization problems for decreasing values of λ, using the solution found for the previous value of λ in order to initialize or approximate the solution for the new value of λ (as some form of warm start in the optimization procedure). One starts from the largest value of λ as the associated minimization problem is usually easier to solve than for λ small. In fact, in the limit $\lambda \to +\infty$, the solution to the minimization problem is simply $\widehat{\theta}_\infty = 0$, so one expects $\widehat{\theta}_\lambda$ to be close to 0 for λ large.

Exercise 2.12 (Leave-one-out estimator for ridge regression). *We consider the usual setting of ridge regression, where one aims at minimizing the following regularized empirical loss for $\lambda > 0$ and a feature function $\varphi : \mathcal{X} \to \mathbb{R}^d$:*

$$\widehat{R}_\lambda(\theta) = \frac{1}{n}\sum_{i=1}^n \left(y_i - \theta^\top\varphi(x_i)\right)^2 + \lambda\|\theta\|_2^2,$$

for n given data points $(x_i, y_i) \in \mathcal{X}\times\mathcal{Y}$, with $\mathcal{Y} = \mathbb{R}$. The parameter λ is chosen by cross validation. We study here a limiting case of K-fold cross validation where the number of folds K is equal to the number of points n. This amounts to fitting the parameters of the model on a training data set of $n-1$ points, and computing the loss on a validation set of a single point; this procedure being performed n times by considering all possible single point validation sets.

We denote by $X \in \mathbb{R}^{n\times d}$ the full design matrix, whose j-th row is $\varphi(x_j)^\top$, and by $X_{-i} \in \mathbb{R}^{(n-1)\times d}$ the design matrix obtained by removing the i-th line $\varphi(x_i)^\top$. Similarly, $Y \in \mathbb{R}^n$ is the column vector whose j-th entry is y_j, and $Y_{-i} \in \mathbb{R}^{n-1}$ the column vector obtained from Y by removing the i-th entry y_i. The weight parameter associated with the data set where the i-th entry is missing is therefore defined as (note that we still divide by a factor n even though the first term on the right hand side involves only $n-1$ terms)

$$\widehat{\theta}_{-i,\lambda} = \operatorname*{argmin}_{\theta\in\mathbb{R}^d} \frac{1}{n}\|Y_{-i} - X_{-i}\theta\|_2^2 + \lambda\|\theta\|_2^2.$$

The prediction for the input x_i is then $\widehat{\theta}_{-i,\lambda}^\top\varphi(x_i)$, which should be compared with the true label y_i, leading to the leave-one-out error

$$\mathcal{R}_\lambda^{\mathrm{LOO}} = \frac{1}{n}\sum_{i=1}^n \left(y_i - \widehat{\theta}_{-i,\lambda}^\top\varphi(x_i)\right)^2.$$

(a) Give the expression of $\widehat{\theta}_{-i,\lambda}$ in terms of X_{-i}, Y_{-i}.

(b) We next consider the training set where the i-th data point (x_i, y_i) is replaced by $(x_i, \widetilde{y}_i)$ with $\widetilde{y}_i = \widehat{\theta}_{-i,\lambda}^\top\varphi(x_i)$. The associated vector of labels is denoted by $\widetilde{Y}_{(i)}$. Show that

$$\widetilde{\theta}_{i,\lambda} = \operatorname*{argmin}_{\theta\in\mathbb{R}^d} \frac{1}{n}\left\|\widetilde{Y}_{(i)} - X\theta\right\|_2^2 + \lambda\|\theta\|_2^2$$

coincides with $\widehat{\theta}_{-i,\lambda}$.

(c) Deduce an expression of $\widehat{\theta}_{-i,\lambda}$ in terms of X and $\widetilde{Y}_{(i)}$.

(d) Define the matrix $H = X(X^\top X + n\lambda\mathrm{Id}_d)^{-1}X^\top \in \mathbb{R}^{d\times d}$, and the diagonal matrix $h \in \mathbb{R}^{d\times d}$ with entries $h_{i,i} = H_{i,i}$. Prove that

$$(1 - H_{i,i})\,(y_i - \widetilde{y}_i) = y_i - (HY)_i.$$

(e) Assume that $H_{i,i} \neq 1$ for all $1 \leqslant i \leqslant n$. Conclude that

$$\mathcal{R}_\lambda^{\mathrm{LOO}} = \frac{1}{n}\left\|(\mathrm{Id}_d - h)^{-1}(\mathrm{Id}_d - H)Y\right\|_2^2.$$

(f) Why is this last formula interesting from a computational viewpoint?

2.3.3 Statistical properties of ridge regression

In order to study the statistical performance of ridge regression, we consider as in Section 2.2.4 the fixed design setting and study the average excess risk, namely the expectation of the excess risk

$$\mathcal{R}(\theta) - \mathcal{R}^* = \mathbb{E}_{Y'}\left[\frac{1}{n}\|Y' - X\theta\|_2^2\right] - \mathcal{R}^*$$

averaged over all possible training sets with inputs fixed, for the predictor based on $\widehat{\theta}_\lambda$ (which depends on the training set Y at hand).

Proposition 2.3. *Suppose that* (2.9) *holds. Then the excess risk for ridge regression is*

$$\mathbb{E}\left[\mathcal{R}\left(\widehat{\theta}_\lambda\right)\right] - \mathcal{R}^\star = \lambda^2\theta_\star^\top\left(\widehat{\Sigma} + \lambda\mathrm{Id}_d\right)^{-2}\widehat{\Sigma}\theta_\star + \frac{\sigma^2}{n}\mathrm{Tr}\left(\widehat{\Sigma}^2\left(\widehat{\Sigma} + \lambda\mathrm{Id}_d\right)^{-2}\right).$$

The decomposition on the right hand side of the above equality again allows to interpret the excess risk as the sum of a squared bias (proportional to λ^2) and a variance in the predictions. This formula deserves several comments:

- the result reduces to the one of Proposition 2.2 for ordinary least squares when $\lambda = 0$, as there is no bias term in this case, while the trace in the variance term is $\mathrm{Tr}(\mathrm{Id}_d) = d$;
- for large values of λ, the squared bias is of order 1 while the variance is of order $1/(n\lambda^2)$; while for small values of λ, the squared bias is of order λ^2 and the variance of order $1/n$. This suggests that the value of λ should be chosen in order to equilibriate the two contributions, namely

$$\lambda \sim \frac{1}{\sqrt{n}}.$$

This corresponds the usual bias/variance tradeoff for minimizing the mean square error. For the above choice for the scaling of λ, the excess risk can be shown to scale as $1/\sqrt{n}$ (see Proposition 2.4 below).

- the bias, of order λ, can be seen as some form of approximation error (see Section 10.1). It increases as λ increases;
- the variance term can be seen as some form of estimation error (see Section 10.1), which decreases when n and λ increase;
- it is useful to interpret the variance term using the concept of effective number of degrees of freedom.[4] More precisely, denoting by $\widehat{\sigma}_j^2$ the eigenvalues of the symmetric, positive semidefinite matrix $\widehat{\Sigma}$, the effective number of degrees of freedom is

$$0 \leqslant \operatorname{Tr}\left(\widehat{\Sigma}^2\left(\widehat{\Sigma}+\lambda \mathrm{Id}_d\right)^{-2}\right)=\sum_{j=1}^d \frac{\widehat{\sigma}_j^4}{\left(\widehat{\sigma}_j^2+\lambda\right)^2} \leqslant d.$$

When $\lambda = 0$, the effective number of degrees of freedom is d as there is no capacity control. This number converges to 0 as $\lambda \to +\infty$.

Exercise 2.13. *The aim of this exercise is to prove Proposition 2.3.*

(a) Prove that $\mathbb{E}_Y\left[\widehat{\theta}_\lambda\right]-\theta_\star=-\lambda\left(\widehat{\Sigma}+\lambda \mathrm{Id}_d\right)^{-1} \theta_\star$.

(b) Use the following equality for $u, v \in \mathbb{R}^d$ *and* $M \in \mathbb{R}^{d \times d}$ *(where* $vu^\top \in \mathbb{R}^{d \times d}$ *is the matrix with entries* $v_i u_j$ *for* $1 \leqslant i, j \leqslant d$*):*

$$u^\top M v=\sum_{i, j=1}^d M_{j i} u_j v_i=\operatorname{Tr}\left[\left(v u^\top\right) M^\top\right],$$

to obtain

$$\mathbb{E}_Y\left[\left\|\widehat{\theta}_\lambda-\mathbb{E}_Y\left(\widehat{\theta}_\lambda\right)\right\|_{\widehat{\Sigma}}^2\right]=\frac{\sigma^2}{n} \operatorname{Tr}\left(\widehat{\Sigma}^2\left(\widehat{\Sigma}+\lambda \mathrm{Id}_d\right)^{-2}\right).$$

(c) Conclude with the risk decomposition (2.10) *applied to* $\widehat{\theta}_\lambda$.

Now that Proposition 2.3 provides an explicit expression for the average excess risk, we can optimize the value of λ in order to obtain the tightest upper bound to the average excess risk.

Proposition 2.4 (Upper bound on the excess risk for ridge regression). *Suppose that* (2.9) *holds, and consider*

$$\lambda_\star=\frac{\sigma \sqrt{\operatorname{Tr}\left(\widehat{\Sigma}\right)}}{\|\theta_\star\|_2 \sqrt{n}}.$$

Then,

$$0 \leqslant \mathbb{E}_Y\left[\mathcal{R}\left(\widehat{\theta}_{\lambda_\star}\right)\right]-\mathcal{R}^\star \leqslant \frac{\sigma \sqrt{\operatorname{Tr}\left(\widehat{\Sigma}\right)}\|\theta_\star\|_2}{2 \sqrt{n}}. \tag{2.15}$$

Let us make a few comments on this result before providing its proof:

- the upper bound (2.15) is dimension free as d does not explicitly appear (in contrast to Proposition 2.2), but only implicitly. The favorable situation is when $\|\theta_\star\|_2$ and

$$\operatorname{Tr}\left(\widehat{\Sigma}\right)=\frac{1}{n} \sum_{i=1}^n\left\|\varphi\left(x_i\right)\right\|_2^2$$

[4] Our convention here differs from the more standard one considered in [41, Section 11.3.2] or [25, Section 3.4.3] for instance, where the discussion is based on the predictions rather than on the risk.

both remain bounded as the dimension increases. A sufficient condition for the latter term to be bounded is that $\|\varphi(x)\|_2 \leqslant \Phi < +\infty$ uniformly in $x \in \mathcal{X}$ for some $\Phi \in \mathbb{R}_+$ as the dimension increases;

- the convergence rate $\mathrm{O}(1/\sqrt{n})$ which is obtained is slower than the rate $\mathrm{O}(1/n)$ of Proposition 2.2 for ordinary least square regression, but it has on the other side a milder dependence on the noise (σ instead of σ^2) and on the dimension d. The estimate may therefore be useful in the nonasymptotic regime where n is not too large, as it corresponds to a slower rate of convergence but potentially a better prefactor in the estimate;
- the expression of the optimal value $\lambda_\star$ cannot be used to determine the regularization strength in practice for two reasons: (i) the values of σ and $\theta_\star$ are usually unknown; (ii) the value of $\lambda_\star$ is obtained by optimizing an upper bound, and the so-obtained value may therefore be suboptimal for the actual average (test) risk. The regularization strength should be determined in practice by (cross)validation, as discussed in Section 2.3.1. Nonetheless, the expression for $\lambda_\star$ still leads to useful theoretical insights, in particular ideas on how the optimal regularization strength behaves as the number of training data points is increased.

Exercise 2.14. *The aim of this exercise is to prove Proposition 2.4.*

(a) Prove that the eigenvalues of $\lambda\widehat{\Sigma}\left(\widehat{\Sigma}+\lambda\mathrm{Id}_d\right)^{-2}$ *are smaller than* $1/4$.
(b) Deduce that

$$0 \leqslant \mathbb{E}_Y\left[\mathcal{R}\left(\widehat{\theta}_\lambda\right)\right] - \mathcal{R}^\star \leqslant \frac{\lambda}{4}\|\theta_\star\|_2^2 + \frac{\sigma^2}{4n\lambda}\mathrm{Tr}\left(\widehat{\Sigma}\right) := u(\lambda).$$

(c) Conclude.

Exercise 2.15. *The aim of this exercise is to characterize how* $\mathrm{Tr}\left(\widehat{\Sigma}\right)$ *scales with respect to the dimension* d *for the particular example of polynomial regression on* $\mathcal{X} = [-1,1]$, *in the limit* $n \to +\infty$. *Recall that in this case* $\varphi(x) = (1, x, x^2, \ldots, x^{d-1})$ *when* $\theta \in \mathbb{R}^d$.

(1) Compute the entries of the limiting matrix $\Sigma = \lim_{n\to+\infty} \widehat{\Sigma}$ *when the inputs* x_i *are independently and identically distributed according to the uniform distribution on* $\mathcal{X}$.
(2) Determine the asymptotic behavior of $\mathrm{Tr}(\Sigma)$ *as* $d \to +\infty$, *and interpret the result.*

Exercise 2.16 (Ridge regression in the random design setting). *We consider the same setting as in Exercise 2.7 in the context of ridge regression. Prove that*

$$\mathbb{E}\left[\mathcal{R}(\widehat{\theta}_\lambda)\right] - \mathcal{R}^\star = \lambda^2\mathbb{E}\left[\theta_\star^\top\left(\widehat{\Sigma}+\lambda\mathrm{Id}_d\right)^{-1}\Sigma\left(\widehat{\Sigma}+\lambda\mathrm{Id}_d\right)^{-1}\theta_\star\right] + \frac{\sigma^2}{n}\mathbb{E}\left[\mathrm{Tr}\left(\left(\widehat{\Sigma}+\lambda\mathrm{Id}_d\right)^{-2}\widehat{\Sigma}\Sigma\right)\right],$$

where the expectation is over the realizations of the data set.

Exercise 2.17 (Partition estimators). *Consider a partition* $A_1, \ldots, A_J$ *of the input space* $\mathcal{X}$, *namely a collection of subsets* $A_j \subset \mathcal{X}$ *such that*

$$A_i \cap A_j = \emptyset \text{ if } i \neq j, \qquad A_1 \cup \cdots \cup A_J = \mathcal{X}.$$

A predictor function is constructed from $\theta = (b, w) \in \mathbb{R}^{J+1}$ *with* $w = (w_1, \ldots, w_J)$ *as*

$$f_\theta(x') = b + w^\top\phi(x') = b + \sum_{j=1}^J w_j \mathbf{1}_{A_j}(x'),$$

where $\mathbf{1}_A(x) = 1$ *if* $x \in A$ *and* $\mathbf{1}_A(x) = 0$ *if* $x \notin A$, *and* $\phi = (\mathbf{1}_{A_1}, \ldots, \mathbf{1}_{A_J}) \in \mathbb{R}^{1\times J}$. *Given a data set* $\mathcal{D} = \{(x_i, y_i)_{1\leqslant i\leqslant n}\}$, *the parameters to use for the prediction are found by minimizing the following penalized empirical risk for* $\lambda > 0$*:*

$$\widehat{\mathcal{R}}_\lambda(\theta) = \frac{1}{n}\sum_{i=1}^n (y_i - f_\theta(x_i))^2 + \lambda\sum_{j=1}^J w_j^2.$$

Note that we consider a ridge regularization on w *only. Denote by* N_j *is the number of inputs* x_i *belonging to* A_j, *and assume that* $N_j \geqslant 1$ *for any* $1 \leqslant j \leqslant J$.

(a) Find the optimal value $\widehat{b}$ in the minimization of $\mathcal{R}_\lambda(\theta)$ over b only, with w fixed.
(b) Find the optimal value of $\widehat{w}_\lambda$ in the minimization of $\mathcal{R}_\lambda(b, w)$ over w only, with b fixed.
(c) What is the prediction for new point $x' \in A_j$?

2.4 LASSO regression

LASSO was introduced by Tibshirani in 1996 (see [54]). The acronym stands for "Least Absolute Shrinkage and Selection Operator". As the name indicates, an essential aim of Lasso regularization is to perform feature selection, and build sparse predictors that depend only a small number of the components of the feature vector $\varphi(x)$. This is useful for two reasons:

(i) it makes models more interpretable (as it allows to decide which features are important for predictions), and/or also allows to add extra features, whose relevance is unclear, as irrelevant features will automatically be discarded;
(ii) as for ridge regression, it also allows to reduce the dimension dependence in the guarantee bounds on the excess risk (recall for instance the bounds $\mathrm{O}(n^{-1/d})$ for K-nearest neighbors obtained in (1.18), and the bound $\sigma^2 d/n$ of Proposition 2.2 for ordinary least square regression).

There are however two difficulties with this approach:

(i) the identity of the relevant variables is not known beforehand. They need therefore to be identified;
(ii) Lasso works only if the Bayes predictor itself involves only a small number of features (at least approximatively).

The presentation in this section follows [41, Section 11.4] and [4, Chapter 8].

2.4.1 Sparsity inducing regularization terms

As in the remainder of this chapter, we focus here on linear methods, and therefore aim at solving

$$\min_{\theta \in \mathbb{R}^d} \mathbb{E}\left[\ell\left(y, \varphi(x)^\top \theta\right)\right].$$

In fact, we will often specify our analysis to the regression setting for which $\ell(y, z) = \|y - z\|_2^2$ is the squared Euclidean norm, but the framework discussed here is general enough to cover other situations.

As motivated in Section 2.3, a regularization term $\Omega(\theta)$ should be added to the training loss in order to prevent overfitting. We discuss two options here, which both lead to sparse solutions:

- a first option to consider $\Omega(\theta) = \|\theta\|_0$, which is the number of non zero entries of θ. This naturally favors sparse solutions where many entries of the optimal parameter θ are 0. However, the associated optimization problem

$$\min_{\theta \in \mathbb{R}^d} \left\{\widehat{\mathcal{R}}_n(\theta) + \lambda \|\theta\|_0\right\}$$

  is difficult to solve as it essentially requires some form of combinatorial approach in order to identify the set of relevant $k = \|\theta\|_0$ variables, or rely on dedicated algorithms such as greedy methods where features are introduced one by one and chosen to minimize the loss ("orthogonal matching pursuit"; see for instance [50, Section 25.1.2]).
- another option is to rely on a ℓ^1 penalty term

$$\Omega(\theta) = \|\theta\|_1 = \sum_{k=1}^{d} |\theta_k|. \tag{2.16}$$

The main interest of this approach is that the function appearing in the minimization problem

$$\min_{\theta\in\mathbb{R}^d} \left\{\widehat{\mathcal{R}}_n(\theta) + \lambda\|\theta\|_1\right\} \tag{2.17}$$

is convex when $\widehat{\mathcal{R}}_n$ is convex (see Exercise 2.18 below). Gradient methods (as described in Chapter 4) can then be used to approximate the minimizer. It is however not clear at first sight why the ℓ^1 penalty (2.16) leads to a sparse minimizer. We next motivate this. The combination of the sparsification property, and the fact that the regularized loss to be minimized has good properties (convexity) makes the method particularly attractive.

Exercise 2.18. *Prove that the function $\theta \mapsto \widehat{\mathcal{R}}_n(\theta) + \lambda\|\theta\|_1$ is convex on $\mathbb{R}^d$ for the empirical risk function $\widehat{\mathcal{R}}_n$ defined in* (2.3) *and any $\lambda \geqslant 0$.*

Motivating the sparsity inducing effect of ℓ^1 regularization. Balls for the ℓ^1 norm have extremal points which "stick out more" (see Figure 2.1). These extremal points correspond to sparse elements $\theta \in \mathbb{R}^d$ where one component at least is 0. The tangency or contact between isolines of $\widehat{\mathcal{R}}_n$ and the unit ball in the ℓ^1 norm tends to happen at the corners or vertices of the unit ball. For ridge regression on the other hand, the relevant unit ball to consider in the one in the ℓ^2 norm, whose boundary is a hypersphere, which is fully symmetric and does not have sparse extremal points.

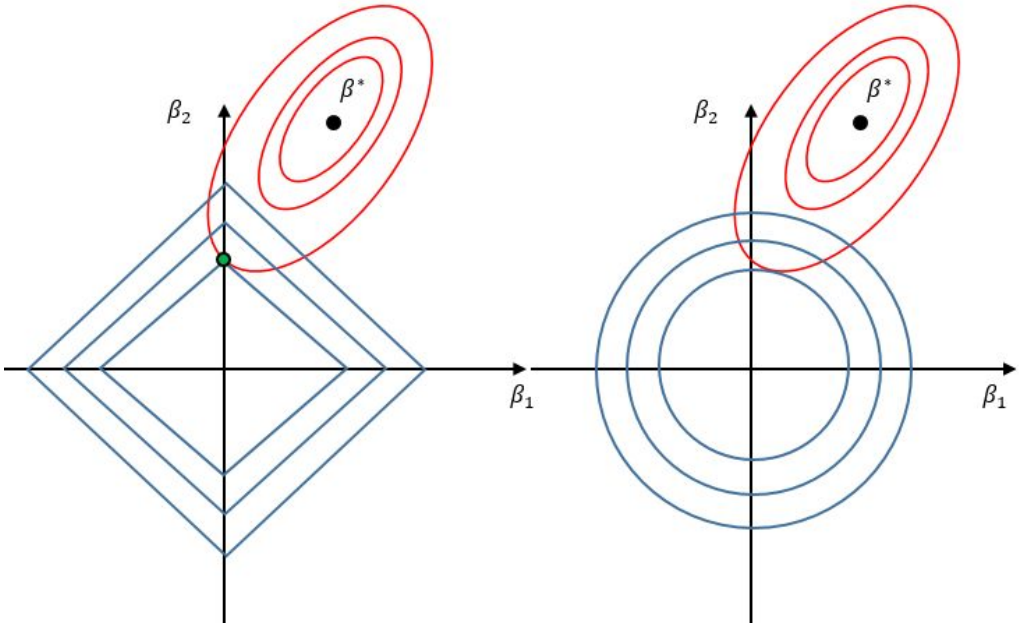

Fig. 2.1: Sparsifying effect of LASSO. Picture taken from https://www.linkedin.com/posts/shristikedia-b17444167_datascience-machinelearning-lasso-activity-7266137633242652672-gaO7/

In order to make contact between the above considerations and the minimization problem (2.17), we recall that (2.17) can be seen as some dual formulation of the constrained minimization problem (2.13). The latter setting is exactly the one considered above. This motivates why sparse solutions are expected for ℓ^1 regularization, but not for ℓ^2 regularization.

2.4.2 Explicit expressions for one dimensional problems

In order to gain some intuition on the behavior of solutions to the Lasso minimization problem (2.17), we consider the simplest possible setting, namely the situation when $\theta \in \mathbb{R}$ and there is

a single one dimensional data point $x = 1$ with associated output $y \in \mathbb{R}$. In this case, the function to minimize is

$$F_\lambda(\theta) = \frac{1}{2}(y-\theta)^2 + \lambda|\theta|, \tag{2.18}$$

where we introduced a factor $1/2$ in front of the first term for convenience. The next exercise shows how to analytically obtain the unique global minimizer of F_λ.

Exercise 2.19. *Consider the function F_λ defined in* (2.18)*.*

(1) Prove that F_λ admits a unique minimizer.
(2) Compute the left and right derivatives of F_λ at any $\theta \in \mathbb{R}$.
(3) Prove that the minimizer $\theta^\star_\lambda$ of F_λ is

$$\theta^\star_\lambda = \begin{cases} y+\lambda & \text{for } y \leqslant -\lambda, \\ 0 & \text{for } -\lambda \leqslant y \leqslant \lambda, \\ y-\lambda & \text{for } y \geqslant \lambda, \end{cases} \tag{2.19}$$

The expression (2.19) for $\theta^\star_\lambda$ can be written in a more condensed way as

$$\theta^\star_\lambda = \operatorname{sign}(y)\max\left(|y|-\lambda, 0\right).$$

This corresponds to a soft thresholding function, where the value of y is shifted by an amount $\lambda > 0$ towards 0 (*i.e.* one add λ if $y < 0$ and subtracts λ if $y > 0$) unless y is too small, in which case it is set to 0.

2.4.3 Expressions for the Lasso least square regression problem

We consider in this section the general Lasso least square regression problem, where one seeks to minimize

$$\widehat{\mathcal{R}}_{n,\lambda}(\theta) = \frac{1}{2n}\|Y - X\theta\|_2^2 + \lambda\|\theta\|_1 = \frac{1}{2n}\sum_{i=1}^n \left(y_i - \theta^\top \varphi(x_i)\right)^2 + \lambda\sum_{k=1}^d |\theta_k|.$$

As in (2.18), we introduce a factor $1/2$ in front of the empirical risk in order to simplify the algebra. This is not a restriction of generality as it amounts to rescaling the values of λ by a factor 2. The function $\widehat{\mathcal{R}}_{n,\lambda}$ is convex as the sum of two convex functions, and strongly convex when X has full rank.

The optimality condition satisfied by a global minimizer reads

$$0_d \in \partial\widehat{\mathcal{R}}_{n,\lambda}(\theta^\star_\lambda),$$

where $\partial\widehat{\mathcal{R}}_{n,\lambda}$ is the subgradient (see Section 4.1.2 for some background material on subgradients). Since $\widehat{\mathcal{R}}_{n,\lambda}$ is smooth on $(\mathbb{R}\setminus\{0\})^d$, the components of the subgradient are simply the partial derivatives for the components of θ which are non zero.

In order to derive necessary conditions of optimality, let us next consider variations with respect to the component θ_k when $\theta_k = 0$. We fix to this end the components $\theta_1, \ldots, \theta_{k-1}, \theta_{k+1}, \ldots, \theta_d$, and introduce the function

$$\begin{aligned}\mathscr{R}_k(\theta_k) &= \widehat{\mathcal{R}}_{n,\lambda}\left(\theta_1, \ldots, \theta_{k-1}, \theta_k, \theta_{k+1}, \ldots, \theta_d\right) \\ &= \frac{1}{2n}\sum_{i=1}^n \left(a_{ik} - \theta_k\varphi_k(x_i)\right)^2 + \lambda\left|\theta_k\right| + L_k(\theta_1, \ldots, \theta_{k-1}, \theta_{k+1}, \ldots, \theta_d),\end{aligned}$$

where L_k does not depend on θ_k, and

$$a_{ik} = y_i - \sum_{\ell\neq k}\theta_\ell\varphi_\ell(x_i). \tag{2.20}$$

We next rewrite $\mathscr{R}_k(\theta_k)$ in a manner similar to (2.18) as follows:

$$\begin{aligned}\mathscr{R}_k(\theta_k) &= \left(\frac{1}{2n}\sum_{i=1}^n \varphi_k(x_i)^2\right)\theta_k^2 - \frac{1}{n}\left(\sum_{i=1}^n a_{ik}\varphi_k(x_i)\right)\theta_k + \lambda\,|\theta_k| + \widetilde{L}_k(\theta_1,\dots,\theta_{k-1},\theta_{k+1},\dots,\theta_d)\\ &= \left(\frac{1}{n}\sum_{i=1}^n \varphi_k(x_i)^2\right)\left[\frac{1}{2}\left(\theta_k - \frac{\displaystyle\sum_{i=1}^n a_{ik}\varphi_k(x_i)}{\displaystyle\sum_{i=1}^n \varphi_k(x_i)^2}\right)^2 + \frac{n\lambda}{\displaystyle\sum_{i=1}^n \varphi_k(x_i)^2}\,|\theta_k|\right] + \widehat{L}_k(\theta_1,\dots,\theta_{k-1},\theta_{k+1},\dots,\theta_d).\end{aligned}$$

In view of (2.19), we therefore find that the optimal parameter $\theta_\lambda^\star$ necessarily satisfies

$$\theta_{k,\lambda}^\star = \begin{cases} \dfrac{\displaystyle\sum_{i=1}^n a_{ik}^\star\varphi_k(x_i) + n\lambda}{\displaystyle\sum_{i=1}^n \varphi_k(x_i)^2} & \text{for } \dfrac{1}{n}\displaystyle\sum_{i=1}^n a_{ik}^\star\varphi_k(x_i) \leqslant -\lambda,\\ 0 & \text{for } -\lambda \leqslant \dfrac{1}{n}\displaystyle\sum_{i=1}^n a_{ik}^\star\varphi_k(x_i) \leqslant \lambda,\\ \dfrac{\displaystyle\sum_{i=1}^n a_{ik}^\star\varphi_k(x_i) - n\lambda}{\displaystyle\sum_{i=1}^n \varphi_k(x_i)^2} & \text{for } \dfrac{1}{n}\displaystyle\sum_{i=1}^n a_{ik}^\star\varphi_k(x_i) \geqslant \lambda. \end{cases} \tag{2.21}$$

where $a_{ik}^\star$ is (2.20) evaluated at $\theta = \theta_\lambda^\star$.

The latter expression motivates the term "shrinkage" in the expression "least absolute shrinkage" for Lasso, as large coefficients (in absolute value) are shrunk towards 0. This also suggests that the Lasso estimator is biased. This may not be an issue for two reasons:

(1) we want a good prediction, and do not care so much about estimating the true parameter $\theta_\star$ or recovering the true model (which is the aim of high dimensional statistics);
(2) introducing some bias may be helpful in view of the bias/variance tradeoff, especially if the magnitude of the bias is determined using cross validation.

2.4.4 Numerical methods to find the minimizers

The optimality conditions (2.21) are cumbersome since the components $\theta_{k',\lambda}^\star$ for $k' \neq k$ appear in the coefficients $a_{ik}^\star$. The numerical methods to find solutions to (2.17) are therefore based on other ideas than solving the nonlinear equation (2.21) (see [41, Section 11.4.9] for further precisions):

- a popular approach, implemented for instance in the default method for Lasso regression in `scikit-learn`, is *coordinate descent*: at each iteration, a coordinate to update is selected (for instance by cycling over all components or by choosing it at random), and a one dimensional minimization, with other coordinates fixed, is performed. The interest of this approach is that the one dimensional minimization can be performed analytically, following the computations in Section 2.4.3. This leads to expressions close to (2.21) with coefficients a_{ik} evaluated at the current value of the parameter θ;
- another approach is based on a *projected gradient method*, for which the parameter $\theta \in \mathbb{R}^d$ is decomposed as $\theta = \theta^+ - \theta^-$, where $\theta^+, \theta^- \in \mathbb{R}_+^d$ have only nonnegative components. The optimization problem (2.17) can then be reformulated as the minimization of a smooth function under positivity constraints:

$$\min_{\theta^+,\theta^- \in \mathbb{R}^d_+} \left\{ \widehat{\mathcal{R}}_n\left(\theta^+ - \theta^-\right) + \lambda \sum_{k=1}^{d} \left(\theta_k^+ + \theta_k^-\right) \right\}.$$

This amounts to doubling the size of the optimization problem and adding $2d$ positivity constraints $\theta_k^+, \theta_k^- \geqslant 0$ for $1 \leqslant k \leqslant d$. These constraints are satisfied by projecting the updates of gradient methods onto the set of admissible values for θ^+, θ^-;

- *iterative soft thresholding* is based on proximal gradient strategies, which builds upon a framework to regularize non differentiable terms in a function to optimize;
- one can also rely on *continuation/homotopy methods*, based on the remark that $\theta_\lambda^\star$ is piecewise affine in λ. The idea is then to start from large values of λ, and build the path of solutions by computing break points one after the other.

Exercise 2.20 (Elastic net and generalized ridge regression). *Consider the minimization over $\theta \in \mathbb{R}^d$ of the following regularized empirical risk for linear least squares regression, when the inputs of the data points are characterized by a feature function $\varphi : \mathcal{X} \to \mathbb{R}^d$:*

$$\widehat{R}_{\lambda_1,\lambda_2}(\theta) = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \theta^\top \varphi(x_i)\right)^2 + \lambda_1 \|\theta\|_1 + \lambda_2 \theta^\top M \theta,$$

where $\lambda_1, \lambda_2 \geqslant 0$ are two regularization parameters, and M is a symmetric definite positive matrix. We assume that the design matrix $X \in \mathbb{R}^{n \times d}$ whose i-th row is $\varphi(x_i)^\top$ has full rank $d \leqslant n$.

(a) Show that $\widehat{R}_{\lambda_1,\lambda_2}$ admits a unique minimizer for any $\lambda_1, \lambda_2 \geqslant 0$.
(b) Give the expression of the minimizer for $\lambda_1 = 0$.
(c) Show that the function to minimize can be rewritten as

$$\widehat{R}_{\lambda_1,\lambda_2}(\theta) = \frac{1}{m} \left\| \widetilde{Y} - \widetilde{X}\theta \right\|_2^2 + \lambda_1 \|\theta\|_1,$$

where $m \in \mathbb{N}$, the matrix $\widetilde{X}$ and the vector $\widetilde{Y}$ should all three be made precise.
(d) Suggest a numerical method to minimize $\widehat{R}_{\lambda_1,\lambda_2}$ (briefly describe it in two lines, without entering into the details).

Exercise 2.21 ("Leave-one-in" estimators for regularized regression). *We consider performing linear regression of features $\varphi : \mathcal{X} \to \mathbb{R}^d$ in order to construct predictors $\theta^\top \varphi(x')$ for a test input x'. We work in the random design setting, where data points $(x_i, y_i)_{1 \leqslant i \leqslant n}$ in the train set are assumed to be independent and identically distributed according to some (unknown) probability distribution p_{data} on $\mathcal{X} \times \mathcal{Y}$, with $\mathcal{Y} = \mathbb{R}$. We assume that*

$$\int_{\mathcal{X}} \int_{\mathcal{Y}} \left(|y|^2 + \|\varphi(x)\|_2^2\right) p_{\mathrm{data}}(dx\, dy) < +\infty.$$

The associated training loss with regularization strength $\lambda > 0$ is

$$\widehat{R}_\lambda(\theta) = \frac{1}{n} \sum_{i=1}^{n} \left(y_i - \theta^\top \varphi(x_i)\right)^2 + \lambda \|\theta\|_\alpha^\alpha,$$

with $\alpha = 2$ (ridge regression) or $\alpha = 1$ (LASSO regression). We study a strategy to propose computationally inexpensive predictors, based on solving the following minimization problems for a single data point:

$$\widehat{\theta}_{\lambda,i} = \operatorname*{argmin}_{\theta \in \mathbb{R}^d} \widehat{R}_{\lambda,i}(\theta), \qquad \widehat{R}_{\lambda,i}(\theta) = \left(y_i - \theta^\top \varphi(x_i)\right)^2 + \lambda \|\theta\|_\alpha^\alpha;$$

and then averaging the associated prediction functions as

$$\widehat{f}(x') = \widehat{\theta}_{\mathrm{avg}}^\top \varphi(x'), \qquad \widehat{\theta}_{\mathrm{avg}} = \frac{1}{n} \sum_{i=1}^{n} \widehat{\theta}_{\lambda,i}.$$

This can be understood as some form of minibatching procedure at the level of the minimization problems.

(a) Show that, for $\theta \in \mathbb{R}^d$ fixed, it holds $\mathbb{E}\left[\widehat{R}_{\lambda,i}(\theta)\right] = \mathbb{E}\left[\widehat{R}_{\lambda}(\theta)\right]$ for any $1 \leqslant i \leqslant n$ (where the expectation is taken over realizations of $(x_i, y_i)_{1 \leqslant i \leqslant n}$), and that

$$\lim_{n \to +\infty} \widehat{R}_{\lambda}(\theta) = \mathbb{E}_{(x,y) \sim p_{\mathrm{data}}}\left[\left(y - \theta^\top \varphi(x)\right)^2\right] + \lambda \|\theta\|_\alpha^\alpha \qquad \text{a.s.}$$

(b) Compute $\widehat{\theta}_{\lambda,i}$ for $\alpha = 2$.

(c) Assume for this question that $d = 1$. Show that $\widehat{\theta}_{\mathrm{avg}}$ coincides with the ridge regression estimator when $\varphi(x_i)^2 = \varphi(x_j)^2$ for all $1 \leqslant i, j \leqslant n$.

(d) We fix $\alpha = 1$ and consider for simplicity the situation when $d = 1$ and $|y\varphi(x)| \leqslant R < +\infty$ almost surely. Compute $\widehat{\theta}_{\lambda,i}$. For which values of λ does one have $\widehat{\theta}_{\mathrm{avg}} = 0$ almost surely?

(e) In this last question, we change the loss function from $\ell(y,z) = (y-z)^2$ to $\ell(y,z) = |y-z|$ in order to perform robust regresssion, and still fix $\alpha = 1$, with $d = 1$ for simplicity. Solve the minimization problem

$$\widehat{\theta}_{\lambda,i} = \operatorname*{argmin}_{\theta \in \mathbb{R}} \left\{ \left| y_i - \theta^\top \varphi(x_i) \right| + \lambda |\theta| \right\},$$

and compute the associated average weight $\widehat{\theta}_{\mathrm{avg}}$.

2.4.5 Theoretical guarantees

In order to provide theoretical upper bounds on the excess risk for Lasso regression, we consider once again the fixed design setting, as in Sections 2.2.4 and 2.3.3. A key point of the analysis performed in this section (as for ridge regression in Section 2.3.3) is to carefully keep track of the dimensionality dependence, possibly upon going from "fast rates" $\mathrm{O}(1/n)$ to "slow rates" $\mathrm{O}(1/\sqrt{n})$.

We still assume for the analysis that (2.9) holds, for some $\theta_\star$ which is sparse – so that $\|\theta_\star\|_1$ remains bounded as the dimension d increases. We start by considering a general norm $\Omega(\theta)$ for the regularization term (recall that $\Omega(\theta) = \|\theta\|_2$ for ridge regression, and $\Omega(\theta) = \|\theta\|_1$ for Lasso; see Section 2.3.1). The dual norm is denoted by Ω^* and defined as

$$\Omega^*(z) = \sup\left\{ z^\top \theta \;\middle|\; \Omega(\theta) \leqslant 1 \right\}.$$

Exercise 2.22. *Fix $1 \leqslant p \leqslant +\infty$ and consider*

$$\Omega(\theta) = \|\theta\|_p = \left(\sum_{k=1}^d |\theta_k|^p \right)^{1/p}$$

for $1 \leqslant p < +\infty$ and, for $p = +\infty$,

$$\|\theta\|_\infty = \max_{1 \leqslant k \leqslant d} |\theta_k|.$$

Show that $\Omega^*(z) = \|z\|_{p'}$ where $1 \leqslant p' \leqslant +\infty$ is the exponent conjugate to p:

$$\frac{1}{p} + \frac{1}{p'} = 1.$$

Exercise 2.23. *Prove that, for any $u, v \in \mathbb{R}^d$, it holds $|u^\top v| \leqslant \Omega(u)\Omega^*(v)$.*

We need some preliminary results on the excess risk

$$\mathcal{R}\left(\widehat{\theta}_\lambda\right) - \mathcal{R}^\star = \frac{1}{n}\left\| X\left(\widehat{\theta}_\lambda - \theta_\star\right)\right\|_2^2 = \left(\widehat{\theta}_\lambda - \theta_\star\right)^\top \widehat{\Sigma}\left(\widehat{\theta}_\lambda - \theta_\star\right)$$

for Lasso regression (recall Exercise 2.5), where $\widehat{\theta}_\lambda$ is a minimizer[5] of $\theta \mapsto \dfrac{1}{2n}\|Y - X\theta\|_2^2 + \lambda\Omega(\theta)$.

[5] Mind the factor $1/2$ here. In contrast, the excess risk relies on the standard quadratic error. Overall this amounts to a rescaling of λ.

Lemma 2.1. *Suppose that* (2.9) *holds.*

(a) If $\Omega^*(X^\top\varepsilon) \leqslant n\lambda/2$, *then*

$$\Omega\left(\widehat{\theta}_\lambda\right) \leqslant 3\Omega(\theta_\star), \qquad \frac{1}{n}\left\|X\left(\widehat{\theta}_\lambda - \theta_\star\right)\right\|_2^2 \leqslant 3\lambda\Omega(\theta_\star).$$

(b) In all cases,

$$\frac{1}{n}\left\|X\left(\widehat{\theta}_\lambda - \theta_\star\right)\right\|_2^2 \leqslant \frac{4}{n}\|\varepsilon\|_2^2 + 4\lambda\Omega(\theta_\star).$$

Exercise 2.24. *The aim of this exercise is to prove Lemma 2.1.*

(a) Show that

$$\left\|X\left(\widehat{\theta}_\lambda - \theta_\star\right)\right\|_2^2 \leqslant 2\varepsilon^\top X\left(\widehat{\theta}_\lambda - \theta_\star\right) + 2n\lambda\Omega\left(\theta_\star\right) - 2n\lambda\Omega\left(\widehat{\theta}_\lambda\right).$$

(b) Prove the first bound in Lemma 2.1.
(c) Prove the second bound.

We can now apply the previous result to Lasso regression by choosing $\Omega(\theta) = \|\theta\|_1$. Then, $\Omega^*\left(X^\top\varepsilon\right) = \left\|X^\top\varepsilon\right\|_\infty$ is a maximum over the absolute values of $2d$ (possibly dependent) random variables $Z_k^\pm = \pm\left(X^\top\varepsilon\right)_k$ for $1 \leqslant k \leqslant d$. Classical results in probability theory (see for instance [4, Section 1.2.4]) show that this quantity scales as $\sqrt{\log(2d)}$ multiplied by the magnitude of the random variables at hand. Since $Z_k^\pm$ is the sum of n independent random variables, the Central Limit Theorem suggests that each random variable $Z_k^\pm$ has values of order $\sqrt{n}$. Overall, this suggests that $\Omega^*\left(X^\top\varepsilon\right)$ is of order $\sqrt{n\log(2d)}$ so that, in order to apply Lemma 2.1(a), one should choose

$$\lambda \sim \sqrt{\frac{\log(d)}{n}}.$$

In order to state a precise result, we denote by $\|\widehat{\Sigma}\|_\infty$ the largest element of $\widehat{\Sigma}$ in absolute value.

Proposition 2.5. *Suppose that* (2.9) *holds true, with* ε *a Gaussian vector with covariance* $\sigma^2\mathrm{Id}_n$. *Then, for*

$$\lambda = \frac{2\sigma}{\sqrt{n}}\sqrt{2\|\widehat{\Sigma}\|_\infty}\sqrt{\log(d) + \log\left(\frac{1}{\delta}\right)},$$

and

$$\widehat{\theta}_\lambda \in \operatorname*{argmin}_{\theta\in\mathbb{R}^d}\left\{\frac{1}{2n}\|Y - X\theta\|_2^2 + \lambda\Omega(\theta)\right\},$$

it holds, with probability larger or equal than $1-\delta$,

$$\mathcal{R}\left(\widehat{\theta}_\lambda\right) - \mathcal{R}^\star = \frac{1}{n}\left\|X\left(\widehat{\theta}_\lambda - \theta_\star\right)\right\|_2^2 \leqslant 6\|\theta_\star\|_1\frac{\sigma}{\sqrt{n}}\sqrt{2\|\widehat{\Sigma}\|_\infty}\sqrt{\log(d) + \log\left(\frac{1}{\delta}\right)}.$$

This result deserves various comments. The first point is that the bound provided by Proposition 2.5 is a so-called PAC bound (where PAC stands for "probably approximately correct"; see Section 10.3), *i.e.* it is a bound which holds with a high probability for the realizations under consideration. It can be considered as more useful than bounds in average as provided by Propositions 2.2 and 2.4 for ordinary least square regression and ridge regression, respectively, and [4, Exercise 8.8] for Lasso regression.

The dimension explicitly appears only through the a factor $\sqrt{\log(d)}$, which is a quite mild dependence, in particular if $n \gg \log(d)$. Note also that $\theta_\star$ needs to be sparse for the estimate to be useful (*i.e.* in order for $\|\theta_\star\|_1$ not to change as the dimension d of $\varphi(x)$ is increased).

Various extensions are possible, in particular obtaining bounds with "fast rates" $\mathrm{O}(1/n)$ for the average excess risk (see [4, Sections 8.3.3-8.3.4]), and extending the analysis to the random design setting (see [4, Section 8.3.5]).

Exercise 2.25. *The aim of this exercise is to prove Proposition 2.5.*

(a) Prove that, for $Z \sim \mathcal{N}(0,1)$ *and* $t \geqslant 0$*, it holds* $\mathbb{P}(|Z| \geqslant t) \leqslant \mathrm{e}^{-t^2/2}$*.*
(b) Using the latter inequality and the union bound on $\left\|X^\top \varepsilon\right\|_\infty$*, prove that*

$$\mathbb{P}\left(\Omega^*\left(X^\top \varepsilon\right) > \frac{n\lambda}{2}\right) \leqslant d \exp\left(-\frac{n\lambda^2}{8\sigma^2 \|\widehat{\Sigma}\|_\infty}\right).$$

(c) Conclude.

3

Classification with logistic regression

We discuss in this chapter show how to perform classification with a method called logistic regression. The presentation mainly focuses on binary classification, but we also mention how to perform multiclass classification. We start by providing in Section 3.1 a mathematical framework which allows to conviently minimize loss functions to perform classification; and then describe in Section 3.2 a method based on a linear model to actually perform classification.

3.1 Convexification of the loss

For simplicity of exposition, we restrict ourselves to binary classification on $\mathcal{Y} = \{-1, 1\}$ in this section. The loss function which naturally appears in classification problems (see Section 1.2.1.1) is the 0-1 loss. Minimizing the associated training loss is however a difficult problem as it is combinatorial in nature (one needs to consider the 2^n possibilities for the labels for binary classification for instance). We show in this section how the 0-1 loss for classification problems can be replaced by a surrogate convex loss function which upper bounds the 0-1 loss, and admits the same Bayes predictor as for the 0-1 loss. The interest of convex losses is that their optimization is easier to perform (see Chapter 4). Our presentation is based on [4, Section 4.1] and [40, Section 4.7].

Two main questions can be asked at this stage:

- which convex surrogate functions are admissible? This is discussed in Section 3.1.1;
- how does the risk associated with the convex surrogate inform us on the original risk? This is made precise in Section 3.1.2.

3.1.1 Convex surrogates and Φ-risk

We discuss in this section how to perform classification with real valued functions (see Section 3.1.1.1), in order to replace the 0-1 loss by a convex upper bound Φ (see Section 3.1.1.2). Admissible convex upper bounds should have the same Bayes predictor as the original problem, which motivates the concept of "classification calibrated" convex surrogates Φ (see Section 3.1.1.3).

3.1.1.1 Classification with real valued functions

Consider a real valued function $g : \mathcal{X} \to \mathbb{R}$. Denoting by $\mathcal{Y} = \{-1, 1\}$, binary classification can be performed using a function $f : \mathcal{X} \to \mathcal{Y}$ defined from g as $f = \text{sign}(g)$, *i.e.*

$$f(x) = \begin{vmatrix} 1 & \text{if} & g(x) > 0, \\ -1 & \text{if} & g(x) < 0, \\ 0 & \text{if} & g(x) = 0. \end{vmatrix}$$

Note that there is some arbitrariness in the choice of values for $f(0)$. We consider here $f(0) = 0$ so that the prediction is always wrong for $g(x) = 0$ (note that, strictly speaking, one would need to extend the output space to $\{-1, 0, 1\}$ instead of $\{-1, 1\}$). Another option would be to choose at random whether $f(x) = 1$ or -1 when $g(x) = 0$.

With some abuse of notation, we denote by $\mathcal{R}(g)$ the risk associated with g:

$$\mathcal{R}(g) = \mathbb{E}\left[\mathbf{1}_{f(X)\neq Y}\right], \qquad f = \text{sign}(g). \tag{3.1}$$

In order to make contact with generalizations of the latter risk, we rewrite it as

$$\mathcal{R}(g) = \mathbb{P}\left[\text{sign}(g(X)) \neq Y\right] = \mathbb{E}\left[\mathbf{1}_{Yg(X)} \leqslant 0\right] = \mathbb{E}\left[\Phi_{0-1}(Yg(X))\right], \tag{3.2}$$

where

$$\Phi_{0-1}(u) = \mathbf{1}_{u\leqslant 0}.$$

There are infinitely many Bayes predictors associated with the original risk (3.1). To derive one, we introduce, as in Section 1.2.2.1,

$$\eta(x') = \mathbb{P}\left(Y = 1 \,|\, X = x'\right). \tag{3.3}$$

Then,

$$\mathcal{R}(g) = \mathbb{E}\left[\eta(X)\mathbf{1}_{g(X)\leqslant 0} + (1 - \eta(X))\mathbf{1}_{g(X)\geqslant 0}\right].$$

A Bayes predictor $g^\star$ is therefore characterized by the conditions $g^\star(x) \leqslant 0$ if $\eta(x) \leqslant 1/2$ and $g^\star(x) \geqslant 0$ if $\eta(x) \geqslant 1/2$. One possible Bayes predictor is

$$g^\star(x) = \eta(x) - \frac{1}{2}.$$

Remark 3.1. *Note that the Bayes predictor $f^\star$ for the 0-1 loss is* $\text{sign}(g^\star)$*, in accordance with the result of Exercise 1.6.*

3.1.1.2 Convex surrogates

The key idea behind considering convex surrogates is to replace the function Φ_{0-1} appearing in the reformulation (3.2) of the original loss by a convex function $\Phi : \mathbb{R} \to \mathbb{R}_+$ and to minimize the Φ-risk

$$\mathcal{R}_\Phi(g) = \mathbb{E}\left[\Phi(Yg(X))\right]. \tag{3.4}$$

The function Φ should be an upper bound of the 0-1 loss:

$$0 \leqslant \Phi_{0-1} \leqslant \Phi. \tag{3.5}$$

Let us give three important examples:

- the *quadratic loss* $\Phi(u) = (u - 1)^2$: in this case $\Phi(yg(x)) = (y - g(x))^2$ since $y^2 = 1$. This situation corresponds to the standard regression paradigm, where the prediction is based on the sign of the function $g : \mathcal{X} \to \mathcal{Y}$ appearing in the regression problem. One possible issue of this approach is the overpenalization of large values of $|yg(x)|$;

- the *logistic loss* $\Phi(u) = \log(1+\mathrm{e}^{-u})/\log(2)$ (see Exercise 3.1 below): in this case,

$$\Phi(yg(x)) = -\frac{1}{\log(2)}\log\left(\frac{1}{1+\mathrm{e}^{-yg(x)}}\right) = -\frac{1}{\log(2)}\log\sigma(yg(x)), \tag{3.6}$$

  where σ is the sigmoid function

$$\sigma(v) = \frac{1}{1+\mathrm{e}^{-v}}. \tag{3.7}$$

  This is the convex surrogate used in Section 3.2 for logistic regression, without the normalization factor $\log(2)$;
- the *hinge loss* $\Phi(u) = \max(1-u, 0)$ used in Chapter 6 for classification with support vector machines. The *squared hinge loss* $\Phi(u) = \max(1-u,0)^2$ is sometimes used to have a smoother model.

Exercise 3.1. *Make precise the asymptotic behavior of the function* $\Psi(u) = \log(1+\mathrm{e}^{-u})$.

3.1.1.3 Φ-risk

The Φ-risk (3.4) can be rewritten as (by taking conditional expectations on Y)

$$\mathcal{R}_\Phi(g) = \mathbb{E}\left[\Phi(Yg(X))\right] = \mathbb{E}\left[\eta(X)\Phi(g(X)) + (1-\eta(X))\Phi(-g(X))\right].$$

Therefore,

$$\mathcal{R}_\Phi(g) = \mathbb{E}\left[C_{\eta(X)}(g(X))\right],$$

where we recall the definition (3.3) of η, and

$$C_\zeta(\alpha) = \zeta\Phi(\alpha) + (1-\zeta)\Phi(-\alpha).$$

A first requirement on the convex surrogate Φ is that the Bayes predictor for the Φ-risk leads to the same predictor as for the original risk $\mathcal{R}$, upon taking the sign. The Bayes predictor for the Φ-risk is characterized as follows:

$$\forall x \in \mathcal{X}, \qquad g^\star_\Phi(x) \in \operatorname*{argmin}_{\alpha\in\mathbb{R}} \Big\{\eta(x)\Phi(\alpha) + (1-\eta(x))\Phi(-\alpha)\Big\}. \tag{3.8}$$

The fact that this Bayes predictor coincides with the Bayes predictor for the original risk therefore translates into the following requirements:

$$\begin{aligned} \zeta > \frac{1}{2} &\iff \operatorname*{argmin}_{\alpha\in\mathbb{R}} C_\zeta(\alpha) \subset (0,+\infty), \\ \zeta < \frac{1}{2} &\iff \operatorname*{argmin}_{\alpha\in\mathbb{R}} C_\zeta(\alpha) \subset (-\infty,0), \end{aligned} \tag{3.9}$$

so that the prediction, which is given by the sign of the argmin, is 1 when $\zeta > 1/2$ and -1 when $\zeta < 1/2$. A function Φ which satisfies these requirements is said to be *classification-calibrated*. The next proposition gives simple sufficient conditions to this end when Φ is convex. Note that the convexity of Φ implies that C_ζ is convex for any $0 \leqslant \zeta \leqslant 1$.

Proposition 3.1. *Consider a convex function* $\Phi : \mathbb{R} \to \mathbb{R}$. *The function* Φ *is classification-calibrated (*i.e. (3.9) *is satisfied) if and only if* Φ *is differentiable at 0 and* $\Phi'(0) < 0$.

Exercise 3.2. *The aim of this exercise is to prove Proposition 3.1. Recall that a convex function from* $\mathbb{R}$ *to* $\mathbb{R}$ *is continuous and admits left/right derivatives at any point (see Exercise 3.3).*

(a) Prove that the condition (3.9) *is equivalent to*

$$\begin{cases} \zeta > \frac{1}{2} \iff C'_\zeta(0^+) = \zeta\Phi'(0^+) - (1-\zeta)\Phi'(0^-) < 0, \\ \zeta < \frac{1}{2} \iff C'_\zeta(0^-) = \zeta\Phi'(0^-) - (1-\zeta)\Phi'(0^+) > 0. \end{cases} \tag{3.10}$$

(b) Prove that Φ is classification calibrated when Φ is differentiable at 0 and $\Phi'(0) < 0$.
(c) Assume now that Φ is classification calibrated. Show first that $\Phi'(0^+) - \Phi'(0^-) \leqslant 0$, and then that (3.10) is satisfied.

Exercise 3.3. *Consider a convex function $\Phi : \mathbb{R} \to \mathbb{R}$. We prove in this exercise that the right derivative is well defined. The fact that the left derivative is well defined can be obtained by similar estimates.*

(i) Fix $x_0, x_1, x_2 \in \mathbb{R}$ with $x_0 < x_1 < x_2$. By considering x_1 as a convex combination of x_0, x_2, prove that

$$\frac{\Phi(x_1) - \Phi(x_0)}{x_1 - x_0} \leqslant \frac{\Phi(x_2) - \Phi(x_0)}{x_2 - x_0}.$$

(ii) Prove that the left hand side of the previous inequality is bounded from below uniformly in $x_1 \in (0, x_2]$.
(iii) Deduce that the right derivative of Φ at x_0 is well defined.

3.1.2 Relationship between original risk and Φ-risk

We have seen in Section 3.1.1.3 that the Bayes predictors for the Φ-risk and the original risk are the same. In practice, one minimizes the Φ-risk associated with the (classification calibrated) convex surrogate Φ, but is ultimately interested in performance guarantees on the original risk, based on the 0-1 loss. This motivates exploring the relationship between these two risks. Ideally, one would want an inequality of the form

$$0 \leqslant \mathcal{R}(g) - \mathcal{R}^\star \leqslant H\left(\mathcal{R}_\Phi(g) - \mathcal{R}_\Phi^\star\right),$$

where $\mathcal{R}_\Phi^\star$ is the risk associated with any Bayes predictor for $\mathcal{R}_\Phi$, and H is some increasing function such that $H(0) = 0$, called *calibration function*. In order to establish such as a result, we reformulate the excess risk using the Bayes predictor, and then upper bound this quantity with the Φ-risk.

Lemma 3.1. *For any Bayes predictor $g^\star : \mathcal{X} \to \mathbb{R}$ and any function $g : \mathcal{X} \to \mathbb{R}$ such that $\mathbb{P}(g(X) = 0) = 0$, it holds*

$$0 \leqslant \mathcal{R}(g) - \mathcal{R}^\star \leqslant \mathbb{E}\left[|2\eta(X) - 1| \mathbf{1}_{g(X)g^\star(X) \leqslant 0}\right]. \tag{3.11}$$

The condition $\mathbb{P}(g(X) = 0) = 0$ is a technical condition related to our choice for sign(0). Other conventions can be considered, in which case the result can be slightly different (see for instance [40, Lemma 4.5]).

Exercise 3.4. *The aim of this exercise is to prove Lemma 3.1. The idea is to rewrite the excess risk using conditional expectations as*

$$\mathcal{R}(g) - \mathcal{R}^\star = \mathbb{E}\left[\mathbb{E}\left(\mathbf{1}_{Y \neq \operatorname{sign}(g(X))} - \mathbf{1}_{Y \neq \operatorname{sign}(g^\star(X))} \,\middle|\, X\right)\right],$$

*and then consider the situations where the predictions are different, namely $\eta(X) > 1/2$ (*i.e. *$g^\star(X) > 0$) and $g(X) < 0$, or $\eta(X) < 1/2$ (*i.e. *$g^\star(X) < 0$) and $g(X) > 0$.*

(a) Show that the cases when $\eta(X) = 1/2$ or $g(X) = 0$ can indeed be discarded.
(b) Prove that $\mathbb{E}\left(\mathbf{1}_{Y \neq \operatorname{sign}(g(X))} - \mathbf{1}_{Y \neq \operatorname{sign}(g^\star(X))} \,\middle|\, X\right) = 2\eta(X) - 1$ when $\eta(X) > 1/2$ and $g(X) < 0$.
(c) Conclude.

Exercise 3.5. *Under the same conditions as in Lemma 3.1, prove that $0 \leqslant \mathcal{R}(g) - \mathcal{R}^\star \leqslant \mathbb{E}\left[|2\eta(X) - 1 - g(X)|\right]$.*

The upper bound of Lemma 3.1 can now be leveraged to obtain bounds on the excess risk in terms of the excess Φ-risk, by adapting [40, Theorem 4.7].

Theorem 3.1. *Consider a convex function $\Phi : \mathbb{R} \to \mathbb{R}$, which is nonincreasing on $\mathbb{R}_-$, and assume that there exist $s \geqslant 1$ and $c \in \mathbb{R}_+$ such that*

$$\forall x \in \mathcal{X}, \qquad \left|\eta(x) - \frac{1}{2}\right|^s \leqslant c^s \left[\Phi(0) - C_{\eta(x)}\left(g_\Phi^\star(x)\right)\right], \tag{3.12}$$

for some Bayes predictor $g_\Phi^\star$ associated with $\mathcal{R}_\Phi$ (recall (3.8)). Then, for any $g : \mathcal{X} \to \mathbb{R}$ such that $\mathbb{P}(g(X) = 0) = 0$,

$$0 \leqslant \mathcal{R}(g) - \mathcal{R}^\star \leqslant 2c\left(\mathcal{R}_\Phi(g) - \mathcal{R}_\Phi^\star\right)^{1/s}.$$

We refer to Exercise 3.8 below for examples of functions Φ satisfying the assumptions of this theorem. See also Exercise 3.6 for checking that the right hand side of (3.12) is indeed nonnegative, and Exercise 3.7 for the proof of this theorem.

Exercise 3.6. *Prove that $\Phi(0) - C_{\eta(x)}\left(g_\Phi^\star(x)\right) \geqslant 0$ for any $x \in \mathcal{X}$.*

Exercise 3.7. *The aim of this exercise is to prove Theorem 3.1.*

(a) Use (3.11) to prove that

$$0 \leqslant \mathcal{R}(g) - \mathcal{R}^\star \leqslant 2c\left(\mathbb{E}\left[\left(\Phi(0) - C_{\eta(x)}\left(g_\Phi^\star(x)\right)\right)\mathbf{1}_{g(X)g^\star(X)\leqslant 0}\right]\right)^{1/s}.$$

(b) Show that

$$\Phi(0)\mathbf{1}_{g(X)g^\star(X)\leqslant 0} \leqslant C_{\eta(x)}(g(x))\mathbf{1}_{g(X)g^\star(X)\leqslant 0}.$$

(c) Conclude the proof.

Exercise 3.8. *Prove that the following functions are classification-calibrated and satisfy the assumptions of Theorem 3.1:*

(1) $\Phi(u) = (1-u)^2$;
(2) $\Phi(u) = \max(1-u, 0)$;
(3) $\Phi(u) = \log(1 + \mathrm{e}^{-u})$ (note that we are not normalizing Φ here in order for (3.5) to hold).

For the latter case, introduce the function $\psi : [0,1] \to \mathbb{R}$ defined as

$$\psi(t) = \log(2) + t\log(t) + (1-t)\log(1-t) - \frac{1}{2}(2t-1)^2,$$

and note that $\psi(t) \geqslant 0$ for all $t \in [0,1]$.

Exercise 3.9. *Consider the function $\Phi(u) = \mathrm{e}^{-2u}\mathbf{1}_{u<0} + \mathrm{e}^{-u}\mathbf{1}_{u\geqslant 0}$. Show that Φ is convex and nonincreasing. Is it classification calibrated?*

Impact on approximation and estimation errors. Let us conclude this section by discussing two points and issues raised by the estimate of Theorem 3.1:

- For the same classification problem, several convex surrogates can be used... The predictor $g_\Phi^\star$ and the associated classification function $f_\Phi^\star = \operatorname{sign}\left(g_\Phi^\star\right)$ will however be different. This may have an impact on the approximation error via the class of functions which are considered (see Section 10.1).
- Note that $s = 2$ for smooth losses (a rather generic fact, see [4, Section 4.1.4]); while $s = 1$ for a non smooth surrogate such as the hinge loss. This means that the possibly nice convergence rate for $\mathcal{R}_\Phi(g) - \mathcal{R}_\Phi^\star$ will be degraded by the application of the square-root when considering the prediction function $f = \operatorname{sign}(g)$. On the other hand, smooth losses are easier to optimize, so that there is a trade-off to be found here.

3.2 Logistic regression

The presentation of logistic regression is based mostly on [41, Chapter 10] and [8, Section 4.3]. There are two ways to introduce the method for binary classification, by motivating it first as a discriminative probabilistic model (see Section 3.2.1), and then connecting this heuristic approach to the framework of Section 3.1 (see Section 3.2.2). In both case, predictions are based on a nonlinear function combined with an affine one, whose parameter need to be estimated, as we discuss in Section 3.2.3. We conclude the presentation in Section 3.2.4 by showing how to perform multiclass classification.

Remark 3.2. *Various extensions of the basic logistic regression can be found in the literature. See for instance [41, Chapter 10] for discussions on robust logistic regression (when outliers are present in the data set), Bayesian logistic regression (where the a posteriori distribution of θ is computed in order to quantify the uncertainty in the predictions), hierarchical classification, ...*

3.2.1 Binary logistic regression as a discriminative probabilistic model

The aim of discriminative probabilistic models[1] is to learn the conditional distributions of the data $\mathbb{P}(Y = c \,|\, X = x)$ for $c \in \mathcal{Y}$. We consider here binary classification with $\mathcal{Y} = \{0, 1\}$.

The idea behind logistic regression is to approximate the conditional distribution as

$$p_\theta(y = 1 \,|\, x) = \sigma\left(w^\top x + b\right), \qquad \theta = (w, b) \in \mathbb{R}^d \times \mathbb{R}, \tag{3.13}$$

where we recall that σ is the sigmoid function (3.7). Note that $p_\theta(y = 1 \,|\, x) \in [0, 1]$ thanks to the properties of the sigmoid function. A linear classifier is then obtained by predicting the label associated with the largest estimated conditional probability:

$$f_\theta(x) = \begin{cases} 1 & \text{if } p_\theta(y = 1 \,|\, x) > p_\theta(y = 0 \,|\, x), \\ 0 & \text{if } p_\theta(y = 1 \,|\, x) < p_\theta(y = 0 \,|\, x), \end{cases}$$

with some rule to break ties when $p_\theta(y = 1 \,|\, x) = p_\theta(y = 0 \,|\, x)$. This can be summarized as

$$f_\theta(x) = \mathbf{1}_{\left\{\log\left(\frac{p_\theta(y=1 \,|\, x)}{p_\theta(y=0 \,|\, x)}\right) > 0\right\}} = \mathbf{1}_{\{w^\top x + b > 0\}},$$

which corresponds to a historic model of classification known as the perceptron. The hyperplane $w^\top x + b = 0$ is called the decision boundary. The vector $w \in \mathbb{R}^d$ is normal to this hyperplane, while b is the offset from the origin.

Remark 3.3. *Note that we present here a model with a symmetric rule for classification,* i.e. *the threshold value of $p_\theta(y = 1 \,|\, x)$ for prediction is $1/2$. In some applications, asymmetric thresholds $p_\theta(y = 1 \,|\, x) > \delta$ should be considered;* e.g. *for spam classification it makes sense to consider $\delta > 1/2$ if $y = 1$ corresponds to spam emails, as one would prefer to not classify an email as spam rather than classifying as spam a genuine email. In this case the decision boundary is $w^\top x + b = \sigma^{-1}(\delta)$.*

It should be clear from the presentation that logistic regression allows to classify data which is linearly separable. For more general datasets, one needs to resort to some featurization function $\varphi(x)$ with values in $\mathbb{R}^D$, and consider the model

$$p_\theta(y = 1 \,|\, x) = \sigma\left(w^\top \varphi(x) + b\right), \qquad \theta = (w, b) \in \mathbb{R}^D \times \mathbb{R}.$$

[1] There are two classes of probabilistic models: discriminative and generative (see [41, Section 9.4] for a discussion on advantages and disadvantages of both approaches). We will see generative models later on. In essence, they model probability distributions $p_{\text{data}}(x, y)$ as $p_{\text{data}}(y)p_{\text{data}}(x|y)$, with $p_{\text{data}}(x|y)$ the generative part (obtaining new inputs from labels y). In contrast, discriminative models rely on the conditional probability $p_{\text{data}}(y|x)$ of the label y given the input x.

A simple example would be two-dimensional data points, the ones with labels 1 being enclosed in the unit disk, while the ones with labels 0 lie outside the disk. In this case, the featurization function $\varphi(x) = (x_1^2, x_2^2)$ would allow to linearly separate the dataset. In fact, in the general case, one should learn the optimal feature function on top of the parameters θ; for instance by using a neural network (see Chapter 8). The idea is that, if the (nonlinear) featurization function is sufficiently good, then a simple classification method based on a linear model, such as logistic regression, can be sufficient to obtain good prediction performances.

The steepness of the sigmoid is controlled by the magnitude of w (but does not depend on b). The larger $\|w\|_2$ is, the sharper the transition, which increases the risk of overfitting. The relative values of $p_\theta(y = 1 \,|\, x)$ and $p_\theta(y = 0 \,|\, x)$ can be used to give some indication on the confidence of the prediction.

The parameter θ to use for predictions is determined by maximizing the log-likelihood of the data for the Bernoulli model under consideration; see Section 3.2.3.

3.2.2 Binary logistic regression through a classification-calibrated convex surrogate

We reinterpret in this section the method as introduced in Section 3.2.1 in the light of the mathematical framework of Section 3.2.2. We consider here the situation when $\mathcal{Y} = \{-1, 1\}$ (which is not the setting considered in Section 3.2.1).

Let us first reformulate the model of Section 3.2.1 for $\mathcal{Y} = \{-1, 1\}$. Equation (3.13) remains unchanged, while (see Exercise 3.10 below)

$$p_\theta(y = -1 \,|\, x) = 1 - \sigma\left(w^\top \varphi(x) + b\right) = \sigma\left(-\left[w^\top \varphi(x) + b\right]\right).$$

This can be summarized as

$$p_\theta(y \,|\, x) = \sigma\left(y g_\theta(x)\right), \qquad g_\theta(x) = \theta^\top \phi(x), \tag{3.14}$$

where

$$\theta = (w, b) \in \mathbb{R}^d, \qquad \phi(x) = (\varphi(x), 1) \in \mathbb{R}^d, \qquad d = D + 1.$$

Let us next introduce the convex surrogate

$$\Phi(u) = \log\left(1 + \mathrm{e}^{-u}\right),$$

which is a classification-calibrated convex surrogate in view of Exercise 3.8. Moreover, recalling (3.6) (up to an unimportant multiplicative factor $\log 2$), we obtain, for some decision function g (not necessarily affine at this stage)

$$\Phi(yg(x)) = -\log \sigma(yg(x)).$$

The associated risk is

$$\mathcal{R}_\Phi(g) = \mathbb{E}\left[-\log \sigma(yg(x))\right],$$

so that the corresponding empirical risk for the affine decision function g_θ is

$$\widehat{\mathcal{R}}_{\Phi,n}(\theta) = -\frac{1}{n}\sum_{i=1}^{n} \log \sigma\left(y_i g_\theta(x_i)\right).$$

Minimizing the empirical Φ-risk can, in view of (3.14), be understood as maximizing the log-likelihood of the data in the framework of the discriminative model; see also (3.15) below.

3.2.3 Estimating θ for prediction

We come back to the setting of Section 3.2.1. In practice, one finds an estimation $\widehat{\theta} = (\widehat{w}, \widehat{b})$ of θ and performs predictions on unseen data points $x' \in \mathcal{X}$ based on $\mathbf{1}_{\{\widehat{\theta}^\top \phi(x') > 0\}}$. The value of $\widehat{\theta}$ is found by minimizing some empirical risk, possibly with the addition of some regularization term.

Logistic regression, as described in Section 3.2.1, relies on a Bernoulli model. Recall that for a Bernoulli random variable $Z \sim \mathcal{B}(p)$, the likelihood of observing the realization z for Z is $p^z(1-p)^{1-z}$. The negative log-likelihood of the training data set for the discriminative probabilistic model of Section 3.2.1 is therefore

$$\widehat{\mathcal{R}}_n(\theta) = -\frac{1}{n}\sum_{i=1}^{n} \log\Big(\sigma\left(g_\theta(x_i)\right)^{y_i}\left[1-\sigma\left(g_\theta(x_i)\right)\right]^{1-y_i}\Big) = \frac{1}{n}\sum_{i=1}^{n} H\Big(y_i, \sigma\left(g_\theta(x_i)\right)\Big), \tag{3.15}$$

with H the cross-entropy

$$H\left(y, \widehat{y}\right) = -y \log \widehat{y} - (1-y)\log\left(1-\widehat{y}\right). \tag{3.16}$$

The quantity $\widehat{\mathcal{R}}_n(\theta)$ should be minimized in order to find $\widehat{\theta}$. Let us however emphasize that, as discussed in Section 3.2.2, this can be equivalently seen as minimizing some empirical Φ-risk, which allows to understand logistic regression in the usual setting of supervised learning.

Exercise 3.10. *Prove the following statements:*

- $\sigma(-t) = 1 - \sigma(t)$ *for any* $t \in \mathbb{R}$*;*
- $\sigma' = \sigma(1-\sigma)$*;*
- *for* $z \in \{0,1\}$ *fixed, the function* $\psi_z : t \mapsto H(z, \sigma(t))$ *is convex on* $\mathbb{R}$*.*

Deduce that the function $h_i : \theta \mapsto H\Big(y_i, \sigma\left(g_\theta(x_i)\right)\Big)$ *is convex.*

In view of Exercise 3.10, the empirical risk (3.15) is convex. This implies that this function should be rather easy to minimize, and that gradient methods such as those presented in Chapter 4 should work fine. Newton or quasi-Newton methods could also be used. These two options are considered in the method implemented in `scikit-learn`.[2] The computation of the gradient and of the Hessian is made precise in the following exercise.

Exercise 3.11. *Compute the gradient and the Hessian of the function* $h_i : \theta \mapsto H\Big(y_i, \sigma\left(g_\theta(x_i)\right)\Big)$*. Deduce the expression of the gradient and Hessian of* $\widehat{\mathcal{R}}_n$*.*

In practice, it is usually beneficial to normalize the data in some way, for instance by standardizing it, or by using a min-max scaler, as explained in Section 1.1.4. For the same reasons as for linear least square regression (recall Sections 2.3 and 2.4), one should consider adding regularization terms whose magnitude is determined by cross validation. These regularization terms are typically $\|\theta\|_2^2$, or $\|\theta\|_1$ if the solution is expected to be sparse. When $\theta = (w, b)$ and the last component of the feature function ϕ is 1, in which case b is simply an offset that allows to recenter the data, it is customary to consider a regularization only on the part w, which determines the steepness of the decision function.

Exercise 3.12. *Consider the classification problem presented in Figure 3.1 below, with prediction based on logistic regression. The input variables are* $x_i \in \mathbb{R}^2$ *for* $i = 1, \ldots, n$*, with associated labels* $y_i \in \mathcal{Y} = \{-1, 1\}$ *(blue points correspond to the label value* -1*, while red points correspond to 1). The parameters used in logistic regression are* $\theta = (w_1, w_2, b) \in \mathbb{R}^3$*.*

(a) Recall the expression of the unregularized empirical training loss $\widehat{R}_n(\theta)$*, and the definition of the decision boundary.*

[2] See Section 1.1.11.3 of `https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression`

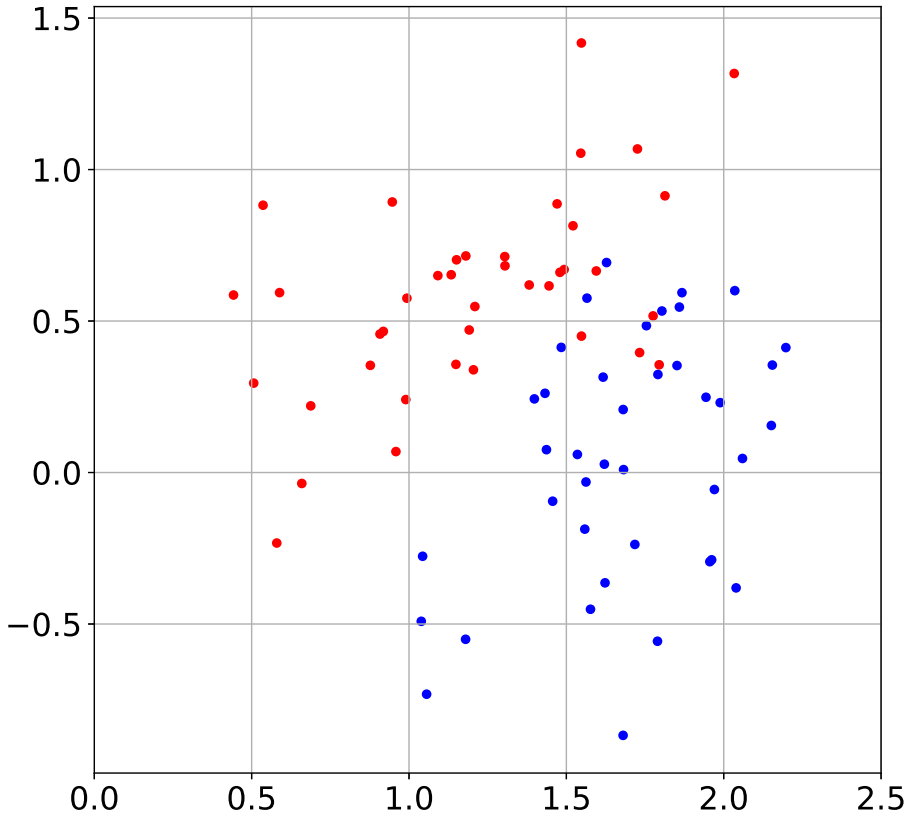

Fig. 3.1: Data set for Exercise 3.12.

(b) Sketch a possible decision boundary.

(c) We next add a regularization term to the loss function, and consider $\widehat{R}(\theta) + \lambda b^2$*. Sketch a possible decision boundary for large* λ*.*

(d) We next consider an alternative regularization term, and consider the regularized training loss $\widehat{R}_n(\theta) + \lambda w_1^2$*. Sketch a possible decision boundary for large* λ*.*

3.2.4 Multiclass logistic regression

We finally discuss how to use logistic regression to predict labels in situations where there are 3 labels or more, building upon the discussion in Section 1.2.1.1. More precisely, we assume that there are $K \geqslant 3$ classes, with $\mathcal{Y} = \{1, \dots, K\}$, and recall the expression of the softmax function, defined for $(a_1, \dots, a_K) \in \mathbb{R}^K$ as

$$S_K(a_1, \dots, a_K) = \left(\frac{\mathrm{e}^{a_1}}{\sum_{k=1}^K \mathrm{e}^{a_k}}, \frac{\mathrm{e}^{a_2}}{\sum_{k=1}^K \mathrm{e}^{a_k}}, \dots, \frac{\mathrm{e}^{a_K}}{\sum_{k=1}^K \mathrm{e}^{a_k}} \right) \in \mathbb{R}^K.$$

Predictions are based on the values of the decision function

$$G_\theta(x) = W\phi(x) + b \in \mathbb{R}^K, \qquad W \in \mathbb{R}^{K\times d}, \quad b \in \mathbb{R}^K,$$

by computing the k-th component of the vector S_K, namely

$$p_\theta(y = k \,|\, x) = S_{K,k}(G_\theta(x)).$$

The label y' predicted for a new input x' is

$$y' \in \underset{1\leqslant k\leqslant K}{\operatorname{argmax}}\, p_\theta\left(y = k \,|\, x'\right).$$

Remark 3.4. *The discussion in Remark 1.1 on the case $K = 2$ justifies a posteriori the choice of the sigmoid function in* (3.13) *for logistic regression in the binary case. In essence, the choice of this function can be traced back to the choice of the exponential function in the softmax to transform real numbers into positive ones.*

The loss function to consider for the training of the model is still the negative log-likelihood of the data, which generalizes (3.15). More precisely, the likelihood of the outputs $(y_1, \ldots, y_n)$ conditionally on the inputs $(x_1, \ldots, x_n)$ is

$$\prod_{i=1}^{n}\prod_{k=1}^{K} S_{K,k}(G_\theta(x_i))^{\underline{y}_{i,k}},$$

where $\underline{y}_i \in \{0,1\}^K$ is the one-hot encoding of y_i, *i.e.* the vector whose components are all equal to 0 except the y_i-th component which is equal to 1 (see the discussion in Section 1.1.4). Therefore, the empirical risk to consider reads

$$\widehat{\mathcal{R}}_n(\theta) = \frac{1}{n}\sum_{i=1}^{n} H_K\Big(\underline{y}_i, S_K\left(G_\theta(x_i)\right)\Big),$$

with H_K the cross-entropy

$$H_K\left(\underline{y}, \underline{\widehat{y}}\right) = -\sum_{k=1}^{K} \underline{y}_k \log \underline{\widehat{y}}_k. \tag{3.17}$$

Discussions similar to the ones written for binary logistic regression apply here concerning the training of the model (see [41, Section 10.3.2] for expressions of the gradient and the Hessian, the model being still convex) and its regularization.

4

(Stochastic) Gradients methods

We discuss in this chapter how to minimize (regularized) training losses to find the parameters $\widehat{\theta}$ allowing to perform predictions. This is done by using gradient methods, often stochastic gradient dynamics, and their extensions – including momentum methods. The reason why gradient methods are preferred is that the optimization problems under consideration are often set in very high dimensional spaces, which makes (quasi-)Newton methods impractical. Our presentation is based on [41, Chapter 8], [4, Chapter 5] and [50, Chapter 14].

We start by presenting the general framework of minimization problems in machine learning in Section 4.1, listing in particular properties of optimization problems encountered when training models, and discussing stochastic approaches where gradients are estimated with minibatches of the data points. We next analyze gradient descent in Section 4.2, in a manner amenable to adaptations to stochastic versions; and discuss extensions in Section 4.3, in particular based on the introduction of momentum variables. We finally present the main results of this chapter in Section 4.4, namely stochastic gradient dynamics and its momentum extensions such as the celebrated Adam method [31].

4.1 Minimization problems to solve and general strategy

We generically formulate the minimization problem to solve as

$$\widehat{\theta} \in \operatorname*{argmin}_{\theta \in \Theta} F(\theta), \tag{4.1}$$

for some function $F : \Theta \to \mathbb{R}$, which depends on the training set. Typically, $\Theta = \mathbb{R}^d$, so we are in the context of continuous optimization, even for classification problems (thanks to the use of surrogate functions as discussed in Section 3.1).

The form of this optimization problem is motivated in Section 4.1.1. We next discuss the various types of minimization problems considered in machine learning in Section 4.1.2. We finally distinguish in Section 4.1.3 between deterministic gradient methods, where the full training set is used, and stochastic methods, where the gradient is approximated over a randomly chosen fraction of the data set.

4.1.1 Formulation of the problem

We consider in this chapter the minimization of (regularized) training losses to find parameters used for predictions. More precisely,

$$\widehat{\theta}_\lambda \in \operatorname*{argmin}_{\theta\in\Theta} \left\{ \widehat{\mathcal{R}}_n(\theta) + \lambda\Omega(\theta) \right\}, \qquad \widehat{\mathcal{R}}_n(\theta) = \frac{1}{n}\sum_{i=1}^{n} \ell\left(y_i, f_\theta(x_i)\right), \tag{4.2}$$

where $\Omega : \mathcal{X} \to \mathbb{R}_+$ determines the regularization and $\lambda \geqslant 0$ is the strength of the regularization term. Predictions are then performed based on $f_{\widehat{\theta}_\lambda}$, for a value of λ chosen by (cross) validation, as discussed in Section 2.3.1. The problem (4.2) can be written as (4.1) upon setting $F = \widehat{\mathcal{R}}_n + \lambda\Omega$.

Remark 4.1. *As discussed in Section 2.3.1 as well, an alternative formulation of the minimization problem is*

$$\widehat{\theta}_D \in \operatorname*{argmin}_{\theta\in\Theta} \left\{ \widehat{\mathcal{R}}_n(\theta) \,\middle|\, \Omega(\theta) \leqslant D \right\}, \tag{4.3}$$

but we will restrict ourselves in this chapter to unconstrained problems of the form (4.2)*.*

A very important remark at this stage is that the aim is not to obtain the best minimizer to (4.1) (*i.e.* the best training error) but the smallest test error, in order to avoid overfitting. To elaborate on this statement, denote by

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

the risk associated with a predictor f, and assume that the minimization of the risk over predictors f_θ admits a global minimizer, namely

$$\mathcal{R}\left(f_{\theta_\star}\right) = \inf_{\theta\in\Theta} \mathcal{R}(f_\theta).$$

Consider $\lambda = 0$ for simplicity. Then, the risk associated with a minimizer $\widehat{\theta}$ of the optimization problem (4.2) can be decomposed as

$$\begin{aligned} \mathcal{R}\left(f_{\widehat{\theta}}\right) - \inf_{\theta\in\Theta} \mathcal{R}(f_\theta) &= \mathcal{R}\left(f_{\widehat{\theta}}\right) - \mathcal{R}\left(f_{\theta_\star}\right) \\ &= \underbrace{\mathcal{R}\left(f_{\widehat{\theta}}\right) - \widehat{\mathcal{R}}_n\left(f_{\widehat{\theta}}\right)}_{\text{estimation error}} + \underbrace{\widehat{\mathcal{R}}_n\left(f_{\widehat{\theta}}\right) - \widehat{\mathcal{R}}_n\left(f_{\theta_\star}\right)}_{\text{optimization error}} + \underbrace{\widehat{\mathcal{R}}_n\left(f_{\theta_\star}\right) - \mathcal{R}\left(f_{\theta_\star}\right)}_{\text{estimation error}}. \end{aligned} \tag{4.4}$$

This equality deserves several comments:

- first, the characterization of the quality of the optimization relies on function values (and not on the distance between approximate minimizers, magnitude of gradients, etc);
- it suffices to aim for an accuracy in the optimization procedure of the order of the estimation error, which is typically of order $1/\sqrt{n}$ (from an argument based on the Central Limit Theorem);
- concerning the optimization procedure, the key point is to ensure that the minimizer $\widehat{\theta}$ found by the numerical method is close enough to the best predictor $\theta_\star$ in the class, measured in terms of function values. This motivates looking at the validation loss during the minimization, as a proxy for the (test) loss. In practice, one should stop the optimization based on values of the validation loss, which corresponds to the procedure known as *early stopping*; see below.

Early stopping. We describe here the procedure known as early stopping, referring to [41, Section 13.5.1] for further references. The first step is to split the data into an actual training set and a validation set. Optimization is performed on the training set only. Typically, the training loss decreases over the iterations of the algorithm; while the validation loss initially decreases but then increases at some point, or at least stops decreasing. This is a sign that the minimization algorithm enters a regime where the training parameters are adjusted to fit the noise in the training data rather than the structure of the data points.

Early stopping consists in ending the optimization procedure when the validation loss no longer decreases, in order to avoid overfitting. In fact, other metrics than the validation loss can be used to assess the convergence, for instance the validation misclassification error. More precisely, one defines a certain number of epochs, called patience, after which the optimization is stopped if the validation loss has not further decreased (see Section 4.1.3 for the definition of the notion of an epoch). The final value of θ which is retained is the one for which the validation loss is minimal. This means that, in practical implementations of the method, one keeps track of the minimal value of the validation loss and the associated parameter, as well as the number of epochs for which the validation loss has not decreased.

4.1.2 Types of problems to solve

Let us list here various characteristics of the optimization problems encountered in machine learning:

- *convex vs. non-convex functionals*: the loss functions for some problems are convex, hence easier to minimize – for instance linear regression with Lasso (see Section 2.4) or logistic regression (see Section 3.2.3). Other problems are inherently non convex. A prominent example is the training of neural networks (see Chapter 8). For non convex problems, optimization methods typically converge to local minima.
  The theoretical analysis of optimization methods is often performed for (strongly) convex functionals, in order to write proofs and obtain clean statements. The associated numerical algorithms can be used for non convex problems, although there are no convergence guarantees in this case.
- *constrained vs. unconstrained problems*: most of the time we will consider unconstrained problems, where the functional to minimize includes some regularization term, as in (4.2). We will only briefly mention constrained problems such as (4.3) in very specific situations.
- *smooth vs. non-smooth optimization*: in certain situations, the function F to minimize in (4.1) is not continously differentiable, so that the gradient is not well defined everywhere. This is the case for instance for (4.2) with a regularization given by $\Omega(\theta) = \|\theta\|_1$, or if the loss function ℓ relies on the hinge loss (see Section 3.1.1.2). The function to minimize is however often the sum of a regular part and a non smooth one. One option to fall back on a smooth optimization problem is to rely on proximal methods (see for instance [41, Section 8.6]). Another option, which we consider here, is to replace gradients in the numerical methods at hand by subgradients (see below).

Sub-gradients. We recall here the notion of subgradients, and compute the subgradients of some functions appearing in machine learning problems. For a function $F : \mathbb{R}^d \to \mathbb{R}$, an element $G \in \mathbb{R}^d$ is a subgradient of F at $\theta \in \mathbb{R}^d$ if

$$\forall z \in \mathbb{R}^d, \qquad F(z) \geqslant F(\theta) + G^\top (z - \theta).$$

The set of subgradients at $\theta \in \mathbb{R}^d$ is called the subdifferential at θ, and is denoted by $\partial F(\theta)$. The subdifferential is non-empty for convex functions F. If F is moreover differentiable at $\theta \in \mathbb{R}^d$, then $\partial F(\theta) = \{\nabla F(\theta)\}$.

Exercise 4.1. *Compute the subdifferential of $F : \mathbb{R} \to \mathbb{R}_+$ defined by $F(\theta) = |\theta|$.*

Exercise 4.2 (Hinge loss). *Fix $y \in \mathbb{R}$ and $x \in \mathbb{R}^d$. Compute the subdifferential with respect to $\theta \in \mathbb{R}^d$ of the function $F(\theta) = \max\left(0, 1 - y\theta^\top x\right)$ based on the hinge loss.*

4.1.3 Stochastic vs. deterministic methods

The computation of the full gradient $\nabla F(\theta)$ for functions such as those appearing in (4.2) has a computational cost proportional to n because one needs to sum over all data points. Since approximate minimizers are sufficient (in view of the discussion after (4.4)), a precise gradient is not needed, in particular at the early stages of the minimization procedure. An unbiased estimate of the gradient may be sufficient.

Along some sequence of parameters $(\theta_t)_{t\geqslant 0}$ obtained with the optimization method at hand, we look for a stochastic approximation $\widehat{g}_t(\theta)$ of the gradient satisfying

$$\mathbb{E}\left[\widehat{g}_t(\theta_{t-1}) \,|\, \theta_{t-1}\right] = \nabla F(\theta_{t-1}). \tag{4.5}$$

Let us next describe three typical strategies to this end, for the specific case when F is the empirical training loss $\widehat{\mathcal{R}}_n$ appearing in (4.2):

(i) *stochastic approximation* is a historical strategy, dating back to Robbins–Monro, which corresponds to setting

$$\widehat{g}_t(\theta) = \nabla_\theta \ell\left(y_{I_t}, f_\theta(x_{I_t})\right),$$

where I_t are independent and identically distributed random variable with uniform distribution in $\{1,\dots,n\}$. It is clear in this context that (4.5) holds. The main interest of this approach is that the computation of the estimator of the gradient is unexpensive as a single data point is needed. On the other hand, the estimator may suffer from a large variance and therefore not be a good enough approximation of the gradient in order to drive the parameter to local minima of the loss function.

(ii) *minibatching* is an extension of stochastic averaging where several data points are chosen. More precisely, the method relies on sampling a random set $\mathcal{I}_t$ of m indices sampled at random from $\{1,\dots,n\}$, with or without replacement. The estimator of the gradient is

$$\widehat{g}_t(\theta) = \frac{1}{m}\sum_{i\in\mathcal{I}_t} \nabla_\theta \ell\left(y_i, f_\theta(x_i)\right).$$

The computational cost of this estimator is proportional to m. Note also that minibatching reduces to stochastic approximation when $m = 1$, and to a full gradient computation when $m = n$ and the sampling of the indices is performed without replacement. Further statistical properties of minibatching are studied in Exercise 4.3.

(iii) *shuffling* is often used in practice, although it does not allow to write (4.5). More precisely, the set of n data points is first randomly permutated, then decomposed in n/m batches of sizes m (assuming that the former number is an integer). One then computes the gradient over each of the n/m batches, and updates the parameter θ using some gradient method. This leads to the very important notion of *epoch*, which corresponds to the associated n/m elementary gradient steps. This means that one passes per construction once (and only once) over all data points in one epoch of this procedure; in contrast to estimators based on sampling with or without replacement with a minibatch of size m, for which a data point can be seen several times or none over n/m elementary steps.

Exercise 4.3 (Statistical properties of minibatching). *Denote the full gradient by*

$$g(\theta) = \frac{1}{n}\sum_{i=1}^{n} G_i(\theta), \qquad G_i(\theta) = \nabla_\theta \ell\left(y_i, f_\theta(x_i)\right) \in \mathbb{R}^d,$$

and consider the minibatched estimator the gradient for the random set $\mathcal{I}^m = \{i_1,\dots,i_m\} \subset \{1,\dots,n\}$*, namely*

$$\widehat{g}^m(\theta) = \frac{1}{m}\sum_{j=1}^{m} G_{i_j}(\theta).$$

Introduce also the empirical covariance

$$\Sigma(\theta) = \frac{1}{n-1}\sum_{i=1}^{n} \left(G_i(\theta) - g(\theta)\right)\left(G_i(\theta) - g(\theta)\right)^\top \in \mathbb{R}^{d\times d}.$$

(a) Show that $\mathbb{E}\left[\left(G_{i_j}(\theta) - g(\theta)\right)\left(G_{i_k}(\theta) - g(\theta)\right)^\top\right] = 0$ *when* $j \neq k$ *and sampling is performed uniformly in* $\{1,\dots,n\}$ *with replacement.*

(b) Deduce that $\mathrm{Cov}\left(\widehat{g}^m(\theta)\right) = \dfrac{1}{m}\left(1 - \dfrac{1}{n}\right)\Sigma(\theta)$ *when sampling is performed with replacement.*

(c) We next consider sampling without replacement for the end of this exercise. Show that

$$\mathrm{Var}\left(\mathbf{1}_{k\in\mathcal{I}^m}\right) = \frac{m}{n}\left(1 - \frac{m}{n}\right), \qquad \mathrm{Cov}\left(\mathbf{1}_{j\in\mathcal{I}^m}, \mathbf{1}_{k\in\mathcal{I}^m}\right) = -\frac{m(n-m)}{n^2(n-1)}.$$

(d) Prove that

$$\mathrm{Cov}\left(\widehat{g}^m(\theta)\right) = \frac{1}{m^2}\sum_{i=1}^{n} \widetilde{G}_i(\theta)\widetilde{G}_i(\theta)^\top \mathrm{Var}\left(\mathbf{1}_{i\in\mathcal{I}^m}\right) + \frac{1}{m^2}\sum_{i\neq j} \widetilde{G}_i(\theta)\widetilde{G}_j(\theta)^\top \mathrm{Cov}\left(\mathbf{1}_{i\in\mathcal{I}^m}, \mathbf{1}_{j\in\mathcal{I}^m}\right),$$

where $\widetilde{G}_i(\theta) = G_i(\theta) - g(\theta)$.

(e) Conclude that $\mathrm{Cov}\left(\widehat{g}^m(\theta)\right) = \dfrac{1}{m}\left(1 - \dfrac{m}{n}\right)\Sigma(\theta)$ *for sampling without replacement.*

4.2 Simple gradient descent

We study in this section gradient descent, which consists in iterating

$$\theta_t = \theta_{t-1} - \gamma_t \nabla F(\theta_{t-1}), \tag{4.6}$$

starting from some given value of θ_0, and using a sequence of learning rates or stepsize sequences $(\gamma_t)_{t\geqslant 0}$. These learning rates can be either kept constant, decay according to a given schedule, or found at each step by some line search procedure.

Remark 4.2. *When the function F to optimize is not smooth, the descent direction should be replaced by any element of the subgradient, namely*

$$\theta_t = \theta_{t-1} - \gamma_t v_{t-1}, \qquad v_{t-1} \in \partial F(\theta_{t-1}).$$

The motivation for gradient dynamics is that it can be considered as a time discretization of the continuous dynamics $\dot{\theta}(t) = -\nabla F(\theta(t))$, which is such that

$$\frac{d}{dt}\Big[F(\theta(t))\Big] = -|\nabla F(\theta(t))|^2 \leqslant 0.$$

The values of F are therefore nonincreasing along the gradient dynamics. There are of course many generalizations of gradient descent, such as conjugate gradient, but our aim here is to understand the algorithms used in machine learning such as stochastic gradient descent and its refinements and extensions.

We therefore provide in this section various convergence results for function values, in scenarios typical of machine learning applications; starting with the simplest case in Section 4.2.1, namely convex Lipschitz losses; and then discuss other settings in Section 4.2.2.

4.2.1 Analysis for a simple case: convex Lipschitz losses

The presentation in this section is taken from [50, Section 14.1]. We consider a convex and ρ-Lipschitz target F, and perform gradient descent with a fixed learning rate $\gamma > 0$. The aim is to obtain bounds on $F(\overline{\theta}_t) - F(\eta_\star)$, where $\eta_\star$ is a minimizer[1] of F, and

$$\overline{\theta}_t = \frac{1}{t}\sum_{s=0}^{t-1} \theta_s \tag{4.7}$$

is obtained by iterate averaging (this makes sense for convex functions, but should not be employed for non-convex ones; see also Section 4.2.2 for convergence results not relying on iterate averaging). Note that, by convexity,

$$F(\overline{\theta}_t) - F(\eta_\star) \leqslant \frac{1}{t}\sum_{s=0}^{t-1} F(\theta_s) - F(\eta_\star). \tag{4.8}$$

Moreover, still by the convexity of F, it holds $F(\theta) - F(\eta_\star) \leqslant (\theta - \eta_\star)^\top \nabla F(\theta)$, so that

$$F(\overline{\theta}_t) - F(\eta_\star) \leqslant \frac{1}{t}\sum_{s=0}^{t-1} (\theta_s - \eta_\star)^\top \nabla F(\theta_s).$$

The next lemma allows to bound the right hand side of the previous equality. We state it in a general manner, with descent directions v_s that need not be $\nabla F(\theta_s)$, as this result will also be used later on in Section 4.4.3 for stochastic gradient descent.

Lemma 4.1. *Consider an arbitrary sequence of vectors $v_0, \dots, v_{t-1} \in \mathbb{R}^d$, and a sequence of parameters iteratively defined from a given $\theta_0 \in \mathbb{R}^d$ as $\theta_t = \theta_{t-1} - \gamma v_{t-1}$. Then,*

$$\sum_{s=0}^{t-1} (\theta_s - \eta_\star)^\top v_s \leqslant \frac{\|\theta_0 - \eta_\star\|^2}{2\gamma} + \frac{\gamma}{2}\sum_{s=0}^{t-1} \|v_s\|^2. \tag{4.9}$$

In particular, if $\|v_s\| \leqslant \rho$ and $\gamma = \dfrac{\|\theta_0 - \eta_\star\|}{\rho\sqrt{t}}$, then

$$\frac{1}{t}\sum_{s=0}^{t-1} (\theta_s - \eta_\star)^\top v_s \leqslant \frac{\rho\|\theta_0 - \eta_\star\|}{\sqrt{t}}.$$

Note that the above result is not an "anytime" result as the learning rate depends on the horizon t and the bound is valid only for this time t.

Exercise 4.4. *The aim of this exercise is to prove Lemma 4.1. A key idea is to rewrite $(\theta_s - \eta_\star)^\top v_s$ as the difference of two terms at subsequent times in order to introduce a telescopic sum.*

(a) Prove that $(\theta_s - \eta_\star)^\top v_s = \dfrac{1}{2\gamma}\left(-\|\theta_{s+1} - \eta_\star\|^2 + \|\theta_s - \eta_\star\|^2\right) + \dfrac{\gamma}{2}\|v_s\|^2$ and deduce (4.9).

(b) Conclude the proof of the lemma.

We can conclude this section by applying the result of Lemma 4.1 for $v_s = \nabla F(\theta_s)$. Note that $\|\nabla F(\theta_s)\| \leqslant \rho$ when F is ρ-Lipschitz (this is easy to prove when $F \in C^1$, and can be extended to subgradients as discussed in [50, Lemma 14.7]).

Corollary 4.1. *Consider a convex and ρ-Lipschitz function F, and a learning rate $\gamma = \dfrac{\|\theta_0 - \eta_\star\|}{\rho\sqrt{t}}$. Then, for the simple gradient dynamics (4.6),*

$$F(\overline{\theta}_t) - F(\eta_\star) \leqslant \frac{\rho\|\theta_0 - \eta_\star\|}{\sqrt{t}}.$$

This means that, to achieve an error of at most $\varepsilon > 0$ in function values, it suffices to run gradient descent for $t \geqslant \rho^2\|\theta_0 - \eta_\star\|^2/\varepsilon^2$ steps.

[1] We do not denote the minimizer of F by $\theta_\star$ as this notation is kept for the argmin of the risk $\theta \mapsto \mathcal{R}(f_\theta)$, and not the empirical risk.

4.2.2 Other situations

We review in this section some of the results presented in [4, Section 5.2], to which we refer for a more complete presentation. The results we highligh are anytime results, which do not require iterate averaging:

- Consider $F \in C^1$ which is L-smooth and μ-strongly convex, namely

$$\forall \eta \in \mathbb{R}^d, \qquad \left| F(\eta) - F(\theta) + \nabla F(\theta)^\top (\eta - \theta) \right| \leqslant \frac{L}{2} \|\eta - \theta\|_2^2. \tag{LS},$$

  and

$$\forall \eta \in \mathbb{R}^d, \qquad F(\eta) \geqslant F(\theta) + \nabla F(\theta)^\top (\eta - \theta) + \frac{\mu}{2} \|\eta - \theta\|_2^2. \tag{SC}.$$

  One can prove that L is the largest eigenvalue of $\nabla^2 F$ (in modulus for non-convex functions) when $F \in C^2$, while μ is the smallest eigenvalues of $\nabla^2 F$. In this context, gradient descent with the learning rate $\gamma = 1/L$ leads to an exponential convergence in function values:

$$F(\theta_t) - F(\eta_\star) \leqslant \mathrm{e}^{-t/\kappa} \left(F(\theta_0) - F(\eta_\star) \right),$$

  where $\kappa = L/\mu \geqslant 1$ is a conditioning number; see Exercise 4.5. Typically, μ is small and of the order of the regularization strength for ridge regression for instance.
- When F is L-smooth, convex and $\gamma = 1/L$, then

$$F(\theta_t) - F(\eta_\star) \leqslant \frac{L}{2t} \|\theta_0 - \eta_\star\|^2.$$

- The convergence rate is smaller for less smooth functions F. When F is convex and ρ-Lipschitz continuous, and upon choosing

$$\gamma_t = \frac{\|\theta_0 - \eta_\star\|}{\rho \sqrt{t}},$$

  one can show that

$$\min_{0 \leqslant s \leqslant t-1} F(\theta_s) - F(\eta_\star) \leqslant \frac{\rho \|\theta_0 - \eta_\star\|}{2\sqrt{t}} (2 + \log t).$$

  This result is the counterpart of the estimates of Corollary 4.1 when no averaging of the parameter is performed. Note that iterate averaging may be a better option for convex targets if the function F itself is expensive to evaluate (so that one would not want to compute its values at all timesteps; this is particularly true for stochastic methods where minibatching is used in order to avoid going over all data points).

Exercise 4.5 (Convergence of gradient methods for strongly convex functions). *We analyse in this exercise the convergence of gradient descent for C^2 functions which are L-smooth and μ-strongly convex. Recall that the notation $A \leqslant B$ for two symmetric matrices means that $B - A$ is positive.*

(a) Show that a function $F \in C^2(\mathbb{R}^d, \mathbb{R})$ satisfying $\nabla^2 F \geqslant \mu \mathrm{Id}$ is μ-strongly convex.
(b) Show that if $F \in C^2(\mathbb{R}^d, \mathbb{R})$ is convex, then $G_\lambda = F + \lambda \| \cdot \|_2^2$ is μ-strongly convex for some parameter μ to be made precise.
(c) Prove that if F is μ-strongly convex, then there exists a unique solution to the problem $\min_{\theta \in \mathbb{R}^d} F$.
(d) Show that if $F \in C^2(\mathbb{R}^d, \mathbb{R})$ satisfies $-L\mathrm{Id} \leqslant \nabla^2 F \leqslant L\mathrm{Id}$ for some $L \in \mathbb{R}_+$, then F is L-smooth.

We assume in the remainder of the exercise that the function F is $C^2(\mathbb{R}^d, \mathbb{R})$, μ-strongly convex and L-smooth, and denote by $\eta_\star$ its unique minimizer. We also define $\kappa = L/\mu$.

(e) Prove that $\kappa \geqslant 1$.
(f) Prove that $\|\nabla F(\theta)\| \geqslant 2\mu(F(\theta) - F(\eta_\star))$ (Hint: optimize the right hand side of (SC) with respect to η).
(g) Show that the iterates of gradient descent with step $\gamma = 1/L$ satisfy

$$F(\theta_t) \leqslant F(\theta_{t-1}) - \frac{1}{\kappa} (F(\theta_{t-1}) - F(\eta_\star)).$$

(h) Conclude that $0 \leqslant F(\theta_t) - F(\eta_\star) \leqslant \mathrm{e}^{-t/\kappa} (F(\theta_0) - F(\eta_\star))$.

4.3 Deterministic methods beyond simple gradient descent

Gradient descent methods are often not used as such in practice, because their convergence can be quite slow. There are two principal ways to accelerate them in applications of machine learning:

(i) rely on second order methods such as Newton and quasi-Newton methods, although this may require to compute the Hessian (often computationally too expensive) or store the gradients over several iterations (which can require a large storage capacity when the parameters are of high dimension as for deep neural networks);
(ii) introduce some momentum in the optimization method.

The current practice in machine learning is more towards the second approach, and we will therefore present only methods related to this option. The key idea is to introduce a new variable ω, thought of as a momentum (mass times velocity in physics), and to go from the non-inertial dynamics $\dot\theta(t) = -\nabla F(\theta(t))$ to the inertial dynamics

$$\begin{cases} \dot\theta(t) = \omega(t), \\ \dot\omega(t) = -\nabla F(\theta(t)) - \xi\omega(t), \end{cases} \tag{4.10}$$

where $\xi > 0$ is some friction coefficient. The inertia inherent in the dynamics motivates the terminology "heavy ball method" which is sometimes used to refer to this class of evolution equations. In fact, (4.10) corresponds to Polyak averaging (which dates back to 1964), and can also be seen as some Hamiltonian dynamics with added dissipation through viscous friction. In particular, denoting by H the Hamiltonian,

$$\frac{dH(\theta(t),\omega(t))}{dt} = -\xi\omega(t)^2 \leqslant 0, \qquad H(\theta,\omega) = F(\theta) + \frac{1}{2}\omega^2.$$

Let us first further elaborate on the notion of inertia, and motivate that (4.10) can be seen as some gradient dynamics with memory. We integrate the second equation in (4.10) by noting that

$$\frac{d}{dt}\left(\mathrm{e}^{\xi t}\omega(t)\right) = -\mathrm{e}^{\xi t}\nabla F(\theta(t)),$$

so that

$$\omega(t) = \mathrm{e}^{-\xi t}\omega(0) - \int_0^t \mathrm{e}^{-\xi(t-s)}\nabla F(\theta(s))\, ds.$$

Therefore, for $\omega(0) = 0$ (or for t large, since the first term disappears in this limit),

$$\dot\theta(t) = -\int_0^t \mathrm{e}^{-\xi(t-s)}\nabla F(\theta(s))\, ds.$$

This should be compared to gradient dynamics where $\nabla F(\theta(t))$ appears on the right hand side. It is clear from the latter equation that (4.10) can be considered as some gradient dynamics where the drift is averaged in time.

In order to relate the dissipative Hamiltonian dynamics (4.10) to the gradient dynamics, note that one can rewrite the second equation as

$$\ddot\theta(t) + \xi\dot\theta(t) = -\nabla F(\theta(t)),$$

and introduce the time-rescaled solution $\theta_\xi(t) = \theta(\xi t)$ to obtain (see Exercise 4.6)

$$\theta_\xi(t) = \theta_\xi(0) - \int_0^t \nabla F\left(\theta_\xi(s)\right)\, ds + \mathrm{O}\left(\frac{1}{\xi}\right). \tag{4.11}$$

This motivates that the limiting dynamics $\theta_\infty(t)$ is a solution to the non inertial dynamics.

Exercise 4.6. *Prove the equality* (4.11).

Nesterov accelerated gradient. Appropriate discretizations of (4.10) lead to numerical methods with decay rates $1/t^2$ for smooth (non strongly) convex functions, instead of the slower rate $1/t$ obtained with gradient methods; see [4, Exercise 5.15]. This motivates the terminology "accelerated", in particular given that the rate $1/t^2$ is known to be optimal. Nesterov's method is one such instance of a discretization of (4.10). It can be formulated as (see for instance [52, Appendix A.1])

$$\begin{cases} \omega_{t+1} = \beta\omega_t - \gamma\nabla F(\theta_t + \beta\omega_t), \\ \theta_{t+1} = \theta_t + \omega_{t+1}. \end{cases} \tag{4.12}$$

This discrete time evolution can be rewritten in order to make the relationship with (4.10) more apparent. One introduces to this end $\omega_t = \sqrt{\gamma}\widehat{\omega}_t$ and writes $\beta = \mathrm{e}^{-\xi\sqrt{\gamma}}$, so that

$$\begin{cases} \widehat{\omega}_{t+1} = \widehat{\omega}_t + \dfrac{\beta-1}{\sqrt{\gamma}}\sqrt{\gamma}\widehat{\omega}_t - \sqrt{\gamma}\nabla F\left(\theta_t + \beta\sqrt{\gamma}\widehat{\omega}_t\right), \\ \theta_{t+1} = \theta_t + \sqrt{\gamma}\widehat{\omega}_{t+1}, \end{cases}$$

or equivalently

$$\begin{cases} \dfrac{\widehat{\omega}_{t+1} - \widehat{\omega}_t}{\sqrt{\gamma}} = -\nabla F\left(\theta_t + \beta\sqrt{\gamma}\widehat{\omega}_t\right) - \dfrac{1-\beta}{\sqrt{\gamma}}\widehat{\omega}_t, \\ \dfrac{\theta_{t+1} - \theta_t}{\sqrt{\gamma}} = \widehat{\omega}_{t+1}, \end{cases}$$

which corresponds to some discretization of (4.10) with a time step $\sqrt{\gamma}$ since $(1-\beta)/\sqrt{\gamma} = \xi + \mathrm{O}(\sqrt{\gamma})$. In practice, γ and β have to be carefully chosen so that the dynamics has a stable behavior. The above derivation, which suggests that $\beta = \mathrm{e}^{-\xi\sqrt{\gamma}}$ for some friction coefficient $\xi > 0$, provides some guidelines to this end.

4.4 Stochastic gradient descent and its extensions

We turn in this section to stochastic gradient dynamics (SGD) and their variations. These methods are currently the most commonly used to minimize empirical risk functions, and result from replacing $\nabla F(\theta)$ in gradient dynamics by some stochastic approximation, typically obtained via minibatching, as described in Section 4.1.3. More precisely, we consider the dynamics

$$\theta_t = \theta_{t-1} - \gamma_t \widehat{g}_t(\theta_{t-1}), \qquad \mathbb{E}\left[\widehat{g}_t(\theta_{t-1}) \,|\, \theta_{t-1}\right] = \nabla F(\theta_{t-1}). \tag{4.13}$$

One can rewrite the estimator of the gradient as

$$\widehat{g}_t(\theta) = \nabla F(\theta) + \Sigma(\theta)^{1/2} Z_t, \qquad \Sigma(\theta) = \mathbb{E}\left[\left(\widehat{g}_t(\theta) - \nabla F(\theta)\right)\left(\widehat{g}_t(\theta) - \nabla F(\theta)\right)^\top\right] \in \mathbb{R}^{d\times d},$$

where $Z \in \mathbb{R}^d$ is a random vector with mean 0 and identity covariance. Typically, $\Sigma(\theta)$ is of order $1/m$ when considering minibatching with batches of size m. When the stepsize is fixed to some value $\gamma > 0$, one can show that the discrete dynamics (4.13) is similar in behavior to solutions of the following stochastic differential equation observed at times multiples of γ (see for instance [49]):

$$d\theta_s = -\nabla F(\theta_s)\, ds + \sqrt{\gamma}\Sigma(\theta_s)^{1/2} dW_s. \tag{4.14}$$

This corresponds to gradient dynamics perturbed by some noise, whose magnitude diminishes as the stepsize is reduced or the minibatch size m is increased.

4.4.1 Choice of stepsizes/learning rates

The timestep/learning rate in (4.13) should be chosen as a compromise between not being too small, otherwise the dynamics takes too much time to converge, but also not being too large,

otherwise there is too much gradient noise and the dynamics may be unstable (similarly to what happens for standard gradient dynamics). A practical approach to setting the timestep consists in starting with a small learning rate, performing the minimization for a (small) subset[2] of the data for a fixed (rather small) number of steps, and then increasing the timestep until the best value achieved for each learning rate starts to increase. The timestep will then be chosen to be of the order of magnitude of the timestep leading to the smallest value of the target function for the chosen subset of data, possibly reduced a bit for cautiousness, in order to take into account that the prediction for the optimal timestep may not be fully reliable as only a subset of the data was used.

Some practitioners advocate using learning schedules. A first option is to resort to decreasing timesteps, inspired by proofs of convergence in the context of stochastic approximation algorithms à la Robbins–Monro [47]. Typical conditions on the learning rates are

$$\sum_{t\geqslant 1}\gamma_t^2 < +\infty, \qquad \sum_{t\geqslant 1}\gamma_t = +\infty,$$

which can be achieved for instance with $\gamma_t = t^{-\alpha}$ with $1/2 < \alpha \leqslant 1$. Another option is to consider optimization in various stages where the learning rate is kept constant, then decreased after a certain number of iterations (either fixed or depending on the behavior of the values of the function to optimize). For deep learning, a practical recommendation can be to start with initially small learning rates as one may be far away from a local minimum, so that the gradient of the loss function can be large; then to increase the learning rate when convergence starts (*i.e.* as the gradient of the function to minimize starts to decrease in magnitude) in order to favor a better exploration of the loss landscape; and finally reduce the learning rate at the end to fine tune the convergence to a clearly defined local minimum.

4.4.2 Variance reduction

It can be beneficial to reduce the minibatching noise in order to accelerate the convergence (from a theoretical perspective, go from convergence rates of order $1/t$, typical of stochastic gradient dynamics, to exponential convergence rates for gradient dynamics on strongly convex targets). An idea to this end is to rely on control variate techniques where the estimator for the gradient is replaced by

$$\widetilde{g}_t(\theta) = \widehat{g}_t(\theta) + \nabla F(\widetilde{\theta}) - \widehat{g}_t(\widetilde{\theta}),$$

with $\widetilde{\theta}$ fixed and $\nabla F(\widetilde{\theta})$ computed exactly. Note that the same minibatch is used for the two estimators on the right hand side of the previous equality. By construction, $\widetilde{g}_t$ is an unbiased estimator of ∇F. Moreover,

$$\widetilde{g}_t(\theta) = \nabla F(\widetilde{\theta}) + \frac{1}{m}\sum_{i\in\mathcal{I}_t}\left[\nabla_\theta \ell\left(y_i, f_\theta(x_i)\right) - \nabla_\theta \ell\left(y_i, f_{\widetilde{\theta}}(x_i)\right)\right],$$

so that the random part of the estimator comes only from the second term, which is small when θ is close to $\widetilde{\theta}$ (typically with a variance of order $\|\theta-\widetilde{\theta}\|^2$). In practice, this is done by computing the exact gradient for certain iterates θ_{kt_0} (with t_0 some fixed period) and using the control variate for the next gradient steps $kt_0 \leqslant t \leqslant (k+1)t_0$. This method is known as "stochastic variance reduced gradient" (SVRG). A popular extension of this approach is provided by SAGA ("stochastic averaged gradient accelerated"), where $\nabla F(\widetilde{\theta})$ is not computed except for the initial condition, and the control variate is obtained by combining this single exact gradient and an average over the estimators of the gradient in subsequent steps.

[2] More generally, cross-validation can be done with a fraction of the data only to coarsely identify interesting regions of parameter space, before fine tuning with more (all) data points.

4.4.3 Theoretical analysis

The convergence analysis of the SGD method (4.13) is similar to the derivation performed in Section 4.2.1 for deterministic gradient dynamics, with expectations at well chosen places. These expectations are with respect to realizations of the sampling noise in the estimator of the gradient, typically through the choice of the minibatch (which is conditionally independent of θ_{t-1}). We rely in particular on Lemma 4.1, where the vectors v_t are replaced by the estimators $\widehat{g}_{t+1}(\theta_t)$ of the gradient.

Theorem 4.1. *Assume that F is convex, and that, for all $s \geqslant 1$,*

$$\mathbb{E}\left[\|\widehat{g}_s(\theta_{s-1})\|^2\right] \leqslant \rho^2, \qquad \mathbb{E}\left[\widehat{g}_s(\theta_{s-1}) \,|\, \theta_{s-1}\right] = \nabla F(\theta_{s-1}). \tag{4.15}$$

Fix $t \geqslant 1$ and consider the learning rate

$$\gamma = \frac{\|\theta_0 - \eta_\star\|}{\rho\sqrt{t}}.$$

Then, the following convergence result holds for the averaged parameter $\overline{\theta}_t$ defined in (4.7)*:*

$$0 \leqslant \mathbb{E}\left[F(\overline{\theta}_t)\right] - F(\eta_\star) \leqslant \frac{\rho\|\theta_0 - \eta_\star\|}{\sqrt{t}}.$$

Exercise 4.7. *The aim of this exercise is to prove Theorem 4.1. Use* (4.8) *to prove that*

$$\mathbb{E}\left[F(\theta_s)\right] - F(\eta_\star) \leqslant \mathbb{E}\left[(\theta_s - \eta_\star)^\top \widehat{g}_{s+1}(\theta_s)\right],$$

and conclude with Lemma 4.1.

Let us conclude this section with some remarks. First, it is a good practice to perform parameter averaging only after some burn-in period, when the current values of the parameters are rather close from each other. This avoids having some bias related to initially irrelevant values of the parameter. Let us next note that the convergence estimate which is obtained for SGD is similar to the one obtained for gradient dynamics (with some adaptation in the assumptions, in the sense that the inequality in (4.15) replaces the condition that F is ρ-Lipschitz), but for a numerical method which is much cheaper as only (possibly crude) unbiased estimators of the full gradient are needed. This motivates very strong gains in performance when using SGD.

Exercise 4.8. *We provide in this exercise a sufficient condition for the first condition in* (4.15) *to hold. More precisely, consider minibatching for the loss function $\widehat{\mathcal{R}}_n$ in* (4.2)*, and assume that*

$$\forall i \in \{1, \dots, n\}, \qquad \|\nabla_\theta \ell\left(y_i, f_\theta(x_i)\right)\| \leqslant \rho.$$

Prove that the first condition in (4.15) *holds and that $F = \widehat{\mathcal{R}}_n$ is ρ-Lipschitz.*

Exercise 4.9 (Stochastic gradient dynamics in the quadratic case). *We consider a symmetric positive definite matrix $H \in \mathbb{R}^{d\times d}$ and a vector $a \in \mathbb{R}^d$, and introduce the following quadratic functional with argument $\theta \in \mathbb{R}^d$:*

$$F(\theta) = \frac{1}{2}\theta^\top H\theta - a^\top\theta.$$

This function is minimized using iterates of the stochastic gradient dynamics with fixed stepsize $\gamma > 0$:

$$\theta_t = \theta_{t-1} - \gamma\left(\nabla F(\theta_{t-1}) + \varepsilon_t\right),$$

where $(\varepsilon_t)_{t\geqslant 1}$ are independent and identically distributed d-dimensional random variables with mean 0 and covariance $\mathbb{E}(\varepsilon\varepsilon^\top) = C \in \mathbb{R}^{d\times d}$. The timestep γ is chosen to be strictly smaller than $1/L$, with L the largest eigenvalue of H. We also denote by $\mu > 0$ the smallest eigenvalue of H.

(a) Show that F admits a unique minimizer $\eta_\star$, and give the expression of $\eta_\star$ in terms of H, a.
(b) Prove that $F(\theta) - F(\eta_\star) = \frac{1}{2}(\theta - \eta_\star)^\top H(\theta - \eta_\star)$.
(c) Identify the matrix N_γ such that $\theta_t - \eta_\star = N_\gamma(\theta_{t-1} - \eta_\star) - \gamma\varepsilon_t$.
(d) Give the expressions of the matrices M_1, M_2 for which

$$\mathbb{E}\left[F(\theta_t) - F(\eta_\star)\right] = \frac{1}{2}\mathbb{E}\left[(\theta_{t-1} - \eta_\star)^\top M_1(\theta_{t-1} - \eta_\star)\right] + \mathbb{E}\left[\varepsilon_t^\top M_2 \varepsilon_t\right].$$

(e) Deduce that $\mathbb{E}\left[F(\theta_t) - F(\eta_\star)\right] \leqslant \rho_\gamma \mathbb{E}\left[F(\theta_{t-1}) - F(\eta_\star)\right] + \delta_\gamma$ *for* $\rho_\gamma = (1-\mu\gamma)^2$ *and a constant δ_γ to be made precise.*
(f) Conclude that, for $\gamma \leqslant \gamma_\star$ with $\gamma_\star > 0$ small enough,

$$\mathbb{E}\left[F(\theta_t) - F(\eta_\star)\right] \leqslant \mathrm{e}^{-2\mu\gamma t}\mathbb{E}\left[F(\theta_0) - F(\eta_\star)\right] + \frac{\gamma}{2\mu}\mathrm{Tr}\left(H^\top C\right).$$

4.4.4 Momentum versions

Let us conclude this section by presenting momentum extensions of SGD, which also include some form of preconditioning in order to tackle the anisotropy of the loss landscape. For gradient dynamics, a preconditioning consists in considering the dynamics

$$\dot{\theta}(t) = -M^{-1}\nabla F(\theta(t)).$$

When $F(\theta) = \theta^\top S\theta/2$, a good preconditioner is $M = S$ as the dynamics then reduces to $\dot{\theta}(t) = -\theta(t)$, for which all components evolve on the same timescale. In general, a convenient choice of preconditioning for θ close to a local minimum $\eta_\star$ is $M = \nabla^2 F(\eta_\star)$ since the gradient of the loss function can be locally approximated as $\nabla F(\theta) \approx \nabla^2 F(\eta_\star)(\theta - \eta_\star)$. In practice, inverting the Hessian or an approximation of it can be quite expensive, so that diagonal preconditioners are typically considered.

Adam [31], introduced in 2014, is currently the most common optimization algorithm in machine learning, in particular for large scale neural network models. The name is derived from "adaptive moment estimation", as it is a momentum extension of SGD, with an adaptive procedure which provides some form of diagonal preconditioning. More precisely, the update of the parameter is performed in a componentwise manner as

$$\theta_{t,k} = \theta_{t-1,k} - \gamma\frac{v_{t-1,k}}{\varepsilon + \sqrt{s_{t-1,k}}}, \tag{4.16}$$

with

$$v_t = \beta_1 v_{t-1} + (1-\beta_1)\widehat{g}_t(\theta_{t-1}), \qquad s_{t,k} = \beta_2 s_{t-1,k} + (1-\beta_2)\widehat{g}_{t,k}(\theta_{t-1})^2,$$

where $\varepsilon > 0$ is some (small) regularization parameter, and $\beta_1, \beta_2 \in [0,1]$. The vector v represents some averaged sum of the estimated gradients, which allows to introduce some inertia in the dynamics. The vector s_t performs some time averaging of the second moment of the components of the gradients, which allows to renormalize the magnitude of the gradient in order to have updates in the parameters of order γ somewhat independently of whether the dynamics is in a region of small or large gradients. In fact, as the variables v and s are initialized as $v_0 = 0$ and $s_0 = 0$, it is customary to correct for the bias incurred by this initialization by considering

$$\widehat{v}_t = \frac{v_t}{1-\beta_1^t}, \qquad \widehat{s}_t = \frac{s_t}{1-\beta_2^t}, \tag{4.17}$$

and update the parameter as

$$\theta_{t,k} = \theta_{t-1,k} - \gamma\frac{\widehat{v}_{t-1,k}}{\varepsilon + \sqrt{s_{t-1,k}}}.$$

See Exercise 4.10 for further motivation.

Remark 4.3. *In fact, as shown in [14], one can consider* (4.16) *as a time discretization of the continuous dynamics*

$$\begin{cases} \dot{\theta}(t) = -\dfrac{m(t)}{\varepsilon + \sqrt{v(t)}}, \\ \dot{m}(t) = \dfrac{\nabla f(\theta(t)) - m(t)}{\alpha_1}, \\ \dot{v}(t) = \dfrac{\left[\partial_{\theta_k} f(\theta(t))\right]^2 - m(t)}{\alpha_2}. \end{cases}$$

The dynamics is written here in the absence of minibatching noise, but similar expressions can be obtained when minibatching is performed (similarly to (4.14) *in the gradient case).*

The default values for γ, ε, β_1 and β_2 in implementations of Adam are $\overline{\gamma} = 10^{-3}$, $\varepsilon = 10^{-6}$, $\overline{\beta}_1 = 0.9$ and $\overline{\beta}_2 = 0.999$. In fact, similarly to the discussion after (4.12), the latter two coefficients should be thought of as $\beta_i = \mathrm{e}^{-\gamma/\alpha_i}$, and should therefore be modified when changing the learning rate to $\gamma \neq \overline{\gamma}$ as $\beta_i = \overline{\beta}_i^{\gamma/\overline{\gamma}}$ (see the discussion in [14, Section 5.2]).

Exercise 4.10. *Give the expression of v_t in terms of v_0 and $\widehat{g}_s(\theta_{s-1})$ for $1 \leqslant s \leqslant t$, and motivate the renormalization procedure* (4.17).

5

Principal component analysis

We study in this chapter a first instance of unsupervised learning (*i.e.* "learning without a teacher"). The aim is to infer properties of the distribution $p_{\text{data}}(dx)$ from the (unlabelled) data points $x_1, \dots, x_n$. For low dimensional situations, one could approximate $p_{\text{data}}(dx)$ using kernel density estimators. Such approaches are however not an option for high dimensional situations typical of machine learning applications. In order to make progress, one therefore has to settle for rather crude global models (*e.g.* represent p_{data} as a mixture of Gaussian distributions), or to give up on identifying the distribution itself but rather find its important modes by some cluster analysis (see Chapter 9). Another option, which we consider here, is to identify a low dimensional manifold around which data points are concentrated. This is done mostly in a linear setting in this chapter, although we hint at some nonlinear extensions.

More precisely, after motivating why it is interesting to reduce the dimension of the data points in Section 5.1, we derive Principal Component Analysis (PCA) through a perspective based on minimizing a reconstruction error in Section 5.2. We then make precise the usual formulation of PCA in Section 5.3, and elaborate on its interpretation in Section 5.4. We finally discuss in Section 5.5 various extensions and generalizations of this method, including kernel PCA, probabilistic PCA and factor analysis, and nonlinear versions of PCA. The presentation here is based on [41, Sections 20.1 and 20.2], [6, Chapters 15 and 21] and [8, Chapter 12].

5.1 Motivating the need for dimensionality reduction

One may want to reduce the dimensionality of the data points $x \in \mathcal{X}$ for various reasons:

- *computational*: the data can be compressed in a preprocessing step to perform faster subsequent operations and/or apply other machine learning algorithms for regression or classification;
- *featurization* is related to the previous item: in certain applications, such as finding so-called reaction coordinates or collective variables in molecular dynamics, it is of interest to find (non)linear functions of the data points which provide relevant inputs for other methods.
- *data exploration/visualization*: high dimensional data cannot be easily represented in a way amenable for humans to understand its organization. It is on the other hand much easier to

plot 2 or 3 dimensional data points, and develop some intuition on how the data points are organized. This makes sense only if the chosen dimensions are particularly representative.

Beyond these motivations, one actually expects that the data points lie around some low dimensional manifold, and that the probability measure from which these data points are distributed is concentrated around this manifold. Indeed, if this was not the case, learning algorithms would not be able to make successful predictions – think for instance of image recognition, even binarized images of low resolution, as the binarized version of the MNIST data set in which images are composed of 28×28 pixels. There are $2^{28\times 28} \approx 10^{236}$ possible images, which is an enormous number in comparison to the size of the training data set (about $60,000$ instances). We however note that many pixels are irrelevant (pixels around the boundary are mostly white), and also expect that only some pixels have a crucial role in digit recognition (a very low resolution image can be enough to distinguish between digits).

Given these elements of context, we think in this chapter of data points $x \in \mathcal{X} = \mathbb{R}^{1\times d}$ as obtained through some latent model (recall that x is considered as a line vector). More precisely, we start by recentering the data by the empirical average of the sample at hand. The idea is next to recover x from a lower dimensional variable $z = xU^\top + \varepsilon \in \mathbb{R}^{1\times \ell}$ with $U \in \mathbb{R}^{\ell\times d}$ and ε some random variable with values in $\mathbb{R}^{1\times\ell}$ (if no centering was considered, one would have to add a bias to the above linear model; see Lemma 5.1). The notation ℓ is chosen to emphasize that this number is the dimension of the latent space. We consider $\ell \leqslant d-1$, and in fact we want ℓ to be much smaller than d. The latent model corresponds to considering an encoding function $\mathbb{R}^{1\times d} \to \mathbb{R}^{1\times\ell}$ acting as $x \mapsto xU^\top$, and a decoding function $\mathbb{R}^{1\times\ell} \to \mathbb{R}^{1\times d}$ acting as $z \mapsto zU$ (it is not clear at this stage why the decoder should use U and not a generic matrix $\mathbb{R}^{\ell\times d}$; this is justified below in Lemma 5.1). If no noise is present, one recovers PCA. Noise can however be injected into the model in order to have a generative viewpoint, and construct a distribution of data points from a distribution of latent variables z, see Section 5.5.3.

5.2 Deriving PCA from reconstruction error

We justify in this section the latent model described at the end of Section 5.1, and show how PCA can be understood from the minimization of some reconstruction error. We consider to this end a general linear dimensionality reduction scheme, where a data point $x \in \mathbb{R}^{1\times d}$ is first embedded in $\mathbb{R}^{1\times\ell}$ as $z = xE$ (encoding), then lifted back to $\mathbb{R}^{1\times d}$ as zD (decoding), and finally shifted by $b \in \mathbb{R}^{1\times d}$. The matrices $E \in \mathbb{R}^{d\times\ell}$ and $D \in \mathbb{R}^{\ell\times d}$, and the bias $b \in \mathbb{R}^{1\times d}$ are found by minimizing the reconstruction loss, which corresponds to considering the following training loss:

$$\widehat{\mathcal{R}}_n(E,D,b) = \frac{1}{n}\sum_{i=1}^n \|x_i - (x_iED + b)\|_2^2 . \tag{5.1}$$

In fact, the following lemma shows that the bias b is known and related to the empirical average of the data

$$\overline{x}_n = \frac{1}{n}\sum_{i=1}^n x_i .$$

Moreover, the matrix E for optimal solutions is related to the matrix D of optimal solutions; and furthermore D has specific orthogonality properties (see [50, Lemma 23.1]).

Lemma 5.1. *The optimal reconstruction loss can be obtained as*

$$\inf_{\substack{E\in\mathbb{R}^{d\times\ell}\\ D\in\mathbb{R}^{\ell\times d}\\ b\in\mathbb{R}^{1\times d}}} \widehat{\mathcal{R}}_n(E,D,b) = \min_{\substack{V\in\mathbb{R}^{\ell\times d}\\ VV^\top = \mathrm{Id}_\ell}} \left\{ \frac{1}{n}\sum_{i=1}^n \left\| x_i - \overline{x}_n - (x_i - \overline{x}_n)V^\top V \right\|_2^2 \right\}.$$

Exercise 5.1. *The aim of this exercise is to prove Lemma 5.1.*

(a) Fix $(E, D) \in \mathbb{R}^{d\times\ell} \times \mathbb{R}^{\ell\times d}$ *and minimize over* $b \in \mathbb{R}^d$ *to prove that*

$$\inf_{\substack{E\in\mathbb{R}^{d\times\ell}\\ D\in\mathbb{R}^{\ell\times d}\\ b\in\mathbb{R}^{1\times d}}} \widehat{\mathcal{R}}_n\left(E, D, b\right) = \inf_{\substack{E\in\mathbb{R}^{d\times\ell}\\ D\in\mathbb{R}^{\ell\times d}}} \frac{1}{n}\sum_{i=1}^{n} \|\widetilde{x}_i - \widetilde{x}_i ED\|_2^2 \,,$$

with $\widetilde{x}_1, \ldots, \widetilde{x}_n$ *to be made precise.*

(b) Denote by $\mathcal{V}$ *the range of the mapping* $\mathbb{R}^d \ni x \mapsto xED$*, and by* $\Pi_{\mathcal{V}}$ *the orthogonal projection onto* $\mathcal{V}$*. Show that*

$$\frac{1}{n}\sum_{i=1}^{n} \|\widetilde{x}_i - \widetilde{x}_i ED\|_2^2 \geqslant \frac{1}{n}\sum_{i=1}^{n} \|\widetilde{x}_i - \Pi_{\mathcal{V}}\widetilde{x}_i\|_2^2 \,.$$

(c) Deduce that

$$\inf_{\substack{E\in\mathbb{R}^{d\times\ell}\\ D\in\mathbb{R}^{\ell\times d}}} \frac{1}{n}\sum_{i=1}^{n} \|\widetilde{x}_i - \widetilde{x}_i ED\|_2^2 \geqslant \inf_{\substack{\mathcal{V}_\ell\subset\mathbb{R}^{1\times d}\\ \dim(\mathcal{V}_\ell)=\ell}} \frac{1}{n}\sum_{i=1}^{n} \|\widetilde{x}_i - \Pi_{\mathcal{V}_\ell}\widetilde{x}_i\|_2^2 \,.$$

(d) Show that the previous inequality can be reformulated as

$$\inf_{\substack{E\in\mathbb{R}^{d\times\ell}\\ D\in\mathbb{R}^{\ell\times d}}} \frac{1}{n}\sum_{i=1}^{n} \|\widetilde{x}_i - \widetilde{x}_i ED\|_2^2 \geqslant \inf_{\substack{V\in\mathbb{R}^{\ell\times d}\\ VV^\top=\mathrm{Id}_\ell}} \frac{1}{n}\sum_{i=1}^{n} \left\|\widetilde{x}_i - \widetilde{x}_i V^\top V\right\|_2^2 \,.$$

(e) Conclude.

In the remainder of this section, in order to alleviate the notation, we assume that the data has been recentered in a pre-processing step, so that $\overline{x}_n = 0$. Lemma 5.1 then shows that the minimization of the reconstruction loss $\widehat{\mathcal{R}}_n$ defined in (5.1) can be equivalently rewritten as

$$\min\left\{\frac{1}{n}\sum_{i=1}^{n} \left\|x_i - x_i V^\top V\right\|_2^2,\ V \in \mathbb{R}^{\ell\times d},\, VV^\top = \mathrm{Id}_\ell\right\}. \tag{5.2}$$

Upon writing

$$V = \begin{pmatrix} \text{—} & v_1 & \text{—} \\ & \vdots & \\ \text{—} & v_\ell & \text{—} \end{pmatrix},$$

with $v_i v_j^\top = \delta_{ij}$ from the condition $VV^\top = \mathrm{Id}_\ell$, we note that

$$x_i V^\top V = \sum_{k=1}^{\ell} \left(x_i v_k^\top\right) v_k$$

is the orthogonal projection of x_i onto $\mathrm{Span}(v_1, v_2, \ldots, v_\ell)$. The properties of orthogonal projections imply that

$$\left\|x_i - x_i V^\top V\right\|_2^2 = \|x_i\|_2^2 - \left\|x_i V^\top V\right\|_2^2 \,,$$

which can also be seen from a direct computation. This shows that (5.2) is equivalent to

$$\max\left\{\frac{1}{n}\sum_{i=1}^{n} \left\|x_i V^\top V\right\|_2^2,\ V \in \mathbb{R}^{\ell\times d},\, VV^\top = \mathrm{Id}_\ell\right\}. \tag{5.3}$$

The interpretation of the latter maximization problem is that one wants to maximize the variance of the data points projected onto a vector subspace of dimension ℓ (this interpretation in terms of variance being valid when the data has been recentered, see Section 5.3).

The formulations (5.3) or (5.2) allow us to solve the problem in an explicit manner. For pedagogical reasons, we start by presenting the solution for one dimensional latent spaces, and then generalize the procedure to latent spaces of arbitrary dimension $\ell \leqslant d - 1$.

One dimensional latent spaces. When the latent space is one dimensional, *i.e.* $\ell = 1$, the problem (5.3) can be reformulated as

$$\max \left\{ \frac{1}{n} \sum_{i=1}^{n} \|(x_i \cdot v)v\|^2 \,\middle|\, v \in \mathbb{R}^{1\times d},\ \|v\|^2 = 1 \right\}.$$

The functional to maximize can be rewritten as

$$\frac{1}{n} \sum_{i=1}^{n} \|(x_i \cdot v)v\|^2 = \frac{1}{n} \sum_{i=1}^{n} (x_i \cdot v)^2 = v \widehat{\Sigma} v^\top, \qquad \widehat{\Sigma} = \frac{1}{n} \sum_{i=1}^{n} x_i^\top x_i \in \mathbb{R}^{d\times d}. \tag{5.4}$$

The Euler–Lagrange equation associated with the maximization problem therefore reads

$$u\widehat{\Sigma} = \lambda u,$$

where $\lambda \in \mathbb{R}$ is the Lagrange multiplier associated with the normalization constraint. This shows that u is an eigenvector associated with the symmetric matrix $\widehat{\Sigma}$ (which is positive semi-definite). In order to maximize $u\widehat{\Sigma}u^\top = \lambda \|u\|^2 = \lambda$, one therefore chooses a normalized eigenvector associated with the largest eigenvalue of $\widehat{\Sigma}$.

Generalization to a latent space of arbitrary dimension. The derivation in the case $\ell = 1$ suggests to consider the eigenvalues $\lambda_1 \geqslant \lambda_2 \geqslant \ldots \geqslant \lambda_d$ and associated normalized eigenvectors $(u_j)_{1\leqslant j\leqslant d}$ of the symmetric positive matrix $\widehat{\Sigma}$:

$$\lambda_j u_j = u_j \widehat{\Sigma}, \qquad \|u_j\|_2^2 = 1.$$

Theorem 5.1. *The reconstruction error*

$$\min \left\{ \frac{1}{n} \sum_{i=1}^{n} \left\| x_i - x_i V^\top V \right\|_2^2,\ V \in \mathbb{R}^{\ell\times d},\ VV^\top = \mathrm{Id}_\ell \right\}$$

is minimized for

$$U = \begin{pmatrix} \text{—} & u_1 & \text{—} \\ & \vdots & \\ \text{—} & u_\ell & \text{—} \end{pmatrix},$$

with $(u_1, \ldots, u_\ell)$ an orthonormal family of eigenvectors associated with the ℓ largest eigenvalues of $\widehat{\Sigma}$. Moreover, the minimal reconstruction error is

$$\frac{1}{n} \sum_{i=1}^{n} \left\| x_i - x_i U^\top U \right\|_2^2 = \sum_{k=\ell+1}^{d} \lambda_k. \tag{5.5}$$

Remark 5.1. *There is of course no uniqueness result on the minimizer matrix $U \in \mathbb{R}^{\ell\times d}$, since, for example, $U^\top U$ remains the same by changing U to RU for any orthogonal matrix $R \in \mathbb{R}^{\ell\times\ell}$.*

Exercise 5.2. *The aim of this exercise is to prove Theorem 5.1.*

(a) Prove the equality (5.5).
(b) Consider an arbitrary matrix $V \in \mathbb{R}^{\ell\times d}$ such that $VV^\top = \mathrm{Id}_\ell$. Denoting by $v_1, \ldots, v_\ell$ the lines of V, show that

$$\frac{1}{n} \sum_{i=1}^{n} \left\| x_i - x_i V^\top V \right\|_2^2 = \frac{1}{n} \sum_{i=1}^{n} \|x_i\|_2^2 - \frac{1}{n} \sum_{i=1}^{n} \sum_{k=1}^{\ell} \left(x_i v_k^\top \right)^2.$$

(c) Expand v_k on the orthonormal basis $(u_1, \dots, u_d)$ of eigenvectors of $\widehat{\Sigma}$ associated with the eigenvalues $0 \leqslant \lambda_1 \geqslant \dots \geqslant \lambda_d$ sorted in decreasing order to establish that

$$\frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{\ell}\left(x_i v_k^\top\right)^2 = \sum_{k'=1}^{d}\lambda_{k'}\left\|u_{k'}V^\top V\right\|_2^2.$$

(d) Prove that the quantity on the right hand side of the previous inequality is smaller than $\lambda_1 + \dots + \lambda_\ell$.

(e) Conclude.

5.3 PCA in practice

We summarize in this section how PCA is performed in practice, given a data set $\{x_1, \dots, x_n\} \subset \mathbb{R}^d$ and a fixed dimension ℓ for the latent space.

(1) A first remark is that the data needs to be centered. When the features are heterogeneous in scale (for instance because they correspond to measurements of quantities in different units), it is also a good practice to standardize it; see for instance the discussion in [41, Section 20.1.3.1]. This amounts to replacing the covariance matrix by the correlation matrix, namely replacing the attributes x_i^j by

$$\widetilde{x}_i^j = \frac{x_i^j - \overline{x}_n^j}{\sigma_{n,j}}, \qquad \overline{x}_n = \frac{1}{n}\sum_{i=1}^{n} x_i, \qquad \sigma_{n,j}^2 = \frac{1}{n}\sum_{i=1}^{n}\left(x_i^j - \overline{x}_n^j\right)^2.$$

The dataset $\{\widetilde{x}_1, \dots, \widetilde{x}_n\}$ has by construction a vanishing empirical mean and unit empirical variances for all attributes.

(2) The second step is to construct the empirical covariance matrix associated with the renormalized data, namely

$$\widehat{\Sigma} = \frac{1}{n}\sum_{i=1}^{n}\widetilde{x}_i^\top \widetilde{x}_i \in \mathbb{R}^{d\times d},$$

and to diagonalize this symmetric positive matrix:

$$u_k\widehat{\Sigma} = \lambda_k u_k, \qquad \|u_k\|^2 = 1, \qquad \lambda_1 \geqslant \lambda_2 \geqslant \dots \geqslant \lambda_d \geqslant 0. \tag{5.6}$$

(3) We next compute the embeddings of the data points in the latent space: $z_i = x_i U_\ell^\top \in \mathbb{R}^{1\times\ell}$, where $U_\ell \in \mathbb{R}^{\ell\times d}$ is the matrix whoses lines are the first ℓ eigenvectors $u_1, \dots, u_\ell$.

(4) Approximate reconstructions of the data points are given by

$$\widehat{x}_i^j = \overline{x}_n^j + \sigma_{n,j}\left(z_i U_\ell\right)_j \in \mathbb{R}.$$

Predictions $\widehat{x'}$ for test points $x' \in \mathcal{X}$ are performed by first normalizing the input as

$$\widetilde{x}'^j = \frac{x'^j - \overline{x}_n^j}{\sigma_{n,j}},$$

and then considering

$$\widehat{x'}^j = \overline{x}_n^j + \sigma_{n,j}\left[\widetilde{x}' U_\ell^\top U_\ell\right]_j \in \mathbb{R}.$$

5.4 Interpretation of PCA

For notational simplicity, we denote by $\{x_1, \dots, x_n\} \subset \mathbb{R}^d$ the recentered data set instead, which can be renormalized or not. The principal components are the vectors

$$c_k = \begin{pmatrix} u_k x_1^\top \\ \vdots \\ u_k x_n^\top \end{pmatrix}$$

associated with the eigenvectors of $\widehat{\Sigma}$ and the normalized data set. The various components of c_k give the score of each data point on the kth loading u_k. These loadings are linear combination of features, and can therefore be thought of as "implicit features".

Some properties of the score vector are given in the next proposition. In order to state it, we introduce the empirical correlation for two sequences $a = (a_i)_{1 \leqslant i \leqslant n}$ and $b = (b_i)_{1 \leqslant i \leqslant n}$ with vanishing empirical averages:

$$\mathrm{Corr}(a, b) = \frac{\displaystyle\sum_{i=1}^n a_i b_i}{\displaystyle\sqrt{\sum_{i=1}^n a_i^2} \sqrt{\sum_{i=1}^n b_i^2}}.$$

Proposition 5.1. *The principal components have a vanishing empirical correlation:*

$$\forall 1 \leqslant k \neq k' \leqslant d, \qquad \mathrm{Corr}\left(c_k, c_{k'}\right) = 0. \tag{5.7}$$

*Moreover, denoting by X^j the jth column of the data matrix X (*i.e. *X^j is the vector of the j-th features),*

$$\forall j \in \{1, \dots, d\}, \qquad \sum_{k=1}^d \mathrm{Corr}\left(c_k, X^j\right)^2 = 1. \tag{5.8}$$

The proof of this result is the content of Exercise 5.3. The second property corresponds to the so-called *correlation sphere*. In particular, the vector $(\mathrm{Corr}\left(c_k, X^j\right))_{1 \leqslant k \leqslant \ell}$ of features which are retained by PCA is necessarily inside the unit ball. For $\ell = 2$, the vector $(\mathrm{Corr}\left(c_1, X^j\right), \mathrm{Corr}\left(c_2, X^j\right))$ lies in a disk in the so-called first factorial plane. Depending on the position of this vector in the unit ball, various situations can be distinguished:

- if $\mathrm{Corr}\left(c_1, X^j\right)$ is close to 1 in absolute value and $\mathrm{Corr}\left(c_2, X^j\right)$ is small in absolute value, then the j-th feature correlates well with the score on the first loading;
- if $\mathrm{Corr}\left(c_2, X^j\right)$ is close to 1 in absolute value and $\mathrm{Corr}\left(c_1, X^j\right)$ is small in absolute value, then the j-th feature correlates well with the score on the second loading;
- if both $\mathrm{Corr}\left(c_1, X^j\right)$ and $\mathrm{Corr}\left(c_2, X^j\right)$ are small in absolute value, then the j-th feature is not correlated with the scores on the first two loadings.

In essence, this representation allows to check whether a feature correlates more to c_1 or c_2, or maybe none of them. Once the features are interpreted this way, one can also look at the projection of data points themselves, *i.e.* consider the scatter plot of the vectors $(c_{1,i}, c_{2,i})$ for $1 \leqslant i \leqslant n$. This allows to group data points depending on whether they are better explained by c_1 (in which case they would be close to $(\pm 1, 0)$), by c_2 (in which case they would be close to $(0, \pm 1)$), or by none of these quantities. This allows to interpret the data in terms of the relevant features identified by the correlation sphere. It is good to focus on an example to gain some intuition here; for instance the decathlon data, or MNIST digits (see the discussions around [25, Figure 14.23] and [41, Figure 20.2] for instance).

Exercise 5.3. *The aim of this exercise is to prove Proposition 5.1.*

(a) Prove (5.7) *by computing* $c_k \cdot c_{k'}$.
(b) Show that $\mathrm{Corr}\left(c_k, X^j\right) = \dfrac{\sqrt{\lambda_k} u_k^j}{\sigma_{n,j}}$.
(c) Prove (5.8).

Choosing the number ℓ of latent dimensions. The reconstruction error is always better when more latent dimensions are considered, even for the test data. There is no issue of overfitting. The common practice is to identify a sufficiently small value of ℓ by looking at the scree plot, where eigenvalues λ_k are plotted as a function of k. One identifies a knee or elbow in this curve, where the decrease of the eigenvalues slows down so that the marginal increase in the sum $\lambda_1 + \cdots + \lambda_k$ becomes smaller with k. Alternatively, this corresponds to looking at the fraction of explained variance

$$f_k = \frac{\sum_{k'=1}^{k} \lambda_{k'}}{\sum_{k'=1}^{d} \lambda_{k'}}$$

as a function of k, and identifying the index ℓ after which the gains become smaller.

5.5 Extensions of PCA

We present in this section various extensions of PCA. We start by discussing in Section 5.5.1 how to adapt the method to high dimensional data (*i.e.* situations in which the dimension d of the features is much larger than the number n of data points). We next show how to include some nonlinearity by using kernel functions, as made precise in Section 5.5.2. This method can be thought of nonlinearly pre-processing the data and then performing some linear dimensionality reduction. A genuinely nonlinear extension of PCA, based on autoencoder neural networks, is discussed later on in Section 8.3. Finally, we present in Section 5.5.3 a generative method based on PCA, known as probabilitic PCA.

Exercise 5.4 (Weighted PCA). *We consider in this exercise a data set* $\{x_i\}_{1 \leqslant i \leqslant n} \subset \mathbb{R}^{1 \times d}$ *(with elements considered as line vectors), where the data points come with a weight* $\omega_i > 0$. *The aim in this case is to minimize the reconstruction error*

$$\min_{\substack{V \in \mathbb{R}^{\ell \times d} \\ VV^\top = \mathrm{Id}_\ell}} \left\{ \sum_{i=1}^{n} \omega_i^2 \left\| x_i - x_i V^\top V \right\|_2^2 \right\}. \tag{5.9}$$

The data is assumed to be recentered in a way such that

$$\sum_{i=1}^{n} \omega_i x_i = 0 \in \mathbb{R}^{1 \times d}.$$

(a) For which values of $\{\omega_i\}_{1 \leqslant i \leqslant n}$ *is standard PCA recovered?*
(b) We consider $\ell = 1$ *in this question. How should* V *be chosen in order to solve* (5.9)*?*
(c) We now turn to the general case $\ell \geqslant 2$. *Solve* (5.9) *by introducing* $\widetilde{x}_i = \omega_i x_i$.

5.5.1 High dimensional data

PCA requires diagonalizing the matrix $\widehat{\Sigma} \in \mathbb{R}^{d \times d}$ defined in (5.4), which has a computational cost $\mathrm{O}(d^3)$. When there are many features d, even more than the number of data points n, the diagonalization of the matrix can become too expensive, and it would be better to reformulate PCA with a diagonalization of a $n \times n$ matrix, similarly to what is done for linear least square

regression in Exercise 2.11. The key point here is to note that the eigenvalue problem (5.6) can be rewritten as follows, upon introducing the design matrix $X \in \mathbb{R}^{n\times d}$ whose ith row is x_i and multiplying (5.6) by $X^\top$ on the right:

$$uX^\top \Gamma = \lambda u X^\top, \qquad \Gamma = \frac{1}{n} X X^\top \in \mathbb{R}^{n\times n}.$$

More explicitly, $\Gamma_{ij} = x_i x_j^\top$. This suggests to solve eigenvalue problems for Γ, namely find $v \in \mathbb{R}^{1\times n} \setminus \{0\}$ and $\lambda > 0$ such that $v\Gamma = \lambda v$. Then,

$$vX\widehat{\Sigma} = v\Gamma X = \lambda v X,$$

so that vX is an eigenvector of $\widehat{\Sigma}$. Note that $vX \neq 0$ otherwise $v\Gamma = 0 = \lambda v$, so $v = 0$, which is not possible. This shows that normalized eigenvectors u of $\widehat{\Sigma} \in \mathbb{R}^{d\times d}$ are obtained from eigenvectors v of $\Gamma \in \mathbb{R}^{n\times n}$ by multiplying them by X on the right, and renormalizing the so-obtained vector as $vX/\|vX\|_2 \in \mathbb{R}^{1\times d}$.

5.5.2 Kernel PCA

Kernel methods allow to go beyond linear methods by introducing some nonlinearity through an implicit featurization procedure. We here only briefly discuss how this works, as a more complete presentation is provided in Section 6.4.2.

Before presenting the implicit featurization, let us start by considering an explicit featurization based on some feature function $\phi : \mathcal{X} \to \mathbb{R}^{1\times D}$. In this context, the design matrix $\widehat{\Sigma}$ in (5.4) is replaced by

$$\widehat{\Sigma} = \frac{1}{n}\sum_{i=1}^n \phi(x_i)^\top \phi(x_i) \in \mathbb{R}^{D\times D}.$$

In fact, some recentering has to be performed in order for this matrix to be interpreted as some covariance matrix, see the discussion in [41, Section 20.4.6].

The value of D can be large, in fact infinite, but, as in Section 5.5.1, this will not pose an issue as one can formulate PCA based on the diagonalization of a $n \times n$ matrix. The eigenvalue equation to be solved in this context is

$$\lambda v = v\widehat{\Sigma} = \frac{1}{n}\sum_{i=1}^n \left(v\phi(x_i)^\top\right)\phi(x_i) \in \mathbb{R}^{1\times D}.$$

This shows that v is a linear combination of $\phi(x_1), \dots, \phi(x_n)$, and can therefore be written as

$$v = \sum_{i=1}^n a_i \phi(x_i).$$

By plugging this equality in the eigenproblem to solve, one finds

$$\frac{1}{n}\sum_{i,j=1}^n a_j \left(\phi(x_j)\phi(x_i)^\top\right)\phi(x_i) = \lambda \sum_{j=1}^n a_j \phi(x_j).$$

We multiply this equality by $\phi(x_k)^\top$ on the right, and introduce the symmetric matrix $K \in \mathbb{R}^{n\times n}$ with entries $K_{ij} = \mathcal{K}(x_i, x_j)$, where

$$\mathcal{K}(x, x') = \phi(x)\phi(x')^\top \in \mathbb{R}.$$

This leads to

$$\frac{1}{n}\sum_{i,j=1}^n a_j K_{ji} K_{ik} = \lambda \sum_{j=1}^n a_j K_{jk}.$$

Since $1 \leqslant k \leqslant n$ is arbitrary, the eigenvalue problem can finally be reformulated as

$$aK^2 = n\lambda aK, \qquad a \in \mathbb{R}^{1\times n}, \qquad K = K^\top \in \mathbb{R}^{n\times n},$$

which can be solved by diagonalizing the matrix K. Once the matrix is diagonalized and the ℓ leading eigenvectors $a_1, \ldots, a_\ell$ are found, reconstruction can be performed with the orthonormal basis of vectors

$$u_k = \frac{\displaystyle\sum_{i=1}^n a_{k,i}\phi(x_i)}{\left\|\displaystyle\sum_{i=1}^n a_{k,i}\phi(x_i)\right\|_2}.$$

More precisely, the reconstruction is based on the scores

$$u_k\phi\left(x'\right)^\top = \frac{\displaystyle\sum_{i=1}^n a_{k,i}\phi(x_i)\phi\left(x'\right)^\top}{\left\|\displaystyle\sum_{i=1}^n a_{k,i}\phi(x_i)\right\|_2}.$$

Since

$$\left\|\sum_{i=1}^n a_{k,i}\phi(x_i)\right\|_2^2 = \sum_{i,j=1}^n a_{k,i}a_{k,j}\phi(x_i)\phi(x_j)^\top = a_k K a_k^\top,$$

we finally obtain

$$u_k\phi\left(x'\right)^\top = \frac{\displaystyle\sum_{i=1}^n a_{k,i}\mathcal{K}\left(x', x_i\right)}{\sqrt{a_k K a_k^\top}}.$$

Note that, crucially, the explicit featurization ϕ is not needed, as the only quantity which appears is the kernel function $\mathcal{K}(x, x')$. It therefore suffices to make precise this kernel function. As mentioned above, we elaborate on this in Section 6.4.2.

In the context of kernel PCA, the test data point x' is replaced by the knowledge of the ℓ scores $u_k\phi\left(x'\right)^\top$ for $1 \leqslant k \leqslant \ell$. The reconstruction of the test point x' itself is not straightforward as it is based on some form of regression problem, see for instance the discussion in [6, Section 15.7]. Applications of kernel PCA are given in the work [48] which introduced the method. One major aim is to featurize the data in a nonlinear way as postprocessing step, for example for denoising images.

5.5.3 Probabilistic PCA

PCA allows to embed training points x_i into a lower dimensional space, but does not allow to generate new points. Probabilistic PCA (PPCA) is a generative method which provides a simple model for $p_{\text{data}}(x)$, known as "predictive distribution". More precisely, considering in this section realizations under the predictive distribution as column vectors, the model is obtained as

$$X = WZ + \varepsilon + b \in \mathbb{R}^d, \qquad Z \sim \mathcal{N}(0, \mathrm{Id}_\ell), \qquad \varepsilon \sim \mathcal{N}\left(0, \sigma^2\mathrm{Id}_d\right),$$

for $\sigma > 0$ and some matrix $W \in \mathbb{R}^{d\times\ell}$, and with ε independent of Z. Note that X is a Gaussian random vector, with mean b and covariance matrix $WW^\top + \sigma^2\mathrm{Id}_d$ since

$$\mathbb{E}[X] = \mathbb{E}\left[WZ + \varepsilon + b\right] = W\mathbb{E}[Z] + b = b,$$

and

$$\mathrm{Var}(X) = \mathbb{E}\left[(WZ+\varepsilon)(WZ+\varepsilon)^\top\right] = \mathbb{E}\left[WZZ^\top W^\top\right] + \mathbb{E}\left[\varepsilon\varepsilon^\top\right] = W\mathbb{E}\left[ZZ^\top\right]W^\top + \sigma^2\mathrm{Id}_d.$$

Probabilistic PCA is therefore fully determined by the parameters W, b and σ^2. These parameters can be found by a maximum likelihood approach (either by reducing the question to an eigenvalue problem, or by using an expectation/maximization approach for high dimensional problems; see [8, Section 12.2]).

Factor analysis is a generative extension of PCA, which goes beyond probabilistic PCA by working with a diagonal covariance matrix $\mathcal{V}$ instead of an isotropic covariance $\sigma^2\mathrm{Id}_d$. The generative model under consideration is obtained by decoding a latent variable distributed according to an isotropic Gaussian random variable, then adding a bias to recenter the data, as well as some Gaussian noise whose intensity depends on the coordinate at hand; see Exercise 5.5 below.

Exercise 5.5 (Factor analysis). *We consider the generative model*

$$X = WZ + \varepsilon + b, \qquad Z \sim \mathcal{N}(0, \mathrm{Id}_\ell), \qquad \varepsilon \sim \mathcal{N}(0, \mathcal{V}), \tag{5.10}$$

with Z, ε independent random variables, $X, b \in \mathbb{R}^d$, $W \in \mathbb{R}^{d\times\ell}$, $\mathcal{V} \in \mathbb{R}^{d\times d}$, $Z \in \mathbb{R}^\ell$, with $\ell \leqslant d-1$ the dimension of the latent space. The covariance matrix for the noise variable ε is assumed to be diagonal:

$$\mathcal{V} = \begin{pmatrix} v_1 & 0 & \dots & \\ 0 & v_2 & 0 & \dots \\ & & \ddots & \\ 0 & \dots & 0 & v_d \end{pmatrix},$$

with entries $v_i \geqslant 0$. The idea is to determine $W, b, \mathcal{V}$ from the dataset $\{x_i\}_{1\leqslant i\leqslant n} \subset \mathbb{R}^d$ by some maximum likelihood procedure (as for PCA, we work here in an unsupervised setting, so the data set does not come with labels).

(a) Prove that X is a Gaussian random variable with mean b and covariance $WW^\top + \mathcal{V}$. How many free parameters enter the definition of the covariance matrix? How does this number compare to the number of parameters used to model the covariance of X without any structural assumption?

*(b) Show that the generative model is unchanged if W is replaced by WR, with $R \in \mathbb{R}^{\ell\times\ell}$ an orthogonal matrix (*i.e. *a matrix R such that $R^\top R = RR^\top = \mathrm{Id}_\ell$).*

The previous question suggests that it is not relevant to look for W alone. In view of the previous question, a better quantity to consider is the part $WW^\top$ of the covariance $\Sigma_W = WW^\top + \mathcal{V}$. We assume in the remainder of this exercise that $v_k > 0$ for any $1 \leqslant k \leqslant d$.

(c) Prove that Σ_W is invertible.

(d) Write the negative log-likelihood $L(W, \mathcal{V}, b)$ corresponding to the dataset $\{x_i\}_{1\leqslant i\leqslant n}$ for the model (5.10) *(assuming that the data points are independent).*

(e) Find the expression of b by minimizing $L(W, \mathcal{V}, b)$ for $W, \mathcal{V}$ fixed.

(f) Show that finding $W, \mathcal{V}$ amounts to minimizing

$$\mathscr{L}(W, \mathcal{V}) = \mathrm{Tr}\left(\Sigma_W^{-1}\widehat{C}\right) + \ln\left(\det\Sigma_W\right),$$

where $\widehat{C} \in \mathbb{R}^{d\times d}$ is the empirical covariance of the data set $\{x_i\}_{1\leqslant i\leqslant n}$.

The algorithm to find $WW^\top$ and $\mathcal{V}$ proceeds in an iterative manner, by alternately minimizing first over $\mathcal{V}$ for W fixed, then over W for $\mathcal{V}$ fixed, until convergence is reached. We make precise how to perform these two steps in the next questions.

The following three questions are more difficult and can be skipped. The last questions should however be treated.

(g) *We start by minimizing* $\mathscr{L}(W, \mathcal{V})$ *for* W *fixed, assuming that this minimization problem is well posed. Using the following identities for symmetric matrices* X, M *(see for instance [41, Section 7.8.7.3])*

$$\nabla_X \left[\mathrm{Tr}\left(X^{-1}M\right)\right] = -X^{-1}MX^{-1}, \qquad \nabla_X \left[\ln(\det X)\right] = X^{-1}, \tag{5.11}$$

show that the minimizer $\mathcal{V}$ *satisfies*

$$\mathrm{diag}\left(\Sigma_W^{-1}\widehat{C}\Sigma_W^{-1}\right) = \mathrm{diag}\left(\Sigma_W^{-1}\right), \tag{5.12}$$

where $\mathrm{diag}(M)$ *for* $M \in \mathbb{R}^{d\times d}$ *is the diagonal matrix with entries* M_{kk} *on the diagonal.*

(h) *We fix* $\mathcal{V}$*, and consider the change of unknown matrix* $\mathscr{W} = U\mathcal{V}^{-1/2}W \in \mathbb{R}^{d\times \ell}$*, where* U *is an orthogonal matrix obtained from the diagonalization of the symmetric positive matrix* $\mathcal{V}^{-1/2}\widehat{C}\mathcal{V}^{-1/2}$*,* i.e.

$$\mathcal{V}^{-1/2}\widehat{C}\mathcal{V}^{-1/2} = U^\top \Lambda U,$$

where $U \in \mathbb{R}^{d\times d}$ *is an orthogonal matrix, and* $\Lambda \in \mathbb{R}^{d\times d}$ *is a diagonal matrix with entries* $\lambda_i \geqslant 0$*, ranked in order of decreasing magnitudes. Show that the minimization of* $\mathscr{L}(W, \mathcal{V})$ *over* $W \in \mathbb{R}^{d\times \ell}$ *is equivalent to the minimization over* $\mathscr{W} \in \mathbb{R}^{d\times \ell}$ *of*

$$\mathfrak{L}_{\mathcal{V}}(\mathscr{W}) = \mathrm{Tr}\left[\left(\mathscr{W}\mathscr{W}^\top + \mathrm{Id}_d\right)^{-1}\Lambda\right] + \ln\left[\det\left(\mathscr{W}\mathscr{W}^\top + \mathrm{Id}_d\right)\right].$$

(i) *Using* (5.11)*, it can be shown that minimizers of* $\mathfrak{L}_{\mathcal{V}}(\mathscr{W})$ *satisfy*

$$\left(\mathscr{W}\mathscr{W}^\top + \mathrm{Id}_d\right)^{-1}\Lambda\left(\mathscr{W}\mathscr{W}^\top + \mathrm{Id}_d\right)^{-1}\mathscr{W} = \left(\mathscr{W}\mathscr{W}^\top + \mathrm{Id}_d\right)^{-1}\mathscr{W}. \tag{5.13}$$

This equality can be reformulated in terms of the variable W *as* $W = \widehat{C}\Sigma_W^{-1}W$*. Use the latter identity and* (5.12) *to establish that* $\mathrm{diag}\left(\Sigma_W^{-1}\widehat{C} - \mathrm{Id}_d\right) = 0$*. By multiplying the previous equation by* $\mathcal{V}$ *and using the identity* $\mathcal{V}\Sigma_W^{-1} = \mathrm{Id}_d - WW^\top\Sigma_W^{-1}$*, prove finally that*

$$\mathcal{V} = \mathrm{diag}\left(\widehat{C} - WW^\top\right). \tag{5.14}$$

Upon multiplying (5.13) *by* $\mathscr{W}\mathscr{W}^\top + \mathrm{Id}_d$ *on the left, and by* $\mathscr{W}^\top\left(\mathscr{W}\mathscr{W}^\top + \mathrm{Id}_d\right)$ *on the right, it follows that*

$$\Lambda\mathscr{W}\mathscr{W}^\top = \left(\mathscr{W}\mathscr{W}^\top + \mathrm{Id}_d\right)\mathscr{W}\mathscr{W}^\top. \tag{5.15}$$

The symmetric positive matrix $\mathscr{W}\mathscr{W}^\top \in \mathbb{R}^{d\times d}$ *can be diagonalized as* $\mathscr{W}\mathscr{W}^\top = V^\top DV$*, for some diagonal matrix* D *with nonnegative entries* d_k *(ranked in decreasing order), and an orthogonal matrix* V*.*

(j) *Consider an eigenvector* ξ_k *of* $\mathscr{W}\mathscr{W}^\top$ *associated with the eigenvalue* d_k*. Write the equation on* ξ_k, d_k *obtained from* (5.15)*.*

(k) *How should the eigenvalues* d_k *be chosen in order to minimize* $\mathfrak{L}_{\mathcal{V}}(\mathscr{W})$*?*

In order to find W *for* $\mathcal{V}$ *fixed, one can choose*

$$W = \mathcal{V}^{1/2}U^\top \begin{pmatrix} | & | & \cdots & | \\ \xi_1 & \xi_2 & \cdots & \xi_\ell \\ | & | & \cdots & | \end{pmatrix} \mathrm{diag}\left(\sqrt{\lambda_1 - 1}, \ldots, \sqrt{\lambda_\ell - 1}\right), \tag{5.16}$$

with $\{\xi_k\}_{1\leqslant k\leqslant \ell}$ *the eigenvectors of* Λ *associated with the largest eigenvalues. All the quantities in the previous equation depend only on* $\mathcal{V}$ *and* $\widehat{C}$*. The equation* (5.16)*, together with* (5.14)*, forms the basis for the iterative method discussed in [59, Section 2.2] to find the parameters in factor analysis. One then alternates between* (5.14) *(which says how to update* $\mathcal{V}$ *for* W *given) and* (5.16) *(which says how to update* W *for* $\mathcal{V}$ *given).*

6

Support vector machines

We present in this chapter a method to perform classification by linearly separating data sets, using the method called support vector machines. The name of this method comes from the fact that a small or sparse subset of the training points is retained to construct decision functions whose signs allow to make predictions for the labels of new data points.

We start in Section 6.1 by describing the method for data sets which can be linearly separated in an exact manner; and then make precise in Section 6.2 how to extend the approach to situations where there are some outliers in the data set which prevent an exact linear separation. We also discuss options to perform multiclass classification in Section 6.3. Our presentation for these sections is based on [41, Section 17.3], [40, Chapter 5] and [8, Section 7.1].

In fact, most data sets are not linearly separable as such, even in an approximate manner. Some form of featurization is needed to transform the data. We discuss in Section 6.4 how to perform SVM with implicit features based on kernel methods (see [41, Section 17.1], [40, Chapter 6], [50, Chapter 16] and [4, Chapter 7] for further references on kernel methods).

6.1 Linear classification for separable data sets

We work here in the context of supervised learning, and aim at performing classification for binary data, with label space $\mathcal{Y} = \{-1, 1\}$ (see Section 6.3 for an extension to multiclass classification). We consider a training data set $\mathcal{D}_{\text{train}} = \{(x_i, y_i),\, 1 \leqslant i \leqslant n\} \subset \mathcal{X} \times \mathcal{Y}$, with $\mathcal{X} \subset \mathbb{R}^d$. As for various other learning methods, it is useful to normalize/standardize the inputs before using the methods described in this chapter.

The aim is to find a prediction function minimizing the risk $\mathcal{R}(f) = \mathbb{E}[\mathbf{1}_{y \neq f(x)}]$. The method described here is based on linear classifiers, *i.e.* functions $x \mapsto \text{sign}(w^\top x + b)$ with $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$.

We first make precise the geometric interpretation of the problem in Section 6.1.1, and then formulate the associated minimization of the loss in primal and dual forms, respectively in Sections 6.1.2 and 6.1.3. We then discuss how to train the model in Section 6.1.4, and conclude by providing theoretical guarantees on the learning problem in Section 6.1.5.

6.1.1 Hard margin SVM

We assume in all this section that the data set $\mathcal{D}_{\text{train}}$ can be linearly separated, *i.e.* that there exists $\theta = (w, b) \in \mathbb{R}^d \times \mathbb{R}$ such that

$$\forall 1 \leqslant i \leqslant n, \qquad y_i \left(w^\top x_i + b\right) > 0. \tag{6.1}$$

Data points with labels -1 are therefore separated from data points with labels 1 by the hyperplane of equation $w^\top x + b = 0$ (also called "decision boundary"). However, there may be infinitely many such separating hyperplanes. The idea is to choose the one with the largest margin, *i.e.* the distance of the closest point of the data set to the separating plane should be as large as possible in order to obtain the solution to the classification problem which is the most robust to perturbations in the inputs $(x_i)_{1 \leqslant i \leqslant n}$ of the data set.

Definition 6.1 (Geometric margin). *Consider $w \in \mathbb{R}^d \setminus \{0\}$ and $b \in \mathbb{R}$. The geometric margin ρ_g of a linear classifier $g(x) = w^\top x + b$ at a point $x \in \mathbb{R}^d$ is its Euclidean distance to the hyperplane $g^{-1}\{0\} = \left\{\widetilde{x} \in \mathbb{R}^d \,\middle|\, w^\top \widetilde{x} + b = 0\right\}$:*

$$\rho_g(x) = \frac{\left|w^\top x + b\right|}{\|w\|_2}. \tag{6.2}$$

The geometric margin for a data set $\mathcal{D}_{\text{train}}$ is

$$\rho_g(\mathcal{D}_{\text{train}}) = \min_{1 \leqslant i \leqslant n} \rho_g(x_i).$$

The equality (6.2) is not immediately clear. The reader is asked to prove it in the next exercise.

Exercise 6.1. *Prove* (6.2).

6.1.2 Primal optimization problem

SVM is based on finding the maximum margin hyperplane. Since $|w^\top x_i + b| = y_i(w^\top x_i + b)$ in view of (6.1), the maximization of the margin for the training data set can be formulated as

$$\max_{(w,b) \in \mathbb{R}^d \times \mathbb{R}} \min_{1 \leqslant i \leqslant n} \left\{ \frac{y_i(w^\top x_i + b)}{\|w\|_2} \;\middle|\; \forall 1 \leqslant i \leqslant n, \quad y_i(w^\top x_i + b) > 0 \right\}.$$

This optimization problem can be seen as a maximization over (w, b) with n inequality constraints. Since $y_i(w^\top x_i + b)/\|w\|_2$ is invariant by rescaling of w, b (*i.e.* the function value is the same if w, b are replaced by $\alpha w, \alpha b$ with $\alpha > 0$), it is possible to normalize the learning problem by requiring that

$$\min_{1 \leqslant i \leqslant n} y_i(w^\top x_i + b) = 1.$$

This leads to the reformulation

$$\max_{(w,b) \in \mathbb{R}^d \times \mathbb{R}} \left\{ \frac{1}{\|w\|_2} \;\middle|\; \forall 1 \leqslant i \leqslant n, \quad y_i(w^\top x_i + b) \geqslant 1 \right\}.$$

In fact, as maximizing $1/\|w\|_2$ is equivalent to minimizing $\|w\|_2^2$, the optimization problem can finally be stated in primal form as

$$\min_{(w,b) \in \mathbb{R}^d \times \mathbb{R}} \left\{ \frac{1}{2}\|w\|_2^2 \;\middle|\; \forall 1 \leqslant i \leqslant n, \quad y_i \left(w^\top x_i + b\right) \geqslant 1 \right\}. \tag{6.3}$$

Well posedness. The optimization problem (6.3) is the minimization of the convex function $(w,b) \mapsto \|w\|_2^2$ under affine constraints $c_i(w,b) = y_i(w^\top x_i + b) - 1 \geqslant 0$. The constraints are therefore qualified. Showing the existence of a solution however requires some care as the functional to minimize is not coercive in the b variable. We prove to this end that the minimization can be restricted to a bounded set, in which case the existence of a minimizer is clear since the function to minimize is continuous.

To prove that the minimization can be restricted to a bounded set, we consider an element $(\widetilde{w},\widetilde{b}) \in \mathbb{R}^d \times \mathbb{R}$ which satisfies the constraints $c_i(\widetilde{w},\widetilde{b}) \geqslant 0$ for all $1 \leqslant i \leqslant n$. It is then sufficient to restrict the minimization over $w \in \mathbb{R}^d$ to the minimization over $w \in \overline{B(0,R)}$ with $R = \|\widetilde{w}\|_2 < +\infty$. Next, when $y_i = 1$, the condition $c_i(w,b) \geqslant 0$ is equivalent to $-b \leqslant w^\top x_i - 1$; while, when $y_i = -1$, it is equivalent to $b \leqslant -w^\top x_i - 1$. Finally,

$$\begin{aligned} |b| = \max(b,-b) &\leqslant \max_{1\leqslant i\leqslant n} \max\left\{w^\top x_i - 1, -w^\top x_i - 1\right\} \\ &\leqslant \max_{1\leqslant i\leqslant n} \|w\|_2\|x_i\|_2 \leqslant \left(\max_{1\leqslant i\leqslant n} \|x_i\|_2\right) \|\widetilde{w}\|_2 := B < +\infty. \end{aligned}$$

This allows to conclude that the minimization problem (6.3) can be restricted to $(w,b) \in \overline{B(0,R)} \times [-B,B]$.

Uniqueness cannot be asserted at this stage, although one minimizes a continuous, convex function over a closed, bounded, convex set in finite dimension. In fact, the global minimizer may not be unique, although there cannot be local minima. Any global minimizer satisfies the following necessary conditions of optimality:

$$\begin{cases} w - \displaystyle\sum_{i=1}^n \alpha_i \nabla_w c_i(w,b) = 0, \\ \displaystyle\sum_{i=1}^n \alpha_i \partial_b c_i(w,b) = 0, \\ \forall 1 \leqslant i \leqslant n, \qquad \alpha_i \geqslant 0, \quad \alpha_i \left[y_i\left(w^\top x_i + b\right) - 1\right] = 0, \end{cases}$$

which can be written more explicitly as

$$\begin{cases} w = \displaystyle\sum_{i=1}^n \alpha_i y_i x_i, \\ \displaystyle\sum_{i=1}^n \alpha_i y_i = 0, \\ \forall 1 \leqslant i \leqslant n, \qquad \alpha_i \geqslant 0, \quad \alpha_i \left[y_i\left(w^\top x_i + b\right) - 1\right] = 0. \end{cases} \tag{6.4}$$

Support vectors. The last condition in (6.4) asserts that $\alpha_i = 0$ when $c_i(w,b) > 0$, *i.e.* the constraint is not active. A value $\alpha_i \neq 0$ can be obtained only for points x_i on the marginal hyperplanes $\{x \in \mathbb{R}^d \,|\, w^\top x + b = \pm 1\}$. Data points which do not lie on one of the two marginal hyperplanes do not affect the definition of w; and in fact not the definition of b either, as shown in the next exercise. It is useful for this to introduce the set of active constraints

$$\mathcal{S} = \{1 \leqslant i \leqslant n \,:\, \alpha_i \neq 0\}. \tag{6.5}$$

Then,

$$w = \sum_{i\in\mathcal{S}} \alpha_i y_i x_i.$$

Exercise 6.2. *Show that* $b = \dfrac{1}{|\mathcal{S}|} \displaystyle\sum_{i\in\mathcal{S}} \left(y_i - \sum_{j\in\mathcal{S}} \alpha_j y_j x_j^\top x_i \right)$.

6.1.3 Dual formulation

The motivation for turning to a dual formulation of the primal problem (6.3) is twofold:

- it is useful to formulate the problem an infinite dimensional setting based on kernel functions (see Section 6.4);
- on the algorithmic side, some reference numerical methods solve the dual problem rather than the primal one (see Section 6.1.4).

To write the dual problem, we introduce the Lagrangian

$$L(w,b,\alpha) = \frac{1}{2}\|w\|_2^2 - \sum_{i=1}^n \alpha_i \left[y_i \left(w^\top x_i + b \right) - 1 \right].$$

The primal minimization problem (6.1.4) corresponds to

$$\min_{(w,b)\in\mathbb{R}^d\times\mathbb{R}} \max_{\alpha\in\mathbb{R}_+^n} L(w,b,\alpha).$$

It always holds that

$$\max_{\alpha\in\mathbb{R}_+^n} \min_{(w,b)\in\mathbb{R}^d\times\mathbb{R}} L(w,b,\alpha) \leqslant \min_{(w,b)\in\mathbb{R}^d\times\mathbb{R}} \max_{\alpha\in\mathbb{R}_+^n} L(w,b,\alpha), \tag{6.6}$$

as can be seen from the inequality

$$L(w,b,\alpha) \leqslant \max_{\alpha'\in\mathbb{R}_+^n} L(w,b,\alpha'),$$

then taking the minimum over $(w,b) \in \mathbb{R}^d \times \mathbb{R}$ first on the left hand side, next on the right hand side, and finally considering the maximum over $\alpha \in \mathbb{R}_+^n$ on the left hand side. In fact, there is no duality gap in (6.6), *i.e.* the inequality in (6.6) is in fact an equality. This comes from the fact that the constraints are qualified and convex, and the objective function is convex. The primal optimization problem (6.3) can finally be reformulated in dual form as

$$\max_{\alpha\in\mathbb{R}_+^n} F_\alpha(w,b), \qquad F_\alpha(w,b) = \min_{(w,b)\in\mathbb{R}^d\times\mathbb{R}} \left\{ \frac{1}{2}\|w\|_2^2 - \sum_{i=1}^n \alpha_i \left[y_i \left(w^\top x_i + b \right) - 1 \right] \right\}.$$

In fact, the inner minimization can be performed explicitly, as shown in the following exercise.

Exercise 6.3. *Prove that the dual problem rewrites*

$$\max_{\alpha\in\mathbb{R}_+^n} \left\{ \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i\alpha_j \left(x_i^\top x_j \right) y_i y_j \;\middle|\; \sum_{i=1}^n \alpha_i y_i = 0 \right\}. \tag{6.7}$$

In its formulation (6.7), the dual optimization problem is the maximization of a quadratic concave function under affine constraints. The function is indeed concave since the Hessian, which is the matrix with entries $-(x_i^\top x_j)y_iy_j$, is negative:

$$\forall \xi \in \mathbb{R}^n, \qquad -\sum_{i,j}^n (x_i^\top x_j) y_i y_j \xi_i \xi_j = -\left\| \sum_{i=1}^n \xi_i y_i x_i \right\|^2 \leqslant 0.$$

6.1.4 Optimization algorithms

Both the primal and dual problems are quadratic programming problems (*i.e.* they correspond to minimizing quadratic functions under affine constraints). There is a huge literature on solvers for these problems, but the challenge is to avoid relying on generic quadratic programming methods, whose computational cost scales as $\mathrm{O}(n^3)$. The implementation in scikit-learn relies on the C library `libsvm` (see [11]). It relies on a version of the algorithm known as Sequential Minimal Optimization (see Exercise 6.5), where the dual problem (6.7) is analytically solved when the minimization is performed only over two coefficients α_k, α_ℓ, the other ones being fixed; going over all possible couples sequentially or randomly.

6.1.5 Learning guarantees by leave-one-out analysis

We provide bounds on the expected risk in this section. The material here is more advanced and can be skipped at first reading. We consider the framework of K-fold cross validation for $K = n$, which corresponds to the so-called leave-one-out cross validation (recall Section 1.3.2.2). It consists in taking out one point of the training data set, performing training on the remaining $n-1$ points, computing the prediction error on a validation set composed of the point that was taken out; and averaging this over the n possible choices for the single point validation set. More precisely,

$$\widehat{\mathcal{R}}_{\mathrm{LOO}}(\mathcal{D}_{\mathrm{train}}) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{f_{\mathcal{D}_{\mathrm{train}}\setminus\{(x_i,y_i)\}}(x_i)\neq y_i}, \tag{6.8}$$

where $f_{\mathcal{D}_{\mathrm{train}}\setminus\{(x_i,y_i)\}}$ is the prediction function obtained from training on the data set $\mathcal{D}_{\mathrm{train}} \setminus \{(x_i, y_i)\}$ (*i.e.* the predictor $x \mapsto \mathrm{sign}(w^\top x + b)$ with w, b found by solving (6.3) or (6.7) with $\mathcal{D}_{\mathrm{train}}$ replaced by $\mathcal{D}_{\mathrm{train}} \setminus \{(x_i, y_i)\}$).

In order to characterize the validation error, we note that removing a data point which is not a support vector does not change the solution. Bounds on the validation error are therefore related to the fraction of support vectors. We write here bounds in expectation over realizations of the training set $\mathcal{D}_{\mathrm{train}}$, distributed according to $p_{\mathrm{data}}^{\otimes n}$. We start by providing a general result on the average leave-one-out error (see [40, Lemma 5.3]).

Lemma 6.1. *The average leave-one-out validation error for a data set $\mathcal{D}_{\mathrm{train}}^n$ of size $n \geqslant 2$ is an unbiased estimator of the average expected risk of a data set $\mathcal{D}_{\mathrm{train}}^{n-1}$ of size $n-1$:*

$$\mathbb{E}_{\mathcal{D}_{\mathrm{train}}^n \sim p_{\mathrm{data}}^{\otimes n}}\left[\widehat{\mathcal{R}}_{\mathrm{LOO}}\left(\mathcal{D}_{\mathrm{train}}^n\right)\right] = \mathbb{E}_{\mathcal{D}_{\mathrm{train}}^{n-1} \sim p_{\mathrm{data}}^{\otimes n-1}}\left[\mathcal{R}\left(f_{\mathcal{D}_{\mathrm{train}}^{n-1}}\right)\right],$$

where $f_{\mathcal{D}_{\mathrm{train}}^{n-1}}$ is the prediction function obtained by training over $\mathcal{D}_{\mathrm{train}}^{n-1}$.

Proof. By using first the definition of the leave-one-out error, then the independence of the data points,

$$\begin{aligned}
\mathbb{E}_{\mathcal{D}_{\mathrm{train}}^n \sim p_{\mathrm{data}}^{\otimes n}}\left[\widehat{\mathcal{R}}_{\mathrm{LOO}}\left(\mathcal{D}_{\mathrm{train}}^n\right)\right] &= \frac{1}{n}\sum_{i=1}^{n} \mathbb{E}_{\mathcal{D}_{\mathrm{train}}^n \sim p_{\mathrm{data}}^{\otimes n}}\left[\mathbf{1}_{f_{\mathcal{D}_{\mathrm{train}}^n\setminus\{(x_i,y_i)\}}(x_i)\neq y_i}\right] \\
&= \mathbb{E}_{\mathcal{D}_{\mathrm{train}}^n \sim p_{\mathrm{data}}^{\otimes n}}\left[\mathbf{1}_{f_{\mathcal{D}_{\mathrm{train}}^n\setminus\{(x_1,y_1)\}}(x_1)\neq y_1}\right] \\
&= \mathbb{E}_{\mathcal{D}_{\mathrm{train}}^{n-1} \sim p_{\mathrm{data}}^{\otimes n-1}}\left(\mathbb{E}_{(x_1,y_1)\sim p_{\mathrm{data}}}\left[\mathbf{1}_{f_{\mathcal{D}_{\mathrm{train}}^{n-1}}(x_1)\neq y_1}\right]\right).
\end{aligned}$$

This leads to the claimed result since the expectation between brackets on the right hand side of the previous equality is $\mathcal{R}\left(f_{\mathcal{D}_{\mathrm{train}}^{n-1}}\right)$. □

The interest of the leave-one-out estimator for SVM (which is very expensive to compute in general unless there is some algebraic simplification which makes it unnecessary to solve n optimization problems) is that it can be easily related to the fraction of support points in the data.

Theorem 6.1. *Assume that the training sets $\mathcal{D}_{\mathrm{train}}^n$ are linearly separable for any $n \geqslant 2$. Recall the definition* (6.5) *of the set of active constraints, and denote by $N_{\mathrm{SV}}(\mathcal{D}_{\mathrm{train}}) = |\mathcal{S}|$ the number of support vector defining the prediction function*

$$f_{\mathcal{D}_{\mathrm{train}}}(x) = \mathrm{sign}\left(\sum_{i\in\mathcal{S}} \alpha_i y_i x_i^\top x + b\right).$$

Then the expected risk can be bounded as

$$\mathbb{E}_{\mathcal{D}_{\mathrm{train}}^n \sim p_{\mathrm{data}}^{\otimes n}}\left[\mathcal{R}\left(f_{\mathcal{D}_{\mathrm{train}}^n}\right)\right] \leqslant \mathbb{E}_{\mathcal{D}_{\mathrm{train}}^{n+1} \sim p_{\mathrm{data}}^{\otimes n+1}}\left[\frac{N_{\mathrm{SV}}(\mathcal{D}_{\mathrm{train}}^{n+1})}{n+1}\right].$$

Proof. By Lemma 6.1, the left hand side of the inequality to prove is equal to

$$\mathbb{E}_{\mathcal{D}_{\text{train}}^{n+1} \sim p_{\text{data}}^{\otimes n+1}} \left[\widehat{\mathcal{R}}_{\text{LOO}} \left(\mathcal{D}_{\text{train}}^{n+1} \right) \right].$$

Now, in the leave-one-out estimator (6.8), the predictors which appear in the sum are unchanged when the data point which is removed is not a support vector, and the prediction at this point is therefore correct as the data sets are assumed to be linearly separable. To misclassify, the removed data point needs to be a support vector, so that

$$\widehat{\mathcal{R}}_{\text{LOO}} \left(\mathcal{D}_{\text{train}}^{n+1} \right) \leqslant \frac{N_{\text{SV}}(\mathcal{D}_{\text{train}}^{n+1})}{n+1},$$

from which the result follows by taking expectations over realizations of the data set on both sides. □

6.2 Linear classification for non separable data sets

Data sets are often not linearly separable. The constraints then need to be relaxed in order to allow for some misclassification of the data. The deviation to the constraint $y_i(w^\top x_i + b) \geqslant 1$ is controlled using slack variables $\xi_i \geqslant 0$, namely

$$y_i \left(w^\top x_i + b \right) \geqslant 1 - \xi_i.$$

Outliers correspond to $\xi_i > 0$. The decision function then equilibriates between large margins (which may lead to a higher number of outliers) and the limitation of the slack. This approach is known as soft margin SVM.

6.2.1 Soft margin SVM

The primal optimization problem generalizes (6.3) as

$$\min_{(w,b,\xi) \in \mathbb{R}^d \times \mathbb{R} \times \mathbb{R}_+^n} \left\{ \frac{1}{2} \|w\|_2^2 + C \sum_{i=1}^n \xi_i^p \;\middle|\; \forall 1 \leqslant i \leqslant n, \quad y_i \left(w^\top x_i + b \right) \geqslant 1 - \xi_i \right\}, \tag{6.9}$$

where $p \geqslant 1$ is some exponent and $C > 0$ allows to control the amount of slack. The parameter C, which is akin to some (inverse) regularization parameter, is typically determined by (cross) validation. Hard SVM is recovered in the limit $C \to +\infty$, where no slack is allowed. Large values of C allow to focus attention on points close to the decision boundaries, as outliers with values of the slack not sufficiently small are strongly penalized.

The most common choices for the exponent p are $p = 1$, which is the value we consider in the sequel, and $p = 2$.

Remark 6.1 (Interpretation as convex surrogate). *The minimization problem* (6.9) *can be rewritten as*

$$\min_{(w,b) \in \mathbb{R}^d \times \mathbb{R}} \left\{ \frac{1}{n} \sum_{i=1}^n \max \left\{ 0, 1 - y_i \left(w^\top x_i + b \right) \right\}^p + \frac{1}{2nC} \|w\|_2^2 \right\}.$$

This formulation makes it apparent that this amounts to a classification problem using convex surrogate functions

$$\Phi_p(u) = \max\{0, 1-u\}^p,$$

and a ridge penalization with strength $\lambda = 1/(2nC)$. *The cases* $p = 1$ *and* $p = 2$ *respectively correspond to the hinge loss and the squared hinge loss.*

6.2.2 Characterization of minimizers

We write here the characterization of the minimizer of (6.9) for $p = 1$ (existence can be proved in Section 6.1.2 since the values of ξ_i can be restricted to a bounded set). The computations are similar to those leading to (6.4). More precisely, the minimization problem under consideration corresponds to minimizing the function

$$f(w, b, \xi) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i,$$

under the constraints

$$c_i(w, b, \xi) = y_i\left(w^\top x_i + b\right) - 1 + \xi_i \geqslant 0, \qquad \widetilde{c}_i(w, b, \xi) = \xi_i \geqslant 0.$$

The associated necessary conditions to be satisfied by a global minimizer read

$$\left\{\begin{array}{rl} & \nabla_w f(w, b, \xi) - \displaystyle\sum_{i=1}^{n} \alpha_i \nabla_w c_i(w, b, \xi) = 0, \\ & \partial_b f(w, b, \xi) - \displaystyle\sum_{i=1}^{n} \alpha_i \partial_b c_i(w, b, \xi) = 0, \\ \forall 1 \leqslant i \leqslant n, \qquad & \partial_{\xi_i} f(w, b, \xi) - \alpha_i \partial_{\xi_i} c_i(w, b, \xi) - \beta_i \partial_{\xi_i} \widetilde{c}_i(w, b, \xi) = 0, \\ & \alpha_i c_i(w, b, \xi) = \beta_i \widetilde{c}_i(w, b, \xi) = 0, \\ & \alpha_i \geqslant 0, \quad \beta_i \geqslant 0, \end{array}\right.$$

so that

$$w = \sum_{i=1}^{n} \alpha_i y_i x_i,$$

with

$$\sum_{i=1}^{n} \alpha_i y_i = 0,$$

and

$$\forall 1 \leqslant i \leqslant n, \qquad \alpha_i + \beta_i = C, \qquad \alpha_i c_i(w, b, \xi) = \beta_i \widetilde{c}_i(w, b, \xi) = 0.$$

Recalling the definition (6.5) of the set of active constraints, we find that

$$w = \sum_{i \in \mathcal{S}} \alpha_i y_i x_i.$$

An index $1 \leqslant i \leqslant n$ belongs to $\mathcal{S}$ only if $y_i(w^\top x_i + b) - 1 + \xi_i = 0$, *i.e.* the constraint $c_i(w, b, \xi) \geqslant 0$ is active. There are therefore two types of support vectors:

- $\xi_i = 0$ and $y_i(w^\top x_i + b) = 1$, so that x_i belongs to one of the two marginal hyperplanes, as for separable data sets;
- $\xi_i > 0$, in which case x_i is an outlier. One necessarily has $\beta_i = 0$ in this case, so that $\alpha_i = C$.

An alternative interpretation of the result is obtained by looking at the values of $\alpha_i \in [0, C]$:

- if $\alpha_i = 0$, the data point is ignored for the prediction;
- if $\alpha_i \in (0, C)$, then $\xi_i = 0$. The data point is used for the prediction, and lies on one of the two marginal hyperplanes. As in Exercise 6.2, it can be shown that the value of b is obtained from data points on the marginal hyperplanes.
- if $\alpha_i = C$, then the data point can lie (but need not to) inside the margin region, either correctly classified if $\xi_i < 1$, or misclassified if $\xi_i > 1$.

6.2.3 Dual formulation

To obtain the dual formulation of the primal problem (6.9), we introduce the Lagrangian

$$L(w,b,\xi,\alpha,\beta) = \frac{1}{2}\|w\|_2^2 + C\sum_{i=1}^n \xi_i - \sum_{i=1}^n \alpha_i \left[y_i\left(w^\top x_i + b\right) - 1 + \xi_i\right] - \sum_{i=1}^n \beta_i \xi_i.$$

The dual problem reads

$$\max_{\alpha,\beta\in\mathbb{R}_+^n} \left\{ \min_{(w,b,\xi)\in\mathbb{R}^d\times\mathbb{R}\times\mathbb{R}_+^n\}} L(w,b,\xi,\alpha,\beta) \right\}.$$

We can then follow the same derivation as in Section 6.1.3, and obtain the following result, similar to the one of Exercise 6.3.

Exercise 6.4. *Prove that the dual version of the primal problem* (6.9) *writes*

$$\max_{\alpha\in[0,C]^n} \left\{ \sum_{i=1}^n \alpha_i - \frac{1}{2}\sum_{i,j=1}^n \alpha_i\alpha_j \left(x_i^\top x_j\right) y_i y_j \;\middle|\; \sum_{i=1}^n \alpha_i y_i = 0 \right\}. \tag{6.10}$$

Note that (6.10) coincides with (6.7) except that the components of α are restricted to lie in the interval $[0,C]$ instead of $\mathbb{R}_+$.

6.3 Multiclass SVM

SVM is intrinsically a method for binary classification. It can however be transformed for multiclass classification. There are two main strategies to this end: combine pairwise decision functions for all possible couples of labels ("one-vs-one" strategy), or base predictions on decision functions where one label is separated from all other ones ("one-vs-all" approach). We successively describe both strategies, for a situation where the labels of the data set belong to $\mathcal{Y} = \{1,\dots,K\}$.

One-vs-one. For any $1 \leqslant k,k' \leqslant K$ with $k \neq k'$, we consider the prediction functions $f_{k,k'}$ obtained by training SVM on the data set containing only points for which the label is either k or k'. For a test input x', the prediction $y' \in \mathcal{Y}$ is the index which has obtained the largest number of assignments for the $K(K-1)/2$ classifiers under consideration. More precisely,

$$y' \in \operatorname*{argmax}_{y\in\mathcal{Y}} \left\{ \sum_{1\leqslant k<k'\leqslant K} \mathbf{1}_{f_{k,k'}(x')=y} \right\}.$$

This approach is however expensive when K is large as one needs to train $K(K-1)/2$ models.

One-vs-all. We select an index $k \in \{1,\dots,K\}$, and transform the initial training set into a training set where there are only two classes, one corresponding to data points with labels k, and the other one gathering all data points whose labels differ from k. This amounts to considering the inputs x_i for $1 \leqslant i \leqslant n$ with new labels $\widetilde{y}_i \in \{-1,1\}$, with

$$\widetilde{y}_i = \mathbf{1}_{y_i=k} - \mathbf{1}_{y_i\neq k}.$$

We then perform SVM as described in the binary case. The corresponding decision function is denoted by g_k. For a new test input x', one then selects the label based on some form of maximal separation, for instance

$$y' \in \operatorname*{argmax}_{1\leqslant k\leqslant K} g_k(x'),$$

i.e. the class label which lead to the largest margin; or

$$y' \in \operatorname*{argmax}_{1 \leqslant k \leqslant K} \log \frac{p_k(y=1 \,|\, x')}{p_k(y=-1 \,|\, x')},$$

where p_k is some probabilistic model (whose hyperparameters should be calibrated with some maximum likelihood procedure on a separate validation set in order to avoid overfitting; this is known as Platt scaling, see [41, Sections 17.3.5 and 17.3.7]). There are however two difficulties in this approach. The first one is class imbalance, as there will typically be many more points with labels in $\{1, \dots, K\} \setminus \{k\}$ than points with labels k. The second one is that the various the values of the decision functions g_k are difficult to compare as the corresponding classification problems may have quite different scales; similarly, the probabilities $p_k(y = \pm 1 \,|\, x')$ may be hard to compare.

6.4 Kernel SVM

There are two difficulties with classification using SVM, and in general with other linear methods, either for classification (think of logistic regression), regression (think of least square regression) or dimensionality reduction (think of PCA):

- the data may not be well represented with a linear model. In the context of classification, this means that the data set may not be linearly separable. On the other hand, it could become linearly separable once a nonlinear featurization has been performed with some feature function ϕ. This function may have values in very high dimensional spaces, as it may be necessary to go to quite high dimensions in order to linearly separate the data. In any case, this approach allows to obtain non affine decision boundaries;
- the computational cost of the method can become important as the dimension of the number of features is increased. The kernel trick, as described below (and already seen in a particular case in Section 5.5.2), allows to alleviate this issue by ensuring that the cost of the numerical methods depends on the number of data points n and not on the dimension of the feature function.

We start by presenting the principle of Kernel SVM in Section 6.4.1, and then give a mathematically more precise introduction to kernel methods in Section 6.4.2.

add plot of decision function in this NL context

6.4.1 Principle of the method

The principle of kernel methods is to rewrite all predictions using expressions such as

$$\mathcal{K}(x, x') = \phi(x) \cdot \phi(x'), \tag{6.11}$$

which should be understood as the scalar product between two elements of a (possibly infinite dimensional) Hilbert space. For SVMs, predictions using kernel methods rely on $w^\top \phi(x) + b$, where the expressions of w, b are the ones obtained in Sections 6.1 and 6.2 but with x_i replaced by $\phi(x_i)$:

$$w = \sum_{i \in \mathcal{S}} \alpha_i y_i \phi(x_i), \qquad b = \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \left(y_i - \sum_{j \in \mathcal{S}} \alpha_j y_j \phi(x_j)^\top \phi(x_i) \right).$$

Therefore,

$$w^\top \phi(x) + b = \sum_{i \in \mathcal{S}} \alpha_i y_i \mathcal{K}(x, x_i) + \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \left(y_i - \sum_{j \in \mathcal{S}} \alpha_j y_j \mathcal{K}(x_j, x_i) \right)$$

can be computed with the knowledge of the kernel $\mathcal{K}$ only. Moreover, the values of the coefficients $\alpha \in \mathbb{R}_+^n$ can be found by considering the dual problem derived similarly to (6.10):

$$\max_{\alpha \in [0,C]^n} \left\{ \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j \mathcal{K}(x_i, x_j) \;\middle|\; \sum_{i=1}^n \alpha_i y_i = 0 \right\}.$$

This makes it possible to substantially enlarge the dimension of the features, and in fact consider infinite dimensional features, as long as the associated kernel $\mathcal{K}$ is easy to evaluate. In fact, in practice, one chooses the kernel function $\mathcal{K}$, in which case the associated features are implicit. Indeed, the feature function ϕ needs not be known explicitly as it suffices to rely on a result saying that there exist a Hilbert space $\mathcal{H}$ and a feature function $\phi : \mathcal{X} \to \mathcal{H}$ associated with a kernel $\mathcal{K} : \mathcal{X}^2 \to \mathbb{R}$ according to (6.11). This is discussed in the next section.

Exercise 6.5 (Sequential minimization optimization for SVM). *The aim of this exercise is to show how to reduce the potentially large quadratic optimization problem corresponding to the dual formulation of soft margin SVM into a series of smaller optimization problems involving only two Lagrange multipliers. This requires less memory and facilitates the implementation of the algorithm. Recall the dual formulation for soft margin SVM with regularization coefficient $C > 0$:*

$$\max_{\alpha \in [0,C]^n} \left\{ \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i,j=1}^n \alpha_i \alpha_j y_i y_j K_{ij} \;\middle|\; \sum_{i=1}^n \alpha_i y_i = 0 \right\}, \tag{6.12}$$

with $K_{ij} = \mathcal{K}(x_i, x_j)$ for some positive definite symmetric kernel function K. We assume that all data points are different: $x_i \neq x_j$ for all $1 \leqslant i, j \leqslant n$.

(a) Show that $\mathcal{K}(x,x)\mathcal{K}(x',x') \geqslant \mathcal{K}(x,x')^2$ for any $x, x' \in \mathcal{X}$.

We assume in the remainder of this exercise that

$$\forall x \neq x', \qquad \mathcal{K}(x,x)\mathcal{K}(x',x') > \mathcal{K}(x,x')^2.$$

(b) Show that, when the maximization in (6.12) *is restricted to a maximization over (α_1, α_2), the problem reduces to*

$$\max_{0 \leqslant \alpha_1, \alpha_2 \leqslant C} \{\Phi(\alpha_1, \alpha_2) \mid \alpha_1 + s\alpha_2 = \gamma\}, \tag{6.13}$$

where $s = y_1 y_2 \in \{-1, 1\}$,

$$\Phi(\alpha_1, \alpha_2) = \alpha_1 + \alpha_2 - \frac{1}{2} K_{11} \alpha_1^2 - \frac{1}{2} K_{22} \alpha_2^2 - s K_{12} \alpha_1 \alpha_2 - y_1 \alpha_1 v_1 - y_2 \alpha_2 v_2,$$

and the parameters $\gamma, v_1, v_2 \in \mathbb{R}$ should be made precise.

(c) We next substitute in the expression of Φ the relation $\alpha_1 = \gamma - s\alpha_2$ obtained from the equality constraint between α_1 and α_2. This leads to the function $\Psi(\alpha_2) = \Phi(\gamma - s\alpha_2, \alpha_2)$. Prove that the unconstrained *minimization of Ψ over $\alpha_2 \in \mathbb{R}$ is well defined and*

$$\alpha_2^{\text{opt}} = \frac{s(K_{11} - K_{12})\gamma + y_2(v_1 - v_2) - s + 1}{\eta},$$

with $\eta > 0$ to identify.

(d) We now consider an iterative procedure where the coefficients are α^t at iteration t, and a new coefficient $\widetilde{\alpha}^{t+1}$ is obtained by updating only (α_1^t, α_2^t) via the minimization (6.13)*. Denoting by v_1^t, v_2^t, γ^t the quantities defined in Question (b) when α is replaced by α^t, and*

$$b^t = \frac{1}{|S^t|} \sum_{i \in S^t} \left(y_i - \sum_{j \in S^t} \alpha_j^t y_j \mathcal{K}(x_i, x_j) \right)$$

the offset found at the previous iteration (with S^t the set of support vectors, i.e. *the set of indices i such that $\alpha_i^t \neq 0$), prove that*

$$v_1^t - v_2^t = f_t(x_1) - f_t(x_2) + \alpha_2^t y_2 \eta - s y_2 \gamma^t (K_{11} - K_{12}),$$

where f_t is the decision function at step t:

$$f_t(x) = \sum_{i=1}^n \alpha_i^t y_i \mathcal{K}(x, x_i) + b^t.$$

(e) Show that $\widetilde{\alpha}_2^{t+1} = \alpha_2^t + \dfrac{y_2}{\eta}\left(y_2 - f_t(x_2) - y_1 + f_t(x_1)\right)$.

For $s = 1$ *define* $L^t = \max(0, \gamma^t - C)$ *and* $U^t = \min(C, \gamma^t)$*; and* $L^t = \max(0, -\gamma^t)$ *and* $U^t = \min(C, C - \gamma^t)$ *for* $s = -1$*. The update of the sequential minimization optimization algorithm corresponds to clipping the value of* $\widetilde{\alpha}_2^{t+1}$ *as*

$$\alpha_2^{t+1} = \begin{cases} L^t & \text{if } \widetilde{\alpha}_2^{t+1} \leqslant L^t, \\ \widetilde{\alpha}_2^{t+1} & \text{if } L^t \leqslant \widetilde{\alpha}_2^{t+1} \leqslant U^t, \\ U^t & \text{if } \widetilde{\alpha}_2^{t+1} \geqslant U^t. \end{cases}$$

The value of α_1^{t+1} *is then deduced from the value of* α_2^{t+1} *as* $\alpha_1^{t+1} = \alpha_1^t + s\left(\alpha_2^t - \alpha_2^{t+1}\right)$.

(f) Why is clipping required? How are the lower/upper bounds on α_2^t *obtained? Why is* α_1^{t+1} *updated in this way?*

The procedure we described here allows to update a pair of coordinates (chosen here to be the first two ones for simplicity). The algorithm then loops over all possible pairs until convergence. Since only two coefficients are changed at each iteration, the update of the values of the function f_t *in Question (d) is unexpensive.*

6.4.2 Kernel methods

We provide in this section the rigorous definition of kernel functions, and give conditions characterizing admissible kernels.

Definition 6.2. *A kernel* $\mathcal{K} : \mathcal{X}^2 \to \mathbb{R}$ *is said to be positive definite symmetric (PDS) if, for any* $(x_1, \ldots, x_n) \in \mathcal{X}^n$*, the matrix* $K \in \mathbb{R}^{n \times n}$ *with entries* $K_{i,j} = \mathcal{K}(x_i, x_j)$ *is symmetric positive semidefinite (*i.e. *for any* $t \in \mathbb{R}^n$*, it holds* $t^\top K t \geqslant 0$*, or equivalently* K *is real symmetric with nonnegative eigenvalues).*

The following important result, due to Aronszajn in 1950, characterizes PDS kernels.

Theorem 6.2. *The function* $\mathcal{K} : \mathcal{X}^2 \to \mathbb{R}$ *is a PDS kernel if and only if there exist a Hilbert space* $\mathcal{H}$ *and a function* $\phi : \mathcal{X} \to \mathcal{H}$ *such that*

$$\mathcal{K}(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{H}}. \tag{6.14}$$

Exercise 6.6. *The aim of this exercise is to prove Theorem 6.2.*

(a) Assume that there exist a Hilbert space $\mathcal{H}$ *and a function* $\phi : \mathcal{X} \to \mathcal{H}$ *such that* $\mathcal{K}(x, x') = \langle \phi(x), \phi(x') \rangle_{\mathcal{H}}$*. Prove that* $\mathcal{K}$ *is a PDS kernel.*

We assume conversely in the sequel that $\mathcal{K}$ *is a PDS kernel. We first construct a Hilbert space* $\mathcal{H}$ *of functions with argument in* $\mathcal{X}$ *by completion of finite linear combinations of elementary functions* $x \mapsto \mathcal{K}(x, x_i)$*:*

$$\mathcal{H}_0 = \left\{ \sum_{i \in I} t_i \mathcal{K}(\cdot, x_i), \quad t_i \in \mathbb{R},\, x_i \in \mathcal{X},\, |I| < +\infty \right\}.$$

This vector space is endowed with the inner product

$$\langle f, g \rangle_{\mathcal{H}} = \sum_{i=1}^{n} \sum_{j=1}^{m} t_i t'_j \mathcal{K}(x_i, x'_j), \qquad f = \sum_{i=1}^{n} t_i \mathcal{K}(\cdot, x_i), \quad g = \sum_{j=1}^{m} t'_j \mathcal{K}(\cdot, x'_j).$$

(b) Show that $\langle f, g \rangle_{\mathcal{H}}$ *does not depend on the representatives of* f *and* g*.*
(c) Prove that $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ *is a symmetric bilinear form with nonnegative values.*

(d) Find a function ϕ for which (6.14) *holds.*
(e) Show that

$$\langle f, \mathcal{K}(\cdot, x)\rangle_{\mathcal{H}}^2 \leqslant \langle f, f\rangle_{\mathcal{H}} \mathcal{K}(x, x), \tag{6.15}$$

and that $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ is a scalar product.
(f) Conclude.

Note that (6.15) ensures that $f \mapsto \langle f, \mathcal{K}(\cdot, x)\rangle_{\mathcal{H}}$ is a Lipschitz mapping from $\mathcal{H}_0$ to $\mathbb{R}$, and hence can be extended to a Lipschitz mapping on $\mathcal{H}$. In particular,

$$\forall f \in \mathcal{H}, \qquad f(x) = \langle \mathcal{K}(\cdot, x), f\rangle_{\mathcal{H}}. \tag{6.16}$$

This property motivates the terminology *Reproducing Kernel Hilbert Space* (RKHS) for the Hilbert space $\mathcal{H}$. The latter Hilbert space is also called feature space, with associated feature map $\phi : \mathcal{X} \to \mathcal{H}$ given by $\phi(x) = \mathcal{K}(\cdot, x) \in \mathcal{H}$. Note that $\mathcal{H}$ contains only continuous functions as one needs to make sense of pointwise values through (6.16). It is typically a space of rather smooth functions, such as a Sobolev space of sufficiently high order.

We next give two examples of PDS kernels, and make precise the associated feature maps.

Example 6.1 (Polynomial kernels). *For $p \in \mathbb{N}$, define*

$$\mathcal{K}(x, x') = (1 + x \cdot x')^p . \tag{6.17}$$

Let us construct, for pedagogical reasons, the feature map when $d = 2$ and $p = 2$. More precisely, for $x = (x_1, x_2) \in \mathbb{R}^2$, introduce

$$\phi(x) = \left(x_1^2, x_2^2, \sqrt{2}x_1x_2, \sqrt{2}x_1, \sqrt{2}x_2, 1\right)^\top \in \mathbb{R}^6.$$

Then, for $x, x' \in \mathbb{R}^2$,

$$\begin{aligned}\phi(x) \cdot \phi(x') &= x_1^2(x_1')^2 + x_2^2(x_2')^2 + 2x_1x_1'x_2x_2' + 2x_1x_1' + 2x_2x_2' + 1 = (1 + x_1x_1' + x_2x_2')^2 \\ &= (1 + x \cdot x')^2 ,\end{aligned}$$

which is indeed (6.17) *for $p = 2$.*

Example 6.2 (Gaussian kernels). *For $\sigma > 0$, define*

$$\mathcal{K}(x, x') = \mathrm{e}^{-\|x-x'\|^2/(2\sigma^2)}.$$

This kernel can be obtained from

$$\widetilde{\mathcal{K}}(x, x') = \mathrm{e}^{-x\cdot x'/\sigma^2}$$

by a renormalization procedure:

$$\mathcal{K}(x, x') = \frac{\widetilde{\mathcal{K}}(x, x')}{\sqrt{\widetilde{\mathcal{K}}(x, x)}\sqrt{\widetilde{\mathcal{K}}(x', x')}}.$$

To prove that $\mathcal{K}$ is PDS, it therefore suffices in view of [40, Lemma 6.9] to prove that $\widetilde{\mathcal{K}}$ is PDS. The latter statement can be obtained either by relying on Fourier analysis using Bochner's theorem (see [40, Theorem 6.24] or [4, Theorem 7.3]), or by explicitly constructing ϕ by noting that

$$\widetilde{\mathcal{K}}(x, x') = \sum_{p=0}^{+\infty} \frac{(x \cdot x')^p}{p!\sigma^{2p}}.$$

There are of course many other examples of PDS kernels. Given PDS kernels, one can also construct new PDS kernels by various rules (sums and products of PDS kernels, composition of PDS kernels with power series with nonnegative coefficients, etc).

Exercise 6.7. *Consider a PDS kernel $\mathcal{K}$. Prove that*

$$\forall x, x' \in \mathcal{X}, \qquad \mathcal{K}(x, x')^2 \leqslant \mathcal{K}(x, x)\mathcal{K}(x', x').$$

We conclude this section by a very important result, the representer theorem, which states that predictors can be written as a linear combination of $\{\mathcal{K}(\cdot, x_i)\}_{1 \leqslant i \leqslant n}$. The following result allows to generalize manipulations performed in Section 6.4.1 for SVM.

Theorem 6.3 (Representer theorem). *Consider a Hilbert space $\mathcal{H}$ and a feature map $\phi : \mathcal{X} \to \mathcal{H}$. Fix $(x_1, \ldots, x_n) \in \mathcal{X}^n$ and consider $\Psi : \mathbb{R}^{n+1} \to \mathbb{R}$ nondecreasing with respect to the last variable. Then,*

$$f^\star \in \operatorname*{argmin}_{f \in \mathcal{H}} \Psi\left(\langle f, \phi(x_1)\rangle_{\mathcal{H}}, \ldots, \langle f, \phi(x_n)\rangle_{\mathcal{H}}, \|f\|_{\mathcal{H}}^2\right)$$

is of the form

$$f^\star = \sum_{i=1}^n t_i \phi(x_i) \in \mathcal{H}.$$

This result can be applied to PDS kernels $\mathcal{K} : \mathcal{X}^2 \to \mathbb{R}$ by considering the associated RKHS $\mathcal{H}$ and $\phi(x_i) = \mathcal{K}(\cdot, x_i)$. In this case, any minimizer

$$f^\star \in \operatorname*{argmin}_{f \in \mathcal{H}} \Psi\left(f(x_1), \ldots, f(x_n), \|f\|_{\mathcal{H}}^2\right)$$

can be written as

$$f^\star = \sum_{i=1}^n t_i \mathcal{K}(\cdot, x_i).$$

Exercise 6.8. *The aim of this exercise is to prove Theorem 6.3. Introduce the following finite dimensional vector subspace of $\mathcal{H}$:*

$$\mathcal{H}_{x_1,\ldots,x_n} = \left\{ \sum_{i=1}^n t_i \phi(x_i),\ t \in \mathbb{R}^n \right\} \subset \mathcal{H},$$

which is the linear span of the feature vectors associated with the training data set. Decompose f into its components in $\mathcal{H}$ and $\mathcal{H}^\perp$ to show that the minimization can be restricted to functions in $\mathcal{H}$, and conclude.

Let us apply Theorem 6.3 to typical training problems in supervised learning, which read (with some squared penalty term)

$$\min_{f \in \mathcal{H}} \left\{ \frac{1}{n} \sum_{i=1}^n \ell\left(y_i, f(x_i)\right) + \lambda \|f\|_{\mathcal{H}}^2 \right\}.$$

The minimization can be restricted to functions of the form

$$f = \sum_{i=1}^n t_i \mathcal{K}(\cdot, x_i),$$

which therefore amounts to minimizing over $t = (t_1, \ldots, t_n) \in \mathbb{R}^n$. Note that the penalty term then reads

$$\|f\|_{\mathcal{H}}^2 = \sum_{i,j=1}^n t_i t_j \mathcal{K}(x_i, x_j) = t^\top K t,$$

where $K \in \mathbb{R}^{n \times n}$ is the matrix with entries $K_{ij} = \mathcal{K}(x_i, x_j)$. The minimization problem can then be seen as some generalized ridge penalization:

$$t^\star \in \operatorname*{argmin}_{t\in\mathbb{R}^n} \left\{ \frac{1}{n}\sum_{i=1}^{n} \ell\left(y_i, (Kt)_i\right) + \lambda t^\top K t \right\},$$

where we used the equality

$$(Kt)_i = \sum_{j=1}^{n} t_j \mathcal{K}(x_i, x_j).$$

Predictions are finally performed with the optimal set of coefficients $t^\star$ as

$$f^\star(x') = \sum_{i=1}^{n} t_i^\star \mathcal{K}\left(x', x_i\right).$$

7

Trees and ensemble methods

We discuss in this chapter how to perform predictions (either classification or regression) with trees. We start by presenting regression and classification trees in Section 7.1, and next discuss how to improve the performance of possibly weak learners such as trees by combining prediction functions, either by some form of averaging (see Section 7.2) or by some greedy reweighting procedure known as boosting (see Section 7.3).

The presentation of trees is based on [41, Chapter 18], [25, Chapter 9] (in particular Section 9.2), [8, Section 14.4] and [50, Chapter 18]; while the elements on ensemble methods and boosting are taken from [41, Chapter 18], [4, Chapter 10], [25, Chapter 11], [8, Sections 14.2 and 14.3] and [40, Chapter 7].

7.1 Regression and classifications trees

We show in this section how to perform predictions with decision trees. We start by making precise the principle of the method in Section 7.1.1, and then specify the strategy to regression trees in Section 7.1.2 and next to classification trees in Section 7.1.3. We conclude by discussing a few practical points of the method in Section 7.1.4.

We think of inputs having ordered values (real variables, or, in the case of discrete inputs, values which can be compared such as shoe sizes, ages, etc); see Section 7.1.4 for a discussion on genuine categorical data.

7.1.1 Principle of the method

The idea behind decision trees is to recursively partition the input space $\mathcal{X}$ and define a local model in each resulting region. Each region is associated with a leaf of the tree. The partitioning

is done with a series of questions to hierarchically decompose the input space, and group samples with the same labels (for classification) or with similar target values (for regression).

We consider recursive binary trees to simplify the presentation, and also to avoid data fragmentation (if there are too many possible choices at each step, the end nodes/leaves can be associated with too small a number of instances). The construction is performed using axis parallel splits, *i.e.* separations are done based on values of certain features only, and not linear combinations of features (which can make it difficult to learn certain separations: think of diagonally separated data in a feature space of dimension 2...). Restricting to axis parallel splits is better for the computational scalability of the method, allows for an easy randomization of the algorithm, and also makes the interpretation easier.

The construction can be formalized by considering a set of nested decision rules. At each node j of the decision tree, the feature x_{k_j} of an input x is compared to a threshold value t_j, *i.e.* one checks whether $x_{k_j} \leqslant t_j$ or $x_{k_j} > t_j$. Outputs are specified at leaves. For the example of Figure 7.1, the outputs are specified in the regions

$$R_1 = \{x_1 \leqslant t_1,\ x_2 \leqslant t_2\}, \qquad R_2 = \{x_1 > t_1,\ x_2 \leqslant t_4\}, \qquad \dots$$

The right part of Figure 7.1 illustrates a key advantage of the method, namely its interpretability.

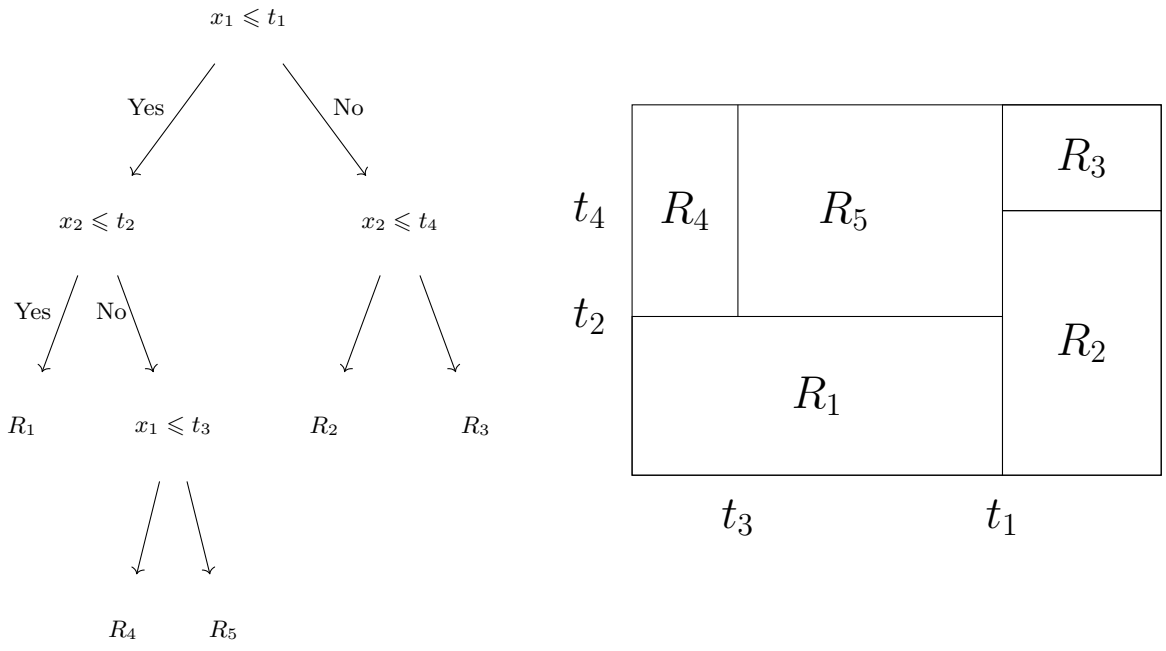

Fig. 7.1: Left: Binary tree. Right: Associated regions.

7.1.2 Regression trees

We consider regression for inputs $\mathcal{X} = \mathbb{R}^d$ and outputs $\mathcal{Y} = \mathbb{R}$ for simplicity. Regression trees implement prediction functions of the form

$$f_\theta(x') = \sum_{j=1}^{J} w_j \mathbf{1}_{x' \in R_j},$$

when there are J regions $R_1, \dots, R_J$ (corresponding to J leaves in the decision tree). The parameter θ gathers the regions and the values $w_j \in \mathbb{R}$ predicted in these regions:

$$\theta = \{R_j, w_j\}_{1 \leqslant j \leqslant J}.$$

For (unregularized) least square regression, the values $(w_j)_{1 \leqslant j \leqslant J}$ are in fact easy to obtain once the regions $R_1, \dots, R_J$ are determined. This is made precise in the next exercise.

Exercise 7.1. *For least square regression, prove that*

$$w_j = \frac{\sum_{i=1}^{n} y_i \mathbf{1}_{x_i \in R_j}}{\sum_{i=1}^{n} \mathbf{1}_{x_i \in R_j}}$$

is the empirical average of the outputs associated with inputs belonging to R_j.

The previous exercise shows that the key point to construct regression trees is to find the partition $R_1, \dots, R_J$. In general, finding the optimal partition is infeasible. In practice, one resorts to a greedy strategy where one node is grown at the time. Popular implementations of this idea are CART, ID3 and C4.5 for instance. The tree is then pruned in order to avoid overfitting.

Growing the tree. Let us explain how to grow a tree when being at node j. We introduce to this end the set $\mathcal{D}_j$ of data points (x_i, y_i) which are still considered at this node, and denote by I_j the associated indices. We then consider the following possible partitions of this set, where we separate points based on the value of the k-th feature of the input:

$$\mathcal{D}_j^-(k,t) = \{(x_i, y_i) \in \mathcal{D}_j : x_{i,k} \leqslant t\}, \qquad \mathcal{D}_j^+(k,t) = \{(x_i, y_i) \in \mathcal{D}_j : x_{i,k} > t\}.$$

We choose k and t so that the following loss function is maximally decreased when partitioning $\mathcal{D}_j$:

$$\begin{aligned}(k_j, t_j) &\in \operatorname*{argmin}_{\substack{1 \leqslant k \leqslant d \\ t \in \mathbb{R}}} \left\{ \min_{w_- \in \mathbb{R}} \sum_{(x_i,y_i) \in \mathcal{D}_j^-(k,t)} (y_i - w_-)^2 + \min_{w_+ \in \mathbb{R}} \sum_{(x_i,y_i) \in \mathcal{D}_j^+(k,t)} (y_i - w_+)^2 \right\} \\ &= \operatorname*{argmin}_{\substack{1 \leqslant k \leqslant d \\ t \in \mathbb{R}}} \left\{ \sum_{(x_i,y_i) \in \mathcal{D}_j^-(k,t)} \left(y_i - w_{j,k,t}^-\right)^2 + \sum_{(x_i,y_i) \in \mathcal{D}_j^+(k,t)} \left(y_i - w_{j,k,t}^+\right)^2 \right\},\end{aligned}$$

where $w_{j,k,t}^{\pm}$ are the empirical averages of the outputs on the sets $\mathcal{D}_j^{\pm}(k,t)$:

$$w_{j,k,t}^{\pm} = \frac{1}{|\mathcal{D}_j^{\pm}(k,t)|} \sum_{(x_i,y_i) \in \mathcal{D}_j^{\pm}(k,t)} y_i.$$

The optimal values for k and t can be determined with a computational cost $\mathrm{O}(dn_j \log n_j)$ when $\mathcal{X} = \mathbb{R}^d$ and there are $n_j = |I_j|$ data points to consider (see [50, Section 18.2.3]). A naive implementation can be performed with a cost $\mathrm{O}(dn_j^2)$. The factor d comes from the fact that one needs to consider all the possible values $1 \leqslant k \leqslant d$. For a given value of k, the optimal treshold could be found by sorting the values $x_{1,k}, \dots, x_{n,k}$ for the n_j data points in $\mathcal{D}_j$ (cost $\mathrm{O}(n_j \log n_j)$) and then looping over the n_j possibles values $(x_{i,k})_{i \in I_j}$ for t. Note indeed that it suffices to consider values of t coinciding with the value of a component of a data point. For each value of t, the loss function to evaluate can be evaluated with a computational cost $\mathrm{O}(n_j)$.

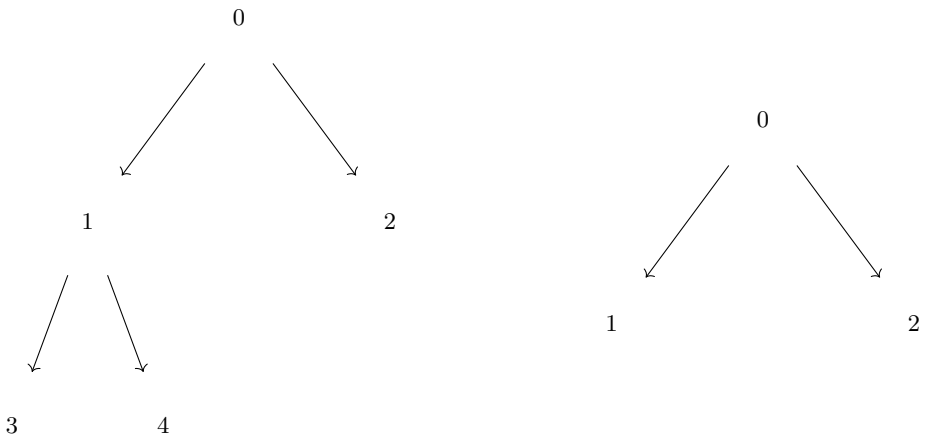

Fig. 7.2: Left: Reference tree T_0. Right: Pruned tree where node 1 has been collapsed.

Pruning the tree. Trees are grown until some minimal node size is reached (*e.g.* 5 data points) or the maximal depth is attained. It is good practice then to prune the tree back in order to avoid overfitting. Note indeed that large enough trees can always perfectly fit the data, but they are then too tightly adjusted to the data at hand and suffer from poor generalization capabilities.

Complexity pruning is one way of proceeding (see [25, Section 9.2.2]). To present the method, denote by T_0 the tree to be pruned. A subtree T of T_0 is any tree that can be obtained by collapsing a certain number of internal (*i.e.* non-terminal) nodes; see Figure 7.2 for an illustration. The metric to find the right tree size is

$$C_\alpha(T) = \sum_{t=1}^{|T|} N_t(T) Q_t(T) + \alpha |T|,$$

where $\alpha \geqslant 0$ is some regularization parameter, $|T|$ is the number of terminal nodes of the tree T, $N_t(T)$ is the number of data inputs at node t, and $Q_t(T)$ is the empirical variance at this node:

$$Q_t(T) = \frac{1}{N_t(T)} \sum_{i : x_i \in R_t} (y_i - w_t)^2, \qquad w_t = \frac{1}{N_t(T)} \sum_{i : x_i \in R_t} y_i.$$

For a given value of $\alpha \geqslant 0$ one then looks for the subtree T_α of T_0 which minimizes $C_\alpha(T)$. Large values of α favor smaller trees, while smaller values of α allow for larger trees. The initial tree is recovered for $\alpha = 0$. To find T_α in practice, one resorts to weakest-link pruning, which consists in collapsing the node that produces the smallest per-node increase in $C_0(T)$; and continues until the single node (root) tree is produced. This produces a finite sequence of subtrees which contains T_α (see [1, Section 1.10.9] for further details).

Let us finally conclude by saying that pruning is not always performed in practice. It makes sense only if one wants a single nice decision tree. In practice, as discussed in Sections 7.2 and 7.3 in particular, one often relies on ensembles of rather small trees, which are individually not providing great predictions (and therefore need not be pruned).

7.1.3 Classification trees

We discuss here how to perform classification with trees, for a label set $\mathcal{Y} = \{1, \dots, K\}$. We rely on the empirical probabilities of $k \in \{1, \dots, K\}$ in $\mathcal{D}$:

$$\widehat{p}_k(\mathcal{D}) = \frac{1}{|\mathcal{D}|} \sum_{(x_i, y_i) \in \mathcal{D}} \mathbf{1}_{y_i = k}.$$

Using the same notation as in Section 7.1.2, the prediction at the jth terminal node is $\operatorname{argmax}_{1\leqslant k\leqslant K} \widehat{p}_k(\mathcal{D}_j)$ in the region R_j.

As for regression trees, splitting is based on the values of the features of data points; but one needs another measure in the greedy approach to select the feature and treshold. The general form of this selection procedure is the following, at node j:

$$(k_j, t_j) \in \operatorname*{argmin}_{\substack{1\leqslant k\leqslant d \\ t\in\mathbb{R}}} \left\{ \frac{|\mathcal{D}_j^-(k,t)|}{|\mathcal{D}_j|} \mathscr{C}\left(\mathcal{D}_j^-(k,t)\right) + \frac{|\mathcal{D}_j^+(k,t)|}{|\mathcal{D}_j|} \mathscr{C}\left(\mathcal{D}_j^+(k,t)\right) \right\},$$

where $\mathscr{C}(\mathcal{D})$ is some cost function taking as an input a set of data points $\mathcal{D}$. Various choices can be considered, which all measure the purity of the labels. More precisely, one can consider

- the misclassification error $1 - \max_{1\leqslant k\leqslant K} \widehat{p}_k(\mathcal{D})$. This expression is motivated by the fact that, for a terminal node, one would predict the argument of the max;
- the Gini index $\sum_{k=1}^{K} \widehat{p}_k(\mathcal{D})\,(1-\widehat{p}_k(\mathcal{D})) = 1 - \sum_{k=1}^{K} \widehat{p}_k(\mathcal{D})^2$. The interpretation of this cost function is that it gives the expected error rate. Indeed, the probability to make a mistake on the label k is the product of observing the label, namely $\widehat{p}_k(\mathcal{D})$, multiplied by the probability to misclassify it, namely $1-\widehat{p}_k(\mathcal{D})$; the total error rate is then the sum over k of the latter quantity;
- the cross entropy or deviance $-\sum_{k=1}^{K} \widehat{p}_k(\mathcal{D}) \log \widehat{p}_k(\mathcal{D})$.

Pruning is traditionally done with the misclassification error; while the splitting is done with the Gini index or cross entropy. All these cost functions favor pure nodes with a single population. This can be easily understood when $K = 2$, as the cost functions are respectively $1-\max(p, 1-p)$, $2p(1-p)$ and $-p\log p - (1-p)\log(1-p)$.

Exercise 7.2. *Consider the set*

$$\mathscr{P} = \left\{ (p_1, \dots, p_K) \in [0,1]^K \;\middle|\; \sum_{k=1}^{K} p_k = 1 \right\}.$$

Show that the mappings

$$\mathscr{P} \ni (p_1, \dots, p_K) \mapsto \sum_{k=1}^{K} p_k(1-p_k), \qquad \mathscr{P} \ni (p_1, \dots, p_K) \mapsto -\sum_{k=1}^{K} p_k \log p_k,$$

have nonnegative values, and that $p^{\star,\ell} = (\mathbf{1}_{k=\ell})_{1\leqslant k\leqslant K}$ are the only minimizers of these functions.

7.1.4 Some practical points

We discuss in this section various practical points.

Categorical values. In some cases, predictions (either regression or classification) are performed with inputs which cannot be ordered or compared, for instance colors of flowers. There are $2^{K-1}-1$ possible partitions of the K labels in two groups (see Exercise 7.3), which is too large a number to be tackled unless K is very small. For regression, an option is to order categories by increasing mean of the output and split them as if they were an ordered predictor. This can also be done for binary classification, but is more complicated for multiclass classification, see the discussion in [25, Section 9.2.4].

Exercise 7.3. *Show by induction that there are $2^{K-1}-1$ possible partitions of the K labels in two groups.*

Regularization. As mentioned above, the training error of trees can be brought down to 0 by considering trees deep enough, in the absence of label noise. A heuristic approach to prevent this is to fix the maximal depth of the tree, and/or the minimal number of data points at the leaves. Another option is to perform backpruning, as discussed in Section 7.1.2.

Variance of the predictors. Trees can have a very low bias as they are flexible enough to accomodate any training set and loss function provided the tree is deep enough. This suggests that the associated predictors will have a large variance, *i.e.* will be unstable with respect to changes in the training data points. This is due in particular to their hierarchical structure, since a modification at a node in the tree during the learning process impacts all the leaves attached to this node (think for example of an error or a difference at the root node, which will lead to two completely different trees a priori). The high variance of predictors based on decision trees motivates considering ensemble methods; see Section 7.2.

Choice of loss function. As for other models, it is possible to consider asymmetric loss functions or splitting criteria, in order to put more emphasis on the correct classification of certain labels, or regression of certain values (think of cancer prediction or spam classification for instance).

Final discussion: advantages and disadvantages of decision trees. We conclude this section by listing the main advantages and disadvantages of trees for classification and regression. Advantages include

- the method is easily interpretable, as the data is broken up directly based on the features. This is probably the main interest of the approach;
- it can handle data of mixed types, with discrete, continuous and categorical inputs;
- there is no need to standardize the data as the method is scale invariant by construction;
- it automatically performs feature selection;
- it is relatively robust to outliers, as the tresholds can be considered as some forms of medians or quantiles (although the predicted values are still based on averages, for regression);
- the method is fast to fit and scales well to large data sets.

All these characteristics make trees an interesting option for data mining. Trees however has some disadvantages, in particular:

- the predictions are usually not very accurate compared to other models, due to the greedy approach used in the fit;
- the variance of the predictions is high due to some instability with respect to changes in the data inputs;
- the predictions are not smooth for regression, as the predictors are piecewise constant over regions of parameter space;
- there is some limitation in expressivity due to the axis splits (think of 2-dimensional data that would need to be separated along the diagonal: this requires rather deep trees).

Exercise 7.4 (Binary classification with ensembles of trees).

(a) Consider the three trees depicted in Figure 7.3. The output of each tree is denoted by $f_i(x) \in \{0,1\}$*. The trees have respective weights* $\alpha_i \in [0,1]$ *for* $1 \leqslant i \leqslant 3$*, with* $\alpha_1 + \alpha_2 + \alpha_3 = 1$*. Plot the values of* $\alpha_1 f_1(x) + \alpha_2 f_2(x) + \alpha_3 f_3(x)$*.*

(b) Consider the function represented in Figure 7.4, which is a piecewise constant function with values in $[0,1]$*. The numbers* $\alpha_1, \alpha_2 \in [0,1]$ *are such that* $\alpha_1 + \alpha_2 = 1$*. How can this function be obtained from an ensemble of binary trees? (recall that a binary tree has values in* $\{0,1\}$*)*

7.2 Bagging and averaging techniques

The idea behind ensemble methods is to combine the results of predictors trained on different datasets, "as independent as possible", in order to obtain a better predictor – or at least a predictor

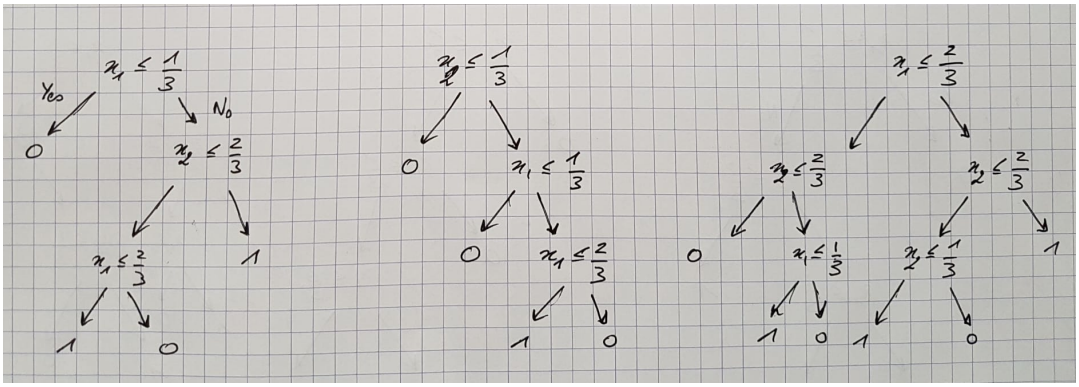

Fig. 7.3: Ensemble of classification trees.

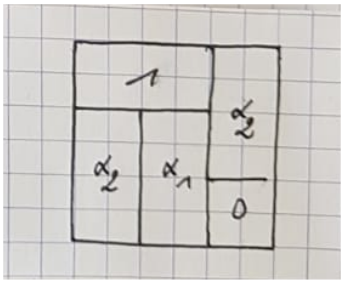

Fig. 7.4: Prediction function with piecewise constant values.

of the same quality as with other techniques, but in reduced wall clock time (as computations on many simple predictors can be parallelized). The prediction obtained from ensemble methods is either an actual average for regression, or a majority vote for classification. Let us emphasize that, although we present this approach in this chapter devoted to trees, it is in fact a general philosophy which can be used in many contexts.

Ensemble methods make sense for local averaging methods such as K nearest neighbors, and for nonlinear models obtained from empirical risk minimization. They do not make sense on the other hand for affine models obtained from empirical risk minimization as the average of affine models is still an affine model. Our presentation is based on regression in a supervised context, but can be extended to classification when considering the Φ-risk associated with convex surrogates.

7.2.1 Motivation: the ideal setting

If M independent and identically distributed data sets $\mathcal{D}_1, \ldots, \mathcal{D}_M$ are available (which is of course often too strong an assumption...), one can construct a predictor $\widehat{f}$ by averaging the predictors $\widehat{f}_{\theta_m}$ trained on each data set $\mathcal{D}_m$:

$$\widehat{f}(x') = \frac{1}{M} \sum_{m=1}^{M} \widehat{f}_{\theta_m}(x').$$

Note that the parameters θ_m are independent and identically distributed. Therefore, $\widehat{f}$ has the same bias as the underlying base predictors, but a variance of order $\mathrm{O}(1/M)$ (more precisely, the variance of the base predictors divided by M).

For example, for K nearest neighbors, the upper bound (1.17) obtained for a single predictor should be modified as

$$0 \leqslant \mathbb{E}_{\mathcal{D}} \left[\mathcal{R}(\widehat{f}) \right] - \mathcal{R}^\star \leqslant \frac{\sigma^2}{KM} + 8B^2 \operatorname{diam}(\mathcal{X})^2 \left(\frac{2K}{n} \right)^{2/d},$$

i.e. the variance term has been divided by M. The optimal value of K then scales as $M^{-d/(d+2)}n^{2/(d+2)}$, and the optimal excess risk as $(Mn)^{-2/(d+2)}$. This bound is similar to the one obtained for single data set of size Mn. The main interest of averaging in this context is that the evaluation of the averaged predictor can be less expensive, or can be parallelized.

7.2.2 Bagging

The situation considered in Section 7.2.1 is an idealized one as one usually does not have independent data sets. This motivates artificially creating data sets with some randomness by replicating them from a baseline data set. This is precisely the idea behind bootstraping, which motivates the terminology "bagging" (bootstrap aggregating). More precisely, given a data set $\mathcal{D} = \{(x_i, y_i)_{1 \leqslant i \leqslant n}\}$, one constructs M data sets by sampling with replacement from $\mathcal{D}$. Of course, there is a non zero probability that two data points are identical in the data set. The average proportion of different data points can in fact be made precise, as the following exercise shows.

Exercise 7.5. *What is the number of different data points seen in average in each of these M datasets, in the limit $n \to +\infty$?*

Note that "out-of-bag" instances (*i.e.* data points which do not show up in the random data set which is sampled) can be used for validation. One interest of bagging is that it prevents algorithms to focus too much on some data points. This gives more robustness to the learning procedure and ensures a better generalization. The method works better for unstable estimators, with high variance; see for instance the mathematical analysis for 1-nearest neighbors presented in [4, Section 10.1.2].

7.2.3 Random forests

Random forests mix bagging, as an ensemble of decision trees is learned on bootstrapped samples of data, and random projections, since at each node of the trees splits are based only on a random subset of all features. The motivation for the latter extra source of randomness is to decorrelate the predictors as much as possible in order to benefit from the decrease in variance offered by ensemble methods. We refer to [7, 38] for a more detailed presentation and elements on the mathematical analysis of the approach.

7.3 Boosting

We present the approach for real-valued outputs. This is natural for regression; for classification it means that one works with convex surrogates to work out classification rules from real-valued outputs (see Section 3.1). The bottom line of the method is to sequentially fit an additive model as

$$f_T(x) = \sum_{t=1}^{T} \alpha_t F_t(x), \tag{7.1}$$

with possibly some reweighting of the data in some cases. We first present the general philosophy of the method in Section 7.3.1, in particular the celebrated AdaBoost method and some of its variations; and then discuss gradient boosting in Section 7.3.2.

7.3.1 General philosophy

Boosting is a greedy procedure where the predictor (7.1) is updated by looking for an extra term which maximally minimizes the empirical risk. More precisely, at iteration $t \geqslant 1$, one first solves the minimization problem

$$(\alpha_t, \theta_t) = \operatorname*{argmin}_{\substack{\alpha\in\mathbb{R}\\ \theta\in\Theta}} \left\{ \frac{1}{n}\sum_{i=1}^{n} \ell\Big(y_i, f_{t-1}(x_i) + \alpha F(x_i;\theta)\Big) \right\}, \tag{7.2}$$

and then sets

$$f_t = f_{t-1} + \alpha_t F(\cdot, \theta_t).$$

The procedure is initialized in some way, typically by setting $f_0 = 0$. A regularization term could be added to the empirical risk to minimize, but we do not introduce such a component here for simplicity of exposition. The details on how the minimization in (7.2) is exactly done depends on the choice of loss function, which is why we next consider two particular cases. Gradient boosting, presented in Section 7.3.2, allows for a more general approach which does not rely on the form of the loss function.

Least-square boosting. We consider the loss function $\ell(y,z) = |y-z|^2$. Introduce the residual $r_{i,t}$ at step t for the prediction associated with x_i, namely

$$r_{i,t} = y_i - f_{t-1}(x_i).$$

In this context, the minimization problem (7.2) can be reformulated as

$$(\alpha_t, \theta_t) = \operatorname*{argmin}_{\substack{\alpha\in\mathbb{R}\\ \theta\in\Theta}} \left\{ \frac{1}{n}\sum_{i=1}^{n} |r_{i,t} - \alpha F(x_i;\theta)|^2 \right\},$$

which is the usual least-square fit of $\{(r_{i,t})_{1\leqslant i\leqslant n}\}$ by $F(x_i,\theta)$ (when integrating the constant $\alpha = 1$ into the definition of F). See for instance [41, Figure 18.6] for an illustration.

AdaBoost. This the historical example of boosting, used for binary classification on $\mathcal{Y} = \{-1, 1\}$. The base classifier functions $F(\cdot;\theta)$ have values in $\mathcal{Y}$. One works here with the convex surrogate loss function $\Phi(u) = \mathrm{e}^{-u}$, the associated elementary loss being $\ell(y,z) = \mathrm{e}^{-yz}$. Note in particular that $\mathbf{1}_{y\neq z} = \mathbf{1}_{yz\leqslant 0} \leqslant \ell(y,z)$, in accordance with the discussion in Section 3.1.1.2. In this context, the minimization problem (7.2) can then be reformulated as

$$(\alpha_t, \theta_t) = \operatorname*{argmin}_{\substack{\alpha\in\mathbb{R}\\ \theta\in\Theta}} \left\{ \frac{1}{n}\sum_{i=1}^{n} \omega_{i,t}\mathrm{e}^{-\alpha y_i F(x_i;\theta)} \right\}, \qquad \omega_{i,t} = \mathrm{e}^{-y_i f_{t-1}(x_i)},$$

the associated classifier being

$$\operatorname{sign}(f_T) = \operatorname{sign}\left(\sum_{t=1}^{T} \alpha_t F(\cdot, \theta_t)\right).$$

In fact, the function to minimize can be rewritten as

$$\begin{aligned}\sum_{i=1}^{n} \omega_{i,t}\mathrm{e}^{-\alpha y_i F(x_i;\theta)} &= \left(\sum_{i:F(x_i;\theta)=y_i} \omega_{i,t}\right)\mathrm{e}^{-\alpha} + \left(\sum_{i:F(x_i;\theta)\neq y_i} \omega_{i,t}\right)\mathrm{e}^{\alpha} \\ &= \left(\mathrm{e}^{\alpha} - \mathrm{e}^{-\alpha}\right)\sum_{i=1}^{n} \omega_{i,t}\mathbf{1}_{F(x_i;\theta)\neq y_i} + \mathrm{e}^{-\alpha}\sum_{i=1}^{n}\omega_{i,t}.\end{aligned}$$

The latter formulation makes it clear that the optimization over θ can be performed independently of α, and corresponds to minimizing some training error for the 0-1 loss and a data set with weights. In fact, we will see below that α_t is positive, so that, ideally, one would like to consider

$$\theta_t \in \operatorname*{argmin}_{\theta\in\Theta} \left\{ \sum_{i=1}^{n} \omega_{i,t}\mathbf{1}_{F(x_i;\theta)\neq y_i} \right\}. \tag{7.3}$$

As explained in Section 3.1, the latter optimization problem is difficult to solve. It is also not necessary to solve it too precisely in the context of boosting, which relies on improving weak classifiers such as trees of limited depth (assuming that such weak classifiers can indeed be obtained for the problem at hand).

Once θ_t has been found, one can minimize over α, as the following exercise shows.

Exercise 7.6. *Find the minimizer of*

$$\alpha_t \in \operatorname*{argmin}_{\alpha\in\mathbb{R}} \left\{ \frac{1}{n}\sum_{i=1}^{n} \omega_{i,t} \mathrm{e}^{-\alpha y_i F(x_i;\theta_t)} \right\}.$$

It is useful to introduce the classification error on the weighted data set:

$$E_t = \frac{\displaystyle\sum_{i=1}^{n} \omega_{i,t} \mathbf{1}_{F(x_i;\theta_t)\neq y_i}}{\displaystyle\sum_{i=1}^{n} \omega_{i,t}} \in [0,1]. \tag{7.4}$$

Check that $\alpha_t > 0$ when $E_t \in (0, 1/2)$ (which corresponds to a better prediction than guessing at random).

Overall, starting from $\omega_{i,1} = 1$ and $f_0 = 0$, the procedure therefore amounts to iterating the following steps for $1 \leqslant t \leqslant T$:

- find an approximate solution θ_t to (7.3) (such that $0 \leqslant E_t < 1/2$ with E_t defined in (7.4));
- choose the value α_t given in Exercise 7.6;
- update the classifier as $f_t = f_{t-1} + \alpha_t F(\cdot;\theta_t)$;
- update the weights as

$$\omega_{i,t+1} = \begin{vmatrix} \omega_{i,t}\, \mathrm{e}^{2\alpha_t} & \text{if } y_i \neq F(x_i;\theta_t), \\ \omega_{i,t} & \text{if } y_i = F(x_i;\theta_t). \end{vmatrix}$$

Note that the latter update is equivalent (up to an unimportant multiplicative constant) to setting $\omega_{i,t}\, \mathrm{e}^{\alpha_t}$ if $y_i \neq F(x_i;\theta_t)$ and $\omega_{i,t}\, \mathrm{e}^{-\alpha_t}$ if $y_i = F(x_i;\theta_t)$.

In this procedure, misclassified data points get larger weights, and will therefore count more in the next iteration. However, $(1 - E_t)\mathrm{e}^{-\alpha_t} = E_t \mathrm{e}^{\alpha_t}$, so the mass of the well classified and misclassified points are the same, but there are less and less misclassified points as their weight increases.

Let us conclude this section by a result showing that the training error decreases exponentially fast with the number of rounds of boosting.

Theorem 7.1. *The training error is upper bounded as*

$$\widehat{\mathcal{R}}(f_T) = \frac{1}{n}\sum_{i=1}^{n} \mathbf{1}_{y_i f_T(x_i)\leqslant 0} \leqslant \exp\left(-2 \sum_{t=1}^{T} \left[\frac{1}{2} - E_t \right]^2 \right).$$

In particular, if there exists $\gamma \in [0, 1/2]$ such that

$$\forall t \in \{1, \ldots, T\}, \qquad E_t \leqslant \frac{1}{2} - \gamma, \tag{7.5}$$

then $\widehat{\mathcal{R}}(f_T) \leqslant \mathrm{e}^{-2\gamma^2 T}$.

The condition (7.5) means that the approximate solution to (7.3) is a weak classifier, slightly better than guessing at random. The result supports the claim that the optimization over θ in (7.3) need not be performed too precisely.

Exercise 7.7. *The aim of this exercise is to prove Theorem 7.1.*

(a) Show that one can update the weights as

$$\omega_{i,t+1} = \omega_{i,t}\,\frac{\mathrm{e}^{-\alpha_t y_i F(x_i;\theta_t)}}{Z_t}, \qquad Z_t = \sum_{j=1}^{n} \omega_{j,t}\,\mathrm{e}^{-\alpha_t y_j F(x_j;\theta_t)}.$$

and start from a specific initial value for weights in order to ensure that

$$\forall t \in \mathbb{N}, \qquad \sum_{i=1}^{n} \omega_{i,t} = 1.$$

(b) Prove that $\dfrac{1}{n}\mathrm{e}^{-y_i f_T(x_i)} = Z_T \dots Z_1\, \omega_{i,T+1}$.
(c) Use the inequality $\mathbf{1}_{u\leqslant 0} \leqslant \mathrm{e}^{-u}$ *for all* $u \in \mathbb{R}$ *to obtain* $\widehat{\mathcal{R}}(f_T) \leqslant Z_1 \dots Z_T$.
(d) Show that $Z_t = 2\sqrt{E_t(1-E_t)}$.
(e) Prove that $2\sqrt{u(1-u)} \leqslant \exp\left(-2\left(\dfrac{1}{2}-u\right)^2\right)$ *for* $0 \leqslant u \leqslant 1$ *and conclude.*

Exercise 7.8 (Boosting with a general convex surrogate). *One issue with the exponential loss used in AdaBoost is that it can put too much weight on missclassified points, which makes the method sensitive to outliers or mislabeled examples. One can generalize the boosting procedure to other convex surrogates* $\Phi \in C^1$ *instead of the exponential loss* $\Phi(u) = \mathrm{e}^{-u}$*, such as the logistic loss* $\Phi(u) = \log(1+\mathrm{e}^{-u})$*. For a data set* $\{(x_i, y_i),\ 1 \leqslant i \leqslant n\}$ *with labels* $y_i \in \mathcal{Y} = \{-1, 1\}$*, and for a given class of functions* $F(\cdot, \theta)$ *with values in* $\mathcal{Y}$*, parametrized by* $\theta \in \Theta$*, the boosting algorithm proceeds by first solving*

$$(\alpha_t, \theta_t) \in \operatorname*{argmin}_{(\alpha,\theta)\in\mathbb{R}\times\Theta} \left\{ \frac{1}{n}\sum_{i=1}^{n} \Phi\left(y_i[f_{t-1}(x_i) + \alpha F(x_i,\theta)]\right) \right\}, \tag{7.6}$$

and then updating the classifier as $f_t = f_{t-1} + \alpha_t F(\cdot, \theta_t)$.

(a) For a given $\theta \in \Theta$*, write out the optimization problem on* $\alpha \in \mathbb{R}$ *as*

$$\min_{\alpha\in\mathbb{R}} \left\{ \sum_{i\in S_\theta^=} \Phi(z_{i,t} + \alpha) + \sum_{i\in S_\theta^{\neq}} \Phi(z_{i,t} - \alpha) \right\},$$

for sets $S_\theta^=, S_\theta^{\neq}$ *and elements* $z_{i,t} \in \mathbb{R}$ *to make precise.*
(b) Write out the necessary condition satisfied by the minimizer $\alpha_\star(\theta)$ *of the previous minimization problem (assuming that this minimizer indeed exists).*
(c) Give the expression of $\alpha_\star(\theta)$ *for the exponential loss* $\Phi(u) = \mathrm{e}^{-u}$*. How does the value compare to the one considered for AdaBoost?*
(d) Give the expression of $\alpha_\star(\theta)$ *for the square loss* $\Phi(u) = (1-u)^2$.
(e) Write out the equation satisfied by $\alpha_\star(\theta)$ *for the logistic loss* $\Phi(u) = \log(1+\mathrm{e}^{-u})$.

This exercise shows how to solve for the value of α *for* θ *fixed in* (7.6)*. This allows, in fortunate situations, to formulate a minimization over* θ *only. The latter optimization problem may however be quite difficult – for the "historic" AdaBoost case, there are fortunate algebraic simplifications which make the problem tractable. There are various alternative options: rely on coordinate descent methods (see [40, Section 7.2.2]); consider only approximate minimizers* θ*, as long as they give rise to weak learners; or turn to gradient boosting (see Section 7.3.2).*

7.3.2 Gradient boosting

One limitation of AdaBoost and its variants is that the problem (7.2) has to be solved, with a procedure depending on the elementary loss function ℓ at hand. Gradient boosting is a more generic procedure. The main idea behind it is to consider the addition of a new term $\alpha F(\cdot;\theta)$ to the sum $\alpha_1 F(\cdot;\theta_1) + \cdots + \alpha_{t-1} F(\cdot;\theta_{t-1})$ as a perturbation. Therefore, the function to minimize in (7.2) can be approximated by

$$\frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f_{t-1}(x_i)) + \frac{\alpha}{n}\sum_{i=1}^{n} \partial_z \ell(y_i, f_{t-1}(x_i)) F(x_i;\theta),$$

the first term not depending on θ. In order to have some maximal decrease of the above approximate functional, it suggests that the vector $(F(x_1;\theta),\dots,F(x_n;\theta))$ should be proportional to the vector $(\partial_z \ell(y_1, f_{t-1}(x_1)),\dots,\partial_z \ell(y_n, f_{t-1}(x_n)))$. There are two ways to making this idea practical:

- minimizing over $\theta \in \Theta$ the second part of the above approximate functional, for α fixed, namely

$$\theta_t \in \operatorname*{argmin}_{\theta\in\Theta} \left\{ \frac{1}{n}\sum_{i=1}^{n} \partial_z \ell(y_i, f_{t-1}(x_i)) F(x_i;\theta) \right\}.$$

  Once θ_t is determined, there are various choices for the update of the predictor, including the magnitude α_t for the new term, and some convex combination with previous iterates using some mixing parameter; see the discussion in [4, Section 10.2.2] for further precisions.
- fitting $F(\cdot;\theta)$ in order for $(F(x_1;\theta),\dots,F(x_n;\theta))$ to be the best possible approximation of $-(\partial_z \ell(y_1, f_{t-1}(x_1)),\dots,\partial_z \ell(y_n, f_{t-1}(x_n)))$. This can be obtained through some least square regression:

$$\min_{\theta\in\Theta} \left\{ \sum_{i=1}^{n} |F(x_i;\theta) + \partial_z \ell(y_i, f_{t-1}(x_i))|^2 \right\}.$$

  One can in addition find the magnitude α_t by some line search, as in usual gradient methods. The so-obtained method is close to least-square boosting when $\ell(y,z) = (y-z)^2$.

When using trees, the new term which is added to the prediction function is taken of the form

$$F(x;\theta_t) = \sum_{j=1}^{J_t} w_{j,t} \mathbf{1}_{x\in R_{j,t}},$$

with regions $R_{j,t}$ which can be learned as for usual decision trees for regression, based on fitting the gradient as discussed above, but with values $w_{j,t}$ which are fit as

$$w_{j,t} \in \operatorname*{argmin}_{w\in\mathbb{R}} \left\{ \sum_{i:x_i\in R_{j,t}} \ell(y_i, f_{t-1}(x_i) + w) \right\}.$$

Indeed, for trees, the difficult part of the learning is to find the regions $R_{j,t}$, but once these regions are given, it is possible to fit the values $w_{j,t}$ somewhat directly in the boosting objective function, instead of its gradient approximation. For the squared error, the value $w_{j,t}$ is just the empirical mean of the residuals $(y_i - f_{t-1}(x_i))_{1\leqslant i\leqslant n}$ in the region (see [41, Section 18.5.5.1] and [25, Section 10.10.2]).

XGBoost. A very efficient and widely used implementation of gradient boosting trees is "extreme gradient boosting" (XGBoost), proposed in [12]. It improves on the above described gradient boosting method by

- adding a regularizer to the tree complexity;
- considering a second order approximation to the functional to minimize in (7.2) instead of a first order one;
- using some random sampling of features as for random forests;
- implementing various computer science tricks to improve the scalability of the method.

See for instance [41, Section 18.5.5.2] for further precisions.

8

Neural networks

We discuss in this chapter nonlinear models based on neural networks, which have nowadays become increasingly important and useful for many machine learning applications. Our focus is on feed-forward neural networks, whose structure is described in Section 8.1, and whose training is made precise in Section 8.2. We will mostly considered supervised learning, both for classification and regression; unsupervised learning based on autoencoders is however considered in Section 8.3. We conclude the chapter with a brief presentation of other types of neural networks in Section 8.4.

The main bibliographical resources for this chapter are [41, Chapters 13 to 15], with an emphasis on Chapter 13 which covers more basic material (see also Section 20.3 for autoencoders); as well as the book by Goodfellow, Bengio and Courville [23] (see in particular Chapters 6, 7, 8 and 11). Concerning the practical implementation, many useful contents can be found in the book [58]. Some (more advanced) elements on the mathematical analysis of neural networks can be read in [4, Chapter 9].

8.1 Feed-forward neural networks

The motivation for considering neural networks is to construct nonlinear mappings to go from the inputs to the outputs, without making explicit feature functions, but by describing them using a very expressive parametrized set of functions. Somehow, neural networks used in supervised learning (either for classification or regression) can be seen as an efficient way of learning a feature function $\phi_p(x)$ (parametrized by a vector p) to be used in standard, possibly linear models, with a prediction function of the form $f_\theta(x) = w^\top \phi_p(x) + b$, with $\theta = (w, b, p)$.

We consider in this section feed-forward neural networks, also known as multilayer perceptrons. Other types of architectures are briefly discussed in Section 8.4.

8.1.1 Structure of multilayer perceptrons

Layers and parameters. The prediction function for feed-forward neural networks is defined in an iterative manner by composing elementary functions as

$$f_\theta(x) = \left(g_{\theta_L} \circ g_{\theta_{L-1}} \circ \cdots \circ g_{\theta_1}\right)(x).$$

This formula corresponds to a network with $L-1$ hidden layers and an output layer, with input layer $x \in \mathbb{R}^d$. Shallow networks have small values of L, while deep neural networks correspond to larger values of L (say, about 20 for current implementations). The input is first (nonlinearly) transformed into an element of $\mathbb{R}^{d_1}$ with $g_{\theta_1} : \mathbb{R}^d \to \mathbb{R}^{d_1}$. Then, each function $g_{\theta_\ell} : \mathbb{R}^{d_{\ell-1}} \to \mathbb{R}^{d_\ell}$ modifies the output of the previous layer, until the last function $g_{\theta_L} : \mathbb{R}^{d_{L-1}} \to \mathbb{R}^{d_L}$ maps it to some output vector. The dimension of the output is $d_L = 1$ for simple regression problems, but it can be a vector in many situations, for instance when performing regression on vector labels, or when performing multiclass classification. In the latter case, d_L is typically the number of classes to be predicted, as the output is passed through a softmax function, following the same strategy as in Section 3.2.4.

The transformation functions for each layer map an input $a_{\ell-1} \in \mathbb{R}^{d_{\ell-1}}$ obtained from the previous layer to an input $a_\ell \in \mathbb{R}^{d_\ell}$ to be used by the next layer, of the form

$$a_\ell = g_{\theta_\ell}(a_{\ell-1}) = \rho_\ell\left(W_\ell a_{\ell-1} + b_\ell\right), \qquad W_\ell \in \mathbb{R}^{d_\ell \times d_{\ell-1}}, \quad b_\ell \in \mathbb{R}^{d_\ell},$$

where $\rho_\ell : \mathbb{R} \to \mathbb{R}$ is a so-called activation function, which is applied componentwise. In the sequel, we denote by $z_\ell = W_\ell a_{\ell-1} + b_\ell \in \mathbb{R}^{d_\ell}$ the argument of the activation. Historically, perceptrons were constructed with Heaviside functions for activations, but the associated models are difficult to train. Nowadays activation functions are continuous and often smooth (see below).

Remark 8.1. *In practice, all operations are usually done in a vectorized manner. The input of the neural network is then the design matrix $X \in \mathbb{R}^{n \times d}$ whose i-th line is the line vector $x_i \in \mathbb{R}^{1 \times d}$; and the output is the vector $y = (y_1, \ldots, y_n)^\top \in \mathbb{R}^{n \times d_L}$. For simplicity of exposition, we however consider that the input is a column vector of size d, and the output is a column vector of size d_L.*

The parameters of the model are $\theta_1 = (W_1, b_1), \theta_2 = (W_2, b_2), \ldots, \theta_L = (W_L, b_L)$, so that $\theta = (\theta_1, \ldots, \theta_L)$. The total number of parameters is therefore

$$\sum_{\ell=1}^{L} d_\ell(1 + d_{\ell-1}),$$

with $d_0 = d$ the dimension of the input, and d_L the dimension of the output. The number of parameters is usually very large, possibly (much) larger than the number of training points; see for instance Exercise 8.1 for a concrete example.

Exercise 8.1. *What is the number of parameters for a neural network classifying the MNIST set of digits (images of 28×28 pixels) based on a single hidden layer? What is this number for a hidden layer of size 50? Comment the magnitude of this number in view of the size of the data set.*

Activation functions. There are various choices for the activation functions $\rho_\ell : \mathbb{R} \to \mathbb{R}$ used to pass from one layer to the other:

- the linear function $\rho_\ell(z) = z$, which can in particular be considered for the final layer, to produce a result such as $w^\top \phi(x) + b$, the first $L-1$ layers being used to come up with a featurization function;
- the sigmoid function $\rho_\ell(z) = \sigma(z) = (1 + \mathrm{e}^{-z})^{-1}$;
- the hyperbolic tangent $\rho_\ell(z) = \tanh(z) = \dfrac{\mathrm{e}^z - \mathrm{e}^{-z}}{\mathrm{e}^z + \mathrm{e}^{-z}}$;
- the rectifier linear unit (ReLU) function $\rho_\ell(z) = \max(0, z)$, or leaky versions such as $\rho_\ell(z) = \max(0, z) + \alpha \min(0, z)$ with $\alpha > 0$ small (the latter option ensuring that ρ_ℓ' does not vanish for $z < 0$, which may cause issue for training, see Section 8.2.2).

The selection of the activation function should be made on the basis of a trial and error process, *i.e.* it can somehow be considered as a hyperparameter to tune using some (cross) validation procedure. However, many choices lead to similar performances. The general trend is that activation functions with gradients that saturate at infinity, such as sigmoid or hyperbolic tangent, are fine for shallow networks; but for deeper networks one should choose activation functions which do not saturate in order for training to be possible. ReLU and its recent variations (for instance smoothed out versions such as GELU (Gaussian Error Linear Unit) or swish) are currently common choices.

The activation function for the last layer (*i.e.* for the output) depends on the task:

- linear functions are considered for regression;
- sigmoid functions are used for binary classification in $\{0,1\}$, and hyperbolic tangent for binary classification in $\{-1,1\}$;
- multiclass classification can be performed with a softmax function, which, given $z_L = (z_{L,1}, \dots, z_{L,d_L})$ returns

$$a_L = \frac{1}{Z_L}\left(\mathrm{e}^{z_{L,1}}, \dots, \mathrm{e}^{z_{L,d_L}}\right), \qquad Z_L = \sum_{j=1}^{d_L} \mathrm{e}^{z_{L,j}}.$$

More advanced choices are discussed in [23, Section 6.2.2.4], for instance predictions based on sampling from a Gaussian distribution with mean $\mu_\theta(x)$ and variance $\sigma_\theta(x)^2$, where $\mu_\theta, \sigma_\theta^2$ are learned by the first $L-1$ layers.

Loss functions. The loss functions to consider for training neural networks are the same as for other machine learning problems, in particular the square loss $(y-\widehat{y})^2$ for regression problems, and, for classification problems, the cross entropy (3.16) in the binary case or (3.17) for multiclass classification.

Architecture design. Neural networks are quite flexible, and a practical question of interest is how to choose the number of units and the way they are connected. For fully connected feed-forward neural networks, this mean choosing the number of hidden layers, and the width of these layers. For a given number of parameters, one can choose for instance between wider and shallower networks or deeper networks with fewer units per layer in order to perform some hierarchical learning. The reader is encouraged to test strategies on the tensorflow playground[1] to get a feeling of various options.

Some historical landmarks. Let us conclude this section with some historical perspective on the development of neural networks. Neural networks have been considered since Rosenblatt's perceptron in the 50s. Many ideas concerning the structure and training of neural networks were already present in the 80s and 90s, but other learning models were more successful at the time. A strong surge of interest arose after 2010 due to breakthrough results in image classification. The progress here is due to two factors:

add references

(i) the increase in computing power, in particular through graphical processing units (GPUs) which can very efficiently handle matrix/vector operations, and also through dedicated hardware such as tensor processing units (TPUs);
(ii) the size of the data sets available for training have substantially increased, which made it possible to train neural networks with more parameters, in particular deep neural networks.

Exercise 8.2 (Architecture of neural networks). *The aim of this exercise is to interpret or explicitly write out the prediction functions provided by neural networks for a given input $x \in \mathbb{R}^d$ (considered as a column vector here).*

(a) Write out the empirical risk obtained for a neural network with two hidden layers and activation function ρ, identity activation function for a one dimensional output and square loss.

[1] See https://playground.tensorflow.org

(b) Consider a neural network with a single hidden layer with sigmoid activation function σ, and identity activation function for the output. Construct a network predicting the same output but using an hyperbolic tangent activation function in the hidden layer (Hint: use an affine transformation of the parameters to relate the sigmoid and the hyperbolic tangent).

(c) Construct a neural network whose output is

$$f_\theta(x) = \sigma\left(b_0 + v_1 \rho\left(w_1 \cdot x + b_1\right) + v_2 \rho\left(w_2 \cdot x + b_2\right)\right) \in [0,1],$$

with ρ some given activation function, and $w_1, w_2 \in \mathbb{R}^d$, $v_1, v_2, b_0, b_1 \in \mathbb{R}$.

8.1.2 Mathematical results on approximation properties

Neural networks have the so-called "universal approximation property", which allows them to approximate arbitrary continuous functions [13], and even any Borel measurable function [27]. The function to represent is however only obtained in the limit of layers of infinite width – in fact, with a single hidden layer when the output is one-dimensional. We present here a simple argument to substantiate these claims, taken from [4, Section 9.3], and, for simplicity of exposition, do not provide convergence rates (which would require extra smoothness on the function to approximate in order to state quantitative results). The proof we write is for ReLU activation functions, the result being however rather easy to extend to any non polynomial activation function [37].

Before writing an approximation argument, let us immediately emphasize that the results mentioned above are representation results, meaning that they state that a given function can be arbitrarily well approximated by neural networks of increasing sizes. This does not mean that, in practice, the optimization algorithms used to train neural networks will be able to find parameters which indeed allow to correctly approximate the function under consideration.

To simplify the presentation, we consider real valued functions over a compact interval. The prediction function associated with the neural network can then be written as

$$f(x) = \sum_{j=1}^{m} \eta_j \rho\left(w_j x + b_j\right), \qquad \eta_j, w_j, b_j \in \mathbb{R}, \tag{8.1}$$

with ρ the activation function, here $\rho(z) = \max(0, z)$. The arguments $w_j x + b_j$ of the activation functions are obtained by an affine transformation of the input, while the numbers η_j allow to linearly combine of the outcomes of the hidden layer to perform the prediction (note that there is no bias for the last layer). We denote by $w = (w_1, \dots, w_m)$, $b = (b_1, \dots, b_m)$ and $\eta = (\eta_1, \dots, \eta_m)$ the parameters of the neural network.

Proposition 8.1. *Consider a continuous function $f : [-R, R] \to \mathbb{R}$. Then there exist a sequence of parameters (w^m, b^m, η^m) such that*

$$\|f - f_m\|_{C^0([-R,R])} \xrightarrow[m \to +\infty]{} 0,$$

where f_m is (8.1) for the parameters (w^m, b^m, η^m).

The above approximation result is based on networks of infinite width. A similar result for deep neural networks (*i.e.* where the depth would grow instead of the width) is currently less clear (see some references in [4, Section 9.7]).

Exercise 8.3. *The aim of this exercise is to prove Proposition 8.1. We proceed in two steps: (i) we first show that any piecewise affine function with $m-2$ kinks in $[-R, R]$ can be represented by a neural network with a single hidden layer of size m; (ii) we next conclude by an approximation argument. We denote by $x_+ = \max(x, 0)$ the ReLU function, and by $-R < X_1 < \cdots < X_{m-2} < R$ the location of the kinks.*

(a) We first consider the case when $f(-R) = 0$. Construct a representation of the function using $m-1$ neurons, by adding terms one after the other on the various subintervals $[X_i, X_{i+1}]$.

(b) Consider next the case $f(-R) \neq 0$. Prove that the function can be represented with one extra neuron compared to the previous case.

(c) Show that any continuous function can be approximated arbitrary closely in the C^0 norm by a piecewise affine function, and conclude.

8.2 Training neural networks

The training of neural networks can be nicely visualized using the playground of tensorflow. This task seems to be a rather difficult one at first sight, as it corresponds to a nonconvex optimization problem involving a very large number of parameters. It can however be made computationally efficient thanks to

- backpropagation and more generally automatic differentiation to compute gradients with respect to parameters and inputs;
- SGD and its variations to leverage minibatching to reduce the cost of one training step;
- good software to make the implementation easy for the user; in particular Tensorflow (Google) and PyTorch (Meta).

We start by discussing how to compute the gradient in Section 8.2.1, then discuss some elements on optimization and regularization which are specific to neural networks (see respectively Sections 8.2.2 and 8.2.3).

8.2.1 Computing the gradient

We discuss in this section how to compute the gradient of functionals of the output of neural networks, using the celebrated backpropagation algorithm. The presentation is based on [43, Chapter 2]. We assume that the activations functions are continuously differentiable for the derivation below.

Network and loss function. We consider a neural network composed of L layers, whose parameters are

$$\theta = \{W_\ell, b_\ell\}_{1 \leqslant \ell \leqslant L}.$$

In accordance with the notation of Section 8.1.1, we denote the weighted input at layer ℓ by z_ℓ and the activation by a_ℓ:

$$a_\ell = \rho_\ell(z_\ell), \qquad z_\ell = W_\ell a_{\ell-1} + b_\ell, \qquad b_\ell \in \mathbb{R}^{d_\ell}, \quad W_\ell \in \mathbb{R}^{d_\ell \times d_{\ell-1}},$$

starting from an input $a_0 = x$. We provide the derivation here for a single input for simplicity. In practice, operations are vectorized when several inputs are considered, so that the neural network takes a matrix as input, and returns a vector.

Let us first make precise the function whose gradient needs to be computed, namely the empirical risk

$$\widehat{\mathcal{R}}_n(\theta) = \frac{1}{n} \sum_{i=1}^{n} \mathscr{L}(y_i, a_{L,i}), \tag{8.2}$$

where $\mathscr{L}$ is some elementary loss function[2] with values in $\mathbb{R}$ (for instance, $\mathscr{L}(y, \widehat{y}) = (y - \widehat{y})^2$ for least square regression), and $a_{L,i}$ is the output of the neural network for the input x_i. It is possible to add a penalization term $\lambda \Omega(\theta)$, but this extra term does not pose any challenge in the evaluation of its gradient, so we do not consider it in this discussion for simplicity of exposition. Also, for simplicity, we write the derivation for a single data point (*i.e.* $n = 1$), and denote by $\widehat{\mathcal{R}}$ the corresponding loss function. It is straightforward to obtain gradients for general values of n from the formulas for $n = 1$.

[2] We do not use here the notation ℓ for the elementary loss function, as in other chapters, in order to avoid confusions with the index of the layer.

Computation of gradients with respect to parameters. We now discuss how to compute the gradient of $\widehat{\mathcal{R}}$ with respect to θ (a similar derivation can be performed to compute the gradient of the output of the neural network with respect to the input x, which is needed for certain applications). This is done by starting from the output, and going back to the input, with the chain rule. More precisely, define

$$\forall j \in \{1, \dots, d_\ell\}, \qquad \delta_j^\ell = \frac{\partial \widehat{\mathcal{R}}}{\partial z_{\ell,j}} \in \mathbb{R},$$

where $\widehat{\mathcal{R}}$ is seen (with some abuse of notation) as a function of z_ℓ. In fact, we write

$$\widehat{\mathcal{R}}(\theta) = R_\ell(z_\ell)$$

in the sequel in order to understand the recursion relation between δ^ℓ and $\delta^{\ell-1}$. Let us start with the output layer, for which

$$\widehat{\mathcal{R}}(\theta) = R_L(z_L) = \mathscr{L}(y, \rho_L(z_L)).$$

Therefore,

$$\delta^L = \rho_L'(z_L)\partial_{\widehat{y}}\mathscr{L}(y, \rho_L(z_L)) \in \mathbb{R}^{d_L}.$$

We next express R_{L-1} in terms of R_L as

$$\widehat{\mathcal{R}}(\theta) = R_{L-1}(z_{L-1}) = R_L\left(W_L\rho_{L-1}(z_{L-1}) + b_L\right),$$

and claim that

$$\delta^{L-1} = \left(W_L^\top \delta^L\right) \odot \rho_{L-1}'(z_{L-1}) \in \mathbb{R}^{d_{L-1}}. \tag{8.3}$$

In the latter equality, $\odot$ is the componentwise product: for two vectors $a, b \in \mathbb{R}^d$,

$$a \odot b = \begin{pmatrix} a_1 b_1 \\ a_2 b_2 \\ \vdots \\ a_d b_d \end{pmatrix}.$$

Note that the dimensions agree in (8.3) as $z_{L-1}, W_L^\top \delta^L \in \mathbb{R}^{d_{L-1}}$, so that the componentwise product indeed makes sense. Note also that for backward operations, it is the transpose of the matrix W_L that appears (while in the forward pass W_L is being used).

Exercise 8.4. *Prove the equality* (8.3).

It can similarly be shown that

$$\delta^{\ell-1} = \left(W_\ell^\top \delta^\ell\right) \odot \rho_{\ell-1}'(z_{\ell-1}) \in \mathbb{R}^{d_{\ell-1}}.$$

The latter expression makes sense as the componentwise multiplication of $\rho_{\ell-1}'(z_{\ell-1}) \in \mathbb{R}^{d_{\ell-1}}$ and $W_\ell^\top \delta^\ell \in \mathbb{R}^{d_{\ell-1}}$. Here as well, the expression of $\delta^{\ell-1}$ involves the transpose matrix $W_\ell^\top$, which encodes the transpose operation allowing to go from the layer ℓ to the layer $\ell - 1$.

Once the vectors $(\delta^\ell)_{\ell=L,\dots,1}$ are determined, one can compute the gradients of the loss with respect to b_ℓ as

$$\nabla_{b_\ell}\widehat{\mathcal{R}}(\theta) = \delta^\ell,$$

where we used that $z_\ell = W_\ell\rho_{\ell-1}(z_{\ell-1}) + b_\ell$. The gradient with respect to W_ℓ is

$$\nabla_{W_\ell}\widehat{\mathcal{R}}(\theta) = \delta^\ell \rho_{\ell-1}(z_{\ell-1})^\top = \delta^\ell a_{\ell-1}^\top. \tag{8.4}$$

Note that the latter equality makes sense since $\delta^\ell \in \mathbb{R}^{d_\ell \times 1}$ while $a_{\ell-1}^\top = \rho_{\ell-1}(z_{\ell-1})^\top \in \mathbb{R}^{1\times d_{\ell-1}}$, so that $\nabla_{W_\ell}\widehat{\mathcal{R}}(\theta) \in \mathbb{R}^{d_\ell \times d_{\ell-1}}$ has the same dimension as W_ℓ.

Exercise 8.5. *Prove the equality* (8.4).

Final backpropagation algorithm. Let us summarize the procedure for the loss based on a single data point and for parameters $\theta = \{W_\ell, b_\ell\}_{1 \leqslant \ell \leqslant L}$:

- one first does a forward pass through the network, starting from the input $a_0 = x$, and evaluating, for $\ell = 1, \dots, L$:
$$z_\ell = W_\ell a_{\ell-1} + b_\ell, \qquad a_\ell = \rho_\ell(z_\ell).$$
This allows to perfom the final prediction $f_\theta(x) = a_L$. The associated loss is $\widehat{\mathcal{R}}(\theta) = \mathscr{L}(y, a_L)$.
- one then performs a backward pass, starting from the gradient of the output
$$\delta^L = \rho_L'(z_L) \partial_{\widehat{y}} \mathscr{L}(y, a_L),$$
and then evaluating, for $\ell = L, \dots, 2$:
$$\delta^{\ell-1} = \big(W_\ell^\top \delta^\ell\big) \odot \rho_{\ell-1}'(z_{\ell-1}).$$
The gradients with respect to the parameters are finally obtained for $\ell = 1, \dots, L$ as
$$\nabla_{W_\ell} \widehat{\mathcal{R}}(\theta) = \delta^\ell a_{\ell-1}^\top, \qquad \nabla_{b_\ell} \widehat{\mathcal{R}}(\theta) = \delta^\ell.$$

Extension to more complex architectures. The most recent neural networks have structures more complicated than feedforward multilayer perceptrons (see some elements in Section 8.4), and the backpropagation algorithm as described above is no longer sufficient. One example of a new element in the architecture is the presence of skip connections, where the value of a node at some layer ℓ is directly fed to a subsequent layer $\ell + k$ with $k \geqslant 2$ (*i.e.* not only to the next layer).

One resorts in these situations to automatic differentiation techniques, see [23, Section 6.5] for a more detailed presentation. In these methods, the dependence of the final output in terms in the input can be made precise with a computational graph (which is a directed acyclic graph). The determination of this dependency graph is made "just in time" in software such as JAX or PyTorch. As an example, one can construct the graph associated with the evaluation of the function
$$f(x_1, x_2) = x_2 \mathrm{e}^{x_1} \sqrt{x_1 + x_2 \mathrm{e}^{x_1}}.$$
This involves going from the input x_1 to e^{x_1}, then multiplying this value by the input x_2. This quantity is then added to the input x_1 using a skip connection from the input, and then passed through a square root; and also directly passed to the output using a skip connection in order to multiply the result of taking the square root. See Figure 8.1 for a graphical illustration.

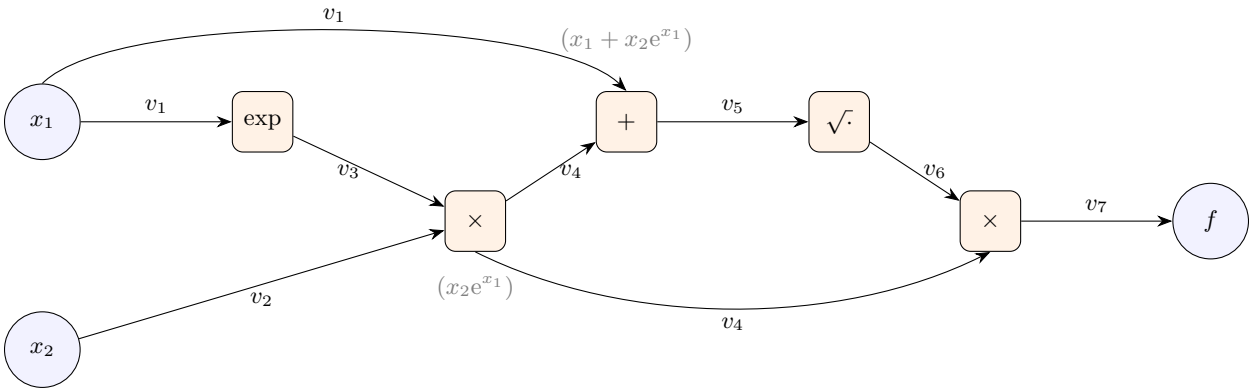

Fig. 8.1: Computational graph for $f(x_1, x_2) = x_2 \mathrm{e}^{x_1} \sqrt{x_1 + x_2 \mathrm{e}^{x_1}}$.

8.2.2 Optimization methods

The various optimization techniques presented in Chapter 4 are used to train neural networks. Early stopping is used throughout to monitor convergence and decide when to stop training. Let us already mention that there is a strong practical knowledge to be used to make training effective, in particular in the choice of the parameters of the algorithms (minibatch size, learning rates, etc). The algorithms which are particularly relevant to train neural networks are SGD (with or without momentum variables) and Adam. The latter algorithm is currently the most common choice. These methods work surprisingly well to minimize (regularized) empirical risks, despite the non-convexity of the problem. This fact is not well understood from a theoretical viewpoint.

Let us next discuss points specific to the training of neural networks, in particular elements which make the optimization easier, and allow to effectively train deep neural networks. The main point is to avoid gradients which degenerate, either by being too small or too large for certain parameters. When gradients are small, it is no longer possible to optimize; while, when gradients are too large, learning rates need to be severely limited, and/or the computational procedure crashes due to some overflow. As the computation of the gradient involves operations over series of layers, it is generically expected that the magnitude of the gradient either exponentially decreases or increases (this can be understood for instance when multiplying vectors by random matrices). It is possible to ensure that gradients are better behaved by

- choosing *activation functions* that do not saturate, for instance ReLU instead of sigmoids or hyperbolic tangent. The idea is that derivatives of activation functions should not be too small when the input of this function are large in absolute values, otherwise gradients will vanish;
- considering *loss functions* such as the cross entropy which involve a log and therefore allow to limit the saturation of the sigmoid function used to predict probabilities for classification problems. The use of properly chosen convex surrogate functions can have an impact on the quality of the training;
- introducing *batch normalization layers* (see [23, Section 8.7.1]). The idea is to add a layer to renormalize the outputs $a_\ell \in \mathbb{R}^{d_\ell}$ for the ℓ-th layer to express them as $a_\ell = \gamma_\ell \overline{a}_\ell + \beta_\ell$, where $\overline{a}_\ell$ has mean 0 and variance 1. The two parameters γ_ℓ, β_ℓ are learnt, and allow to set the scale of the outputs of the ℓ-th layer. This procedure does not change the expressivity of the neural network, but renders the numerical method better behaved;
- modifying the *architecture*. This is where, historically, most of the benefit came from. One idea is to consider "more linear" functions, as in ResNets where the update can be written as $a_{\ell+1} = a_\ell + \mathcal{F}_\ell(a_{\ell-1}; \theta_{\ell-1})$ for some of the blocks instead of the update $a_{\ell+1} = \mathcal{F}_\ell(a_{\ell-1}; \theta_{\ell-1})$, *i.e.* there is a dominant linear part to which a perturbation is added. Another idea is to resort to skip connections in order for certain parameters to have a shorter computational path to the output;
- *clipping gradients*, which means that a component g_k of the gradient is replaced by $\min(1, c/|g_k|)g_k$ for some parameter $c > 0$, so that the effective gradient is g_k for $|g_k| \leqslant c$, and $c\,\mathrm{sign}(g_k)$ otherwise;
- carefully *initializing the parameters* of the neural network. There are various rules here, depending on the activation function and the topology of the neural network. A typical rule is that the variance of the random variables used to initialize the parameters at a given node should be proportional to $1/(n_{\mathrm{in}} + n_{\mathrm{out}})$, where n_{in} is the number of incoming connections and n_{out} is the number of outgoing connections. In order to motivate such a rule, consider the simple case when

$$g = \sum_{m=1}^{n_{\mathrm{in}}} w_m x_m,$$

where $w_m \sim \mathcal{N}(0, \sigma^2)$ (one could also consider $w_m \sim \mathcal{U}[-\sigma, \sigma]$). Note that $\mathbb{E}(g) = 0$ and $\mathrm{Var}(g) = \mathbb{E}(g^2) = n_{\mathrm{in}}\sigma^2\gamma^2$ when the inputs $(x_m)_{1 \leqslant m \leqslant n_{\mathrm{in}}}$ are i.i.d. with $\mathbb{E}(x_m^2) = \gamma^2$. The variance of g is of order 1 for σ^2 of order $1/n_{\mathrm{in}}$. The above computation was motivated by the computation of properties in the forward pass of the neural network, but a similar argument can be made for the backward pass, in which case n_{in} should be replaced by n_{out}.

Refined heuristics can be worked out depending on the activation function at hand, see [23, Section 8.4]).

In terms of computational efficiency, it is often better to consider minibatches of sizes 2^k for some integer $k \geqslant 1$ due to the GPU memory formats. It is also good practice to shuffle the data before starting training in order to possibly destroy spurious correlations in the data set (which can arise for instance from the way data was collected). Note that second order methods (Newton or quasi-Newton techniques) are typically not used because of their computational cost; and also because there are generically many saddle points in high dimension, and not so many local minima in proportion, which is not good for Newton methods.

8.2.3 Regularization

Recall that regularization allows to trade some bias in the predictions for a limitation of the variance in these predictions, by limiting the capacity of the model. Regularization is particularly important for neural networks, which can have millions of parameters. Let us first emphasize that early stopping, which is used throughout, already provides some form of regularization. In addition to this, it is customary to consider additional penalization terms in the empirical risk function similar to the ones considered in Sections 2.3 and 2.4, namely $\|\theta\|_2$ or $\|\theta\|_1$. In fact, in practice, one usually considers only a penalization of the weights W_ℓ and not of the biases b_ℓ, as the latter parameters are merely used to recenter the outputs from one layer to the other. A ℓ^2 penalization of the weights corresponds to what is known as "weight decay" (the naming coming from the update of the parameter, which is of the form $\theta^{n+1} = (1 - \lambda\gamma)\theta^n + \gamma\widehat{F}(\theta^n)$ for some stochastic estimator of the gradient of the unregularized loss, so that the first term decreases the magnitude of the coefficients of θ); while a ℓ^1 penalization encourages some form of sparsity.

One regularization specific to neural networks is dropout. This consists in randomly switching off some connections in the neural network at training time, in order to avoid the co-adaptation of units to the data at hand, and allow for a more robust learning. In practice this is implemented by adding a Bernoulli random variable at each neuron, and removing the corresponding neuron with the associated probability[3] (no incoming nor outgoing connection). At validation/test time, several options are possible, such as not turning off units but renormalizing the weights ("weight scaling inference rule") or averaging the predictions over an ensemble of neural networks with randomly deactivated units. This arises from an interpretation of dropout as training an ensemble of networks, similarly to bagging strategies but with parameter sharing, see [23, Section 7.12] for further precisions.

A last comment is that it is currently believed that SGD-like algorithms have some regularizing effect, as they allow to converge to flat local minima which better generalize. This is known as "implicit regularization" and is an active field of research.

8.3 Unsupervised learning with neural networks

Autoencoders networks are an elegant tool for dimensionality reduction, denoising and generative modeling. Autoencoders have been considered early on in the neural network literature, where they were also called auto-associative neural networks [34]. The models considered in these early works correspond to what is currently known as bottleneck autoencoders, and were rather shallow. Deep autoencoders were used later on with the advent of modern computing architectures [26]. Bottleneck autoencoders were initially introduced to provide a nonlinear generalization of PCA, as it was shown that the linear neural networks obtained by minimizing the mean-square error were essentially equivalent to PCA [10, 5]. We refer for instance to [8, Section 12.4.2], [23, Chapter 14] and [41, Section 20.3] for textbook presentations of autoencoders, which include some historical perspectives, and mention many variations and extensions that were considered.

[3] It is not very clear what happens if all the neurons in a hidden layer are deactivated... This depends on the implementation of the model.

We first discuss the structure of autoencoders in Section 8.3.1, and then more precisely interpret their loss function in Section 8.3.2. The presentation of this section is taken from [36].

8.3.1 Structure of autoencoders

Autoencoders fall into the class of unsupervised machine learning methods. For a given input data point $x \in \mathcal{X} \subset \mathbb{R}^D$, we denote by $f_\theta(x)$ the prediction of the neural network. The parameters $\theta \in \Theta$ are chosen to minimize the expected risk

$$\mathcal{R}(\theta) = \mathbb{E}[\ell(X, f_\theta(X))], \tag{8.5}$$

where ℓ is a given elementary loss function, and the expectation is over the realizations of the input data X distributed according to some probability measure denoted by p_{data}. The typical choice for the latter elementary loss function is the square loss $\ell(x, y) = \|x - y\|^2$, although other choices, such as the mean absolute loss $\ell(x, y) = \|x - y\|$ could also be considered in order to give less weight to outliers. In practice, the risk $\mathcal{R}$ is replaced by the empirical risk over a training set of n given input data points $\mathcal{D} = \{x_1, \dots, x_n\}$:

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

Families of autoencoders. There are various classes of autoencoders. It is useful to distinguish between undercomplete and overcomplete models. Undercomplete models have a limited capacity that prevents them from achieving zero training loss. The most prominent example is provided by bottleneck autoencoders for which

$$f_\theta = f_{\text{dec},\theta_2} \circ f_{\text{enc},\theta_1}, \tag{8.6}$$

where the parameters $\theta = (\theta_1, \theta_2)$ have been decomposed into parameters used in the encoder and decoder parts, respectively (see Figure 8.2 below). The limitation in the capacity of the autoencoder arises from the fact that the encoding function f_{enc,θ_1} has values in a latent space $\mathcal{Z} \subset \mathbb{R}^D$ of dimension D strictly smaller than the dimension d of the input/output space $\mathcal{X}$, usually much smaller in fact. Overcomplete models can on the other hand achieve zero training error. These models are of course useless as such since they would simply copy the input without extracting the salient features explaining the data at hand. This is why some regularization process should be considered to limit the capacity of the neural network. Regularization is however also useful for undercomplete models. Standard examples of regularization mechanisms include:

- resorting to regularization strategies commonly used for neural networks in general, in particular early stopping, dropout, and standard weight decay to name a few options (see Section 8.2.3);
- sparse autoencoders where a penalization term is added to the loss function to prevent too many neurons to be active [42];
- denoising autoencoders [55], where outputs should be predicted from inputs corrupted by some noise, which forces networks to learn the structure of the distribution of the data [2];
- contractive autoencoders, where the Jacobian of the encoder is penalized in order to ensure smoother variations of this function, and limit its sensitivity to small variations in the input [46];
- variational autoencoders (VAEs) [32, 33, 22] can also be interpreted as some regularized version of the usual autoencoder framework.

Other variations/extensions of autoencoders were also considered for more specific purposes, and are still studied, in particular for manifold learning, where the preservation of neighborhood relationships and/or geometric information in the dimensionality reduction process are important [29, 17, 35].

In some applications, for instance molecular dynamics, overcomplete models in particular are not directly interesting in terms of dimensionality reduction.

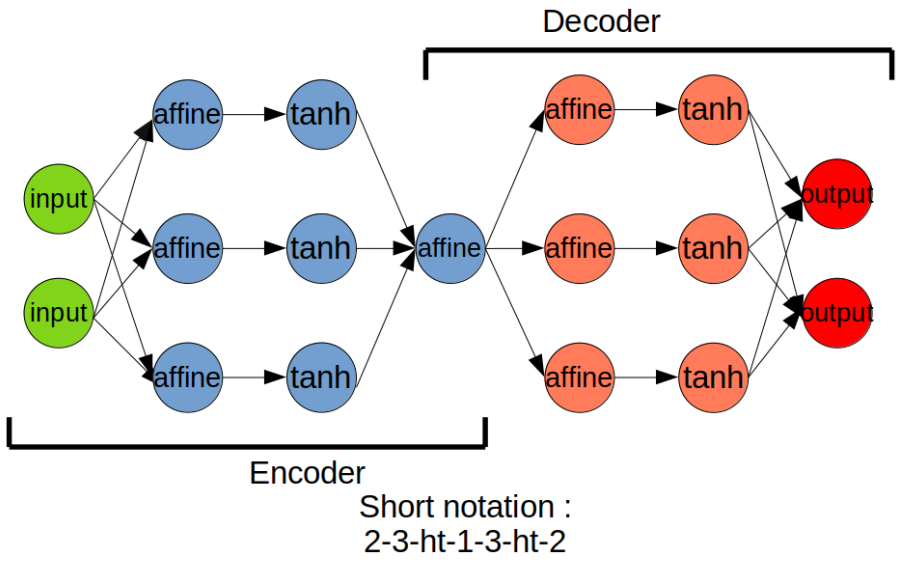

Fig. 8.2: Schematic representation of a symmetric autoencoder. Blue neurons correspond to hidden layers, while the input and output layers are respectively in green and red. Activation functions are hyperbolic tangents, except for the bottleneck and output layers, for which linear activation functions are considered in order not to restrict the range of values.

Autoencoders architectures. Autoencoders are often symmetric in their structures. In some cases, tied weights are being used, *i.e.* the weights θ_2 are the transpose of the weights θ_1 when writing the prediction function as (8.6). This choice reproduces at the level of autoencoders the symmetric structure of PCA, where the decoder matrix is the transpose of the encoder matrix (recall Lemma 5.1). In this case, a regularization on the encoder part only, as in contractive autoencoders, in fact also regularizes the decoder part. However, there is no particular motivation to use symmetric architectures, and various works, such as [51], studied the impact of an asymmetric architecture on the performance of the model. This point is further discussed below, where we motivate that decoders should be rather expressive in order to appropriately reproduce conditional expectations in the context of dimensionality reduction.

Another important consideration, more specific to bottleneck autoencoders, is the choice of the bottleneck dimension. In PCA, the number of dimensions to retain usually corresponds to some "knee" in the plot of eigenvalues of the empirical covariance matrix. Similar plots can be drawn for bottleneck autoencoders, for which the reconstruction error can be reported as a function of the bottleneck dimension (the remainder of the architecture being fixed). The optimal dimension should ideally coincide with the intrinsic dimension of the data set in manifold learning [15], which can be quantified using the Frechet inception distance (see for instance the method described in [28] in the context of VAEs).

8.3.2 Interpretations of the loss function

We discuss here various reformulations and reinterpretations of the loss function (8.5) for bottleneck autoencoders (8.6) when the loss function is the square loss, and discuss in particular the relationship with principal curves/manifolds [24, 53].

Three viewpoints on the loss function. We consider an ideal setting where we minimize upon all measurable functions $f_{\text{enc}} : \mathcal{X} \to \mathcal{Z}$ and $f_{\text{dec}} : \mathcal{Z} \to \mathcal{X}$. We denote by $\mathcal{F}_{\text{enc}}$ and $\mathcal{F}_{\text{dec}}$ the sets of measurable functions from $\mathcal{X}$ to $\mathcal{Z}$ and from $\mathcal{Z}$ to $\mathcal{X}$, respectively; and by $\mathcal{F}$ the set of measurable functions obtained by composing functions of $\mathcal{F}_{\text{enc}}$ with functions of $\mathcal{F}_{\text{dec}}$:

$$\mathcal{F} = \{f = f_{\text{dec}} \circ f_{\text{enc}},\ f_{\text{enc}} \in \mathcal{F}_{\text{enc}},\ f_{\text{dec}} \in \mathcal{F}_{\text{dec}}\}\,.$$

Minimizing the reconstruction error over the set of functions in $\mathcal{F}$ can be then rewritten as

$$\inf_{f \in \mathcal{F}} \mathbb{E}\left[\|X - f(X)\|^2\right]. \tag{8.7}$$

Note that we do not consider a regularization term here, so that overfitting may occur in practice when considering empirical risks (for instance, even with $\mathcal{Z}$ of dimension 1, f_{dec} can parametrize a space-filling curve).

As discussed in [19] (which complements [21] which was already hinting at autoencoders), the unsupervised least-square problem (8.7) can be thought of in various ways. In particular, there is some duality in the way the minimization over $f \in \mathcal{F}$ is performed, as one can decide to either

(i) simultaneously minimize over f_{enc} and f_{dec}, which is the standard way to proceed when training neural networks;
(ii) minimize first over the encoder part, which allows to reformulate the minimization as the well-known problem of finding principal manifolds;
(iii) minimize first over the decoder part, which is natural when thinking of the reconstruction error as some total variance to be decomposed using a conditioning on the values of the encoder.

The chosen numerical approach has a natural impact on the topology of the networks which are considered: in situation (i), encoders and decoders are treated on an equal footing, and it is therefore natural to consider them to be of a similar complexity; whereas options (ii) and (iii) suggest to consider asymmetric autoencoders. For instance, in option (iii), the minimization over the decoder part, which is performed first, could be done more carefully, with more expressive networks in order to better approximate the optimal decoder for a given encoder.

Principal manifold reformulation. We start by minimizing the reconstruction error (8.7) over the encoder part for a given decoder:

$$\begin{aligned}\inf_{f \in \mathcal{F}} \mathbb{E}\left[\|X - f(X)\|^2\right] &= \inf_{f_{\mathrm{dec}} \in \mathcal{F}_{\mathrm{dec}}} \left\{ \inf_{f_{\mathrm{enc}} \in \mathcal{F}_{\mathrm{enc}}} \mathbb{E}\left[\|X - f_{\mathrm{dec}} \circ f_{\mathrm{enc}}(X)\|^2\right] \right\} \\ &= \inf_{f_{\mathrm{dec}} \in \mathcal{F}_{\mathrm{dec}}} \mathbb{E}\left[\left\|X - f_{\mathrm{dec}} \circ h^\star_{f_{\mathrm{dec}}}(X)\right\|^2\right], \end{aligned} \tag{8.8}$$

where the optimal encoder $h^\star_{f_{\mathrm{dec}}} : \mathcal{X} \to \mathcal{Z}$ for a given decoder $f_{\mathrm{dec}} : \mathcal{Z} \to \mathcal{X}$ is defined pointwise as

$$h^\star_{f_{\mathrm{dec}}}(x) \in \operatorname*{argmin}_{z \in \mathcal{Z}} \|x - f_{\mathrm{dec}}(z)\|,$$

provided that this minimization problem admits a solution. When f_{dec} is smooth and has an invertible Jacobian, the principal manifold is then the set $f_{\mathrm{dec}}(\mathcal{Z}) \subset \mathcal{X}$. For any $x \in \mathcal{X}$, $h^\star_{f_{\mathrm{dec}}}(x) \in \mathcal{Z}$ gives the coordinates in the latent space of the projection of x on the principal manifold (the so-called projection index).

The reformulation of the minimization problem in terms of the decoder function leads to a minimization problem similar to the one encountered when searching for principal curves and manifolds. These mathematical concepts generalize in some sense PCA to curves and surfaces rather than lines and hyperplanes, as already discussed in the work introducing principal curves [24]. The minimization problems associated with finding principal manifolds are in general difficult to solve as these manifolds correspond to saddle-points of the loss functional [16]. In practice, this means that it is possible to move away from a critical point without increasing the test loss. This leads to overfitting issues and prevents the use of traditional cross-validation procedures to tune regularization hyperparameters.

Reformulating autoencoders with conditional expectations. We discuss here how to reformulate the training of autoencoders with conditional expectations, and provide alternative interpretations to the reconstruction error. In contrast to (8.8), we minimize here the reconstruction error (8.7) by first minimizing over the decoder part for a given encoding function, as already considered in [20]. This approach is natural in molecular dynamics. From a mathematical viewpoint, it corresponds to introducing conditional averages associated with fixed values of the encoder.

The loss function for unsupervised least-squares can be rewritten as

$$\begin{aligned}\inf_{f\in\mathcal{F}} \mathbb{E}\left[\|X-f(X)\|^2\right] &= \inf_{f_{\rm enc}\in\mathcal{F}_{\rm enc}} \left\{ \inf_{f_{\rm dec}\in\mathcal{F}_{\rm dec}} \mathbb{E}\left[\|X-f_{\rm dec}\circ f_{\rm enc}(X)\|^2\right]\right\} \\ &= \inf_{f_{\rm enc}\in\mathcal{F}_{\rm enc}} \mathbb{E}\left[\left\|X-g^\star_{f_{\rm enc}}\circ f_{\rm enc}(X)\right\|^2\right], \end{aligned} \tag{8.9}$$

where the ideal decoder $g^\star_{f_{\rm enc}}$ for a given encoder $f_{\rm enc}$ is the Bayes predictor associated with the least square regression problem (similarly to what is done in Exercise 1.6; see Exercise 8.6):

$$g^\star_{f_{\rm enc}}(z) = \mathbb{E}[\,X \,|\, f_{\rm enc}(X)=z]. \tag{8.10}$$

Let us recall that, in all these expressions, expectations are taken with respect to the probability distribution $p_{\rm data}$ of the input data. Equations (8.9)-(8.10) show that the question of finding the best autoencoder can be reduced to finding the best encoding function, provided that one is able to compute good approximations of the conditional expectation.

Exercise 8.6. *Prove the equality* (8.10).

The reconstruction error (8.9) can be reinterpreted in terms of variances. Indeed, on the one hand,

$$\begin{aligned}\mathbb{E}\left[\left\|X-g^\star_{f_{\rm enc}}\circ f_{\rm enc}(X)\right\|^2\right] &= \mathbb{E}\left[\|X-\mathbb{E}\left[X|f_{\rm enc}(X)\right]\|^2\right] \\ &= \mathbb{E}\left[\mathbb{E}\left(\|X-\mathbb{E}\left[X|f_{\rm enc}(X)\right]\|^2\Big| f_{\rm enc}(X)\right)\right] \\ &= \mathbb{E}\left[{\rm Var}(X|f_{\rm enc}(X))\right]. \end{aligned} \tag{8.11}$$

On the other hand,

$$\begin{aligned}\mathbb{E}\left[\left\|X-g^\star_{f_{\rm enc}}\circ f_{\rm enc}(X)\right\|^2\right] &= \mathbb{E}\left[\|X-\mathbb{E}\left[X|f_{\rm enc}(X)\right]\|^2\right] \\ &= \mathbb{E}\left(\|X\|^2\right) - \mathbb{E}\left(\mathbb{E}\left[X|f_{\rm enc}(X)\right]^2\right) \\ &= {\rm Var}(X) - {\rm Var}\left[\mathbb{E}(X|f_{\rm enc}(X))\right]. \end{aligned} \tag{8.12}$$

These two equalities yield the well-known formula for the total variance decomposition, namely ${\rm Var}(X) = \mathbb{E}\left[{\rm Var}(X|f_{\rm enc}(X))\right] + {\rm Var}\left[\mathbb{E}(X|f_{\rm enc}(X))\right]$.

A consequence of (8.12) is that the minimization problem (8.9) can be reformulated as the following equivalent maximization problem:

$$\sup_{f_{\rm enc}\in\mathcal{F}_{\rm enc}} {\rm Var}\left[\mathbb{E}(X|f_{\rm enc}(X))\right]. \tag{8.13}$$

In words, this reformulation translates the equivalence between (the "classes" referring here to the level sets of $f_{\rm enc}$)

- minimizing the intraclass dispersion (8.11): the distribution of configurations $x\in\mathcal{X}$ for a fixed value z of $f_{\rm enc}$ should concentrate around the mean value $g^\star_{f_{\rm enc}}(z)$ by having a variance as small as possible;
- maximizing the interclass dispersion (8.13): the values of the conditional averages of X for fixed values of $f_{\rm enc}$ should be as spread out as possible over the range of $f_{\rm enc}$.

Let us conclude for providing a formal characterization of the optimal encoding function. A key equality to establish (8.12), namely

$$\mathbb{E}\left[\left\|X-g^\star_{f_{\rm enc}}\circ f_{\rm enc}(X)\right\|^2\right] = \mathbb{E}\left[\|X\|^2\right] - \mathbb{E}\left[\left\|g^\star_{f_{\rm enc}}\circ f_{\rm enc}(X)\right\|^2\right],$$

shows that the minimization of the reconstruction error is equivalent to the maximization of the second moment of the conditional expectation:

$$\sup_{f_{\mathrm{enc}} \in \mathcal{F}_{\mathrm{enc}}} \mathbb{E}\left[\left\| g^{\star}_{f_{\mathrm{enc}}} \circ f_{\mathrm{enc}}(X) \right\|^2\right] . \tag{8.14}$$

This alternative viewpoint allows to characterize the optimal encoding function f_{enc} by some orthogonality condition similar to the self-consistency condition of principal curves, see [21, Section 2]. In fact, it can be formally shown that critical points of (8.14) satisfy

$$\forall j \in \{1, \dots, d\}, \quad \forall x \in \mathrm{Supp}(p_{\mathrm{data}}), \qquad \left[x - g^{\star}_{f_{\mathrm{enc}}}(f_{\mathrm{enc}}(x))\right]^{\top} \partial_{z_j} g^{\star}_{f_{\mathrm{enc}}}(f_{\mathrm{enc}}(x)) = 0, \tag{8.15}$$

where $\mathrm{Supp}(p_{\mathrm{data}})$ is the support of the probability measure p_{data}. The derivation of this condition, which can be read in [36, Appendix A], can be seen as a variation of derivations of optimality conditions for principal curves, as written already in [24]; see also [21] where (8.15) is used to construct a new objective function to minimize in order to find f_{enc}.

An interesting implication of (8.15) is that the intersection of $\mathrm{Supp}(p_{\mathrm{data}})$ and the submanifold

$$\Sigma_z = f_{\mathrm{enc}}^{-1}\{z\} = \{x \in \mathcal{X} \,|\, f_{\mathrm{enc}}(x) = z\}$$

is in fact included in the $(d-D)$-dimensional hyperplane containing the point $g^{\star}_{f_{\mathrm{enc}}}(z)$ and orthogonal to the vectors $\partial_{z_1} g^{\star}_{f_{\mathrm{enc}}}(z), \dots, \partial_{z_d} g^{\star}_{f_{\mathrm{enc}}}(z)$ (recalling that $\mathcal{X}$ and $\mathcal{Z}$ have dimensions d and D, respectively). As these hyperplanes generally have a non-empty intersection, finding a regular function f_{enc} which satisfies (8.15) is only possible for distributions p_{data} which have a support sufficiently concentrated around the principal manifold. The issue of having hyperplanes intersecting can be seen as the counterpart in the context we consider here of the concept of ambiguity points for principal curves [24].

8.4 Other types of neural networks

We (very) briefly mention in this section neural network architectures which go beyond the simple feedforward multilayer perceptrons. These modern architectures, which can be rather complicated, are key to the current successes of machine learning, for classification and regression, and also more and more for generative tasks such as those related to natural language processing.

Convolutional neural networks. Convolutional neural networks (CNNs) are a default choice for image processing. Their motivation is threefold:

- they allow to treat inputs of variable sizes (depending on the resolution of the image under consideration for instance);
- they can handle inputs of large dimension, coming in the form width × height × input channel. For images, the input channel is typically RGB, *i.e.* the input for color images is composed of three matrices of pixels intensities, for the colors red, green and blue;
- they naturally encode some translation invariance and various symmetries. This is important for image classification for instance, as the main element in the image can be rotated, translated, etc.

From a structural viewpoint, matrix multiplications in CNN are replaced by convolution operations, which work with filters that take local averages of pictures. In practice, CNNs alternate between convolutional layers and pooling layers (see for instance [41, Figure 14.13]). Pooling layers take the minimum or maximum of an input over certain regions; this allows to reduce the sensitivity to the location of elements in the image.

Famous examples of CNNs include (see [41, Section 14.3]): LeNet, AlexNet (2012; this is the network which reintroduced neural networks in machine learning, by its dramatic improvement in classification performance), GoogLeNet (2015), ResNet (2015, Microsoft), DenseNet, ... All these networks are constructed on the similar building blocks, which are rearranged in various manners.

They can be used for other tasks than classification, such as image tagging, object detection, segmentation, ...

Complete the paragraphs below, and also possibly

Graph neural networks. These networks use connections between nodes which are not organized in a strictly sequential manner, but rather on a graph. This can be useful for various tasks, for instance the prediction of atomic forces in molecular models; in this situation, the nodes of the graph correspond to an atom, and the connections between the nodes, which provide some transfer of information, mimick the interactions between atoms.

Recurrent neural networks. These networks are used to generate sequences or trajectories $y_{1:T}$, for instance for language modeling. They include long short term memory (LSTM) blocks to have better gradients for training and ensure that the memory in sequences is better taken into account.

Attention networks. These networks use a dictionary lookup where a query/input is compared to various keys, and the output depends on the key.

Transformers. Many successful current large language models, such as ChatGPT, are based on networks of GPT type ("generative pre-trained transformers"). They include attention layers – self-attention in fact. They also make use of positional encoding. Finally, they are made to scale efficiently with the data size. See [9, Chapter 12] for further precisions.

9

Clustering methods

We discuss in this chapter a set of unsupervised techniques aiming at grouping together data points which are similar in a common class. We discuss more precisely the aim of clustering in Section 9.1, and then successively present various techniques, starting in Section 9.2 with K-means clustering, turning next in Section 9.3 to hierarchical clustering, then in Section 9.4 to clustering with mixture models, and concluding with density based clustering in Section 9.5 and spectral clustering in Section 9.6. Our presentation is based on [41, Chapter 20], [50, Chapter 22 and Section 24.4], [25, Sections 14.3 and 6.8], as well as [39, Section 13].

9.1 Aims and scope of clustering

We consider a set $\mathcal{D}$ of n data points $\{x_1, \ldots, x_n\} \subset \mathcal{X}$, which are unlabelled. The informal aim of clustering is to group data points into clusters – similar elements should be in the same class, while dissimilar elements should be in different classes. In fact, one should talk about segmentation rather than clustering when the data is not well separated. Applications include grouping families of genes in biology, customers depending on their behavior for marketing, communities in social media, atomic configurations in molecular dynamics, etc.

There are two major difficulties with clustering:

(1) The precise definition of clustering (*i.e.* a quantitative definition) is rather difficult to make rigorous as the notion of similarity is not a transitive relation, while cluster assignement is. To illustrate the issue at stake, think of two parallel lines of data points regularly spaced, the separation between the two lines being quite larger than the separation between the points in each line. If one emphasizes grouping close-by points, then each line would be a cluster, each data point being close to its neighbors, but the extremal points on each line can be quite different;

(2) Another difficulty is the lack of ground truth, which is a usual problem in unsupervised learning. There may be many ways to cluster a given data set. In particular, the distance or metric used for clustering can have a dramatic impact on the results. Think for example of grouping movies by main actor, topic, rating, year, etc. The quality of the clustering can be assessed if labelled data is available, by evaluating the purity of each cluster; or if some reference clustering is available (through the Rand index computed from the number of true/false positives/negatives; see [41, Section 21.1.1.2]).

On the practical side, let us mention two points:

(i) normalizing the data set is not a good idea in general as the notion of closedness is not invariant by rescaling. Normalization could however still be relevant for some problems. Whether to normalize or not is problem specific, requires some domain knowledge, and possibly some trial and error procedure;
(ii) clustering in high dimension can be challenging as data points are generically all far away from each other, see Exercise 9.1 below. It makes sense to perform some dimensionality reduction as a pre-processing step, using for instance PCA (see Chapter 5) or autoencoders (see Section 8.3).

Exercise 9.1. *Consider a data composed of two clusters of points in $\mathbb{R}^d$, independently drawn from Gaussian distributions with means 0 and μ respectively, and identity covariance in both cases. Compute the expected distance between points within one of the clusters, and the expected distance between two points in different clusters. How easy is it to distinguish these quantities as d increases?*

9.2 *K*-means and center-based clustering methods

The aim of K-means clustering is to group the data points into K clusters. Using a combinatorial algorithm to this end is impossible because the number of possible partitions into K classes is too large (see [25, Equation (14.30)] for the formula giving the number of classes). We consider here a method to perform an approximate clustering based on some cost function, with an optimization procedure to find the assignement function $\mathcal{X} \to \{1, \dots, K\}$. The cost function is itself based on a distance function $\mathtt{d} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}_+$, for instance the Euclidean distance.

Loss function. Let us write the loss function for the K-means algorithm when $\mathcal{X} = \mathbb{R}^d$, the chosen distance is the Euclidean norm, *i.e.* $\mathtt{d} = \|\cdot\|_2$, and the number of classes $K \geqslant 2$ is fixed/predetermined. The algorithm updates a partition $C_1, \dots, C_K$ of $\mathcal{D}$, and considers the loss function

$$J(C_1, \cdots, C_K) = \min_{(m_1,\dots,m_K)\in(\mathbb{R}^d)^K} \left\{ \sum_{k=1}^{K} \sum_{x\in C_k} \|x - m_k\|_2^2 \right\} = \sum_{k=1}^{K} \sum_{x\in C_k} \|x - \overline{x}_k\|_2^2 , \tag{9.1}$$

where

$$\overline{x}_k = \frac{1}{|C_k|} \sum_{x\in C_k} x$$

is the empirical mean of the data points in the cluster C_k.

Exercise 9.2. *Prove that the second equality in* (9.1) *holds.*

Remark 9.1 (Formulation using an assigmement function/matrix). *Note that*

$$\sum_{k=1}^{K} \sum_{x\in C_k} \|x - m_k\|_2^2 = \sum_{i=1}^{n} \left\|x_i - m_{z(i)}\right\|_2^2 = \left\|X - ZM^\top\right\|^2 ,$$

where $X \in \mathbb{R}^{n\times d}$ is the matrix whose i-th line is x_i, the matrix $M = [m_1|m_2|\dots|m_K] \in \mathbb{R}^{d\times K}$ has the vectors m_k as columns, and $Z \in \{0,1\}^{n\times K}$ is the assignement matrix, with entries $Z_{i,k} = \mathbf{1}_{z(i)=k}$ for

$$z(i) \in \operatorname*{argmin}_{1\leqslant k\leqslant K} \|x_i - m_k\|_2$$

the assignement function.

***K*-means algorithm.** The precise algorithm is the following.[1] Starting from some initial partition $(C_1^{(0)}, \cdots, C_K^{(0)})$, and given a partition $(C_1^{(t)}, \cdots, C_K^{(t)})$ at step $t \geqslant 0$:

- compute the centers $\overline{x}_k^{(t)}$ of the cluster $C_k^{(t)}$;
- for every data point x_i with $1 \leqslant i \leqslant n$, find the index $z_{t+1}(i) \in \{1, \ldots, K\}$ of the center $(\overline{x}_k^{(t)})_{1 \leqslant k \leqslant K}$ closest to x_i (with some rule to break ties):

$$z_{t+1}(i) \in \operatorname*{argmin}_{1 \leqslant k \leqslant K} \left\| x_i - \overline{x}_k^{(t)} \right\|_2 ;$$

- update the clusters as $C_k^{(t+1)} = \{x_j \in \mathcal{D} : z_{t+1}(j) = k\}$.

The algorithm is iterated until convergence (*i.e.* $(C_1^{(t+1)}, \cdots, C_K^{(t+1)}) = (C_1^{(t)}, \cdots, C_K^{(t)})$). Note that the computational cost of a single iteration is $\mathrm{O}(n)$. The assignement step is based on a Voronoi tesselation of the space based on the cluster centers.

It is good practice to run several times the algorithm with random initializations and retain the clustering leading to the smallest value of the objective function (9.1). A typical way to start with a random partition is to draw at random K cluster centers, either fully at random (all points are uniformly sampled over the full space) or by emphasizing more uniform clusters in order to better cover the data. This can be achieved by choosing the random centers in a sequential manner, and emphasizing points which are further apart from the current centers (the so-called K-means++ method, which is the default initialization method in scikit-learn for instance); see [41, Section 21.3.4].

Convergence of the *K*-means algorithm. We present a result on the evolution of the loss function (9.1), which however does not give insight on the number of iterations of the algorithm. This also does not provide indication on the quality of the so-obtained clustering – for instance because the minimization problem is non convex, and there is no guarantee that the minimal loss is obtained.

Lemma 9.1. *Each iteration of the K-means algorithm does not increase the objective function* (9.1)*:*

$$\forall t \geqslant 0, \qquad J\left(C_1^{(t+1)}, \cdots, C_K^{(t+1)}\right) \leqslant J\left(C_1^{(t)}, \cdots, C_K^{(t)}\right). \tag{9.2}$$

Exercise 9.3. *The aim of this exercise is to prove Lemma 9.1. Show first that*

$$J\left(C_1^{(t)}, \cdots, C_K^{(t)}\right) \geqslant \sum_{k=1}^{K} \sum_{x_i \in C_k^{(t+1)}} \left\| x_i - \overline{x}_k^{(t)} \right\|_2^2,$$

next that

$$\sum_{x_i \in C_k^{(t+1)}} \left\| x_i - \overline{x}_k^{(t)} \right\|_2^2 \geqslant \sum_{x_i \in C_k^{(t+1)}} \left\| x_i - \overline{x}_k^{(t+1)} \right\|_2^2,$$

and conclude.

The proof of Lemma 9.1 shows that if equality holds in (9.2), then the partitions $\left(C_1^{(t+1)}, \cdots, C_K^{(t+1)}\right)$ and $\left(C_1^{(t)}, \cdots, C_K^{(t)}\right)$ coincide[2] as the centers of the clusters are unchanged.

[1] See the website http://www.bytemuse.com/post/k-means-clustering-visualization/ for a nice graphical illustration.

[2] Except maybe for points equidistant from two centers or more.

Variations of the K-means algorithm. We present three variations of the K-means algorithm:

- *K-medoids*, for which the empirical average in a cluster is replaced by some element of the dataset. This allows to apply K-means type algorithms to data which is not in $\mathbb{R}^d$, for which averages may be undefined (think of categorical variables, or matrices with positivity or norm constraints). The objective function to be considered instead of (9.1) reads

$$\min_{(m_1,\dots,m_K)\in\mathcal{D}^K} \sum_{k=1}^{K} \sum_{x\in C_k} \mathtt{d}(x,m_k)^2.$$

  There are various approaches to try and solve this problem. A common one is "partitioning around medoids" [30], but it is computationally rather expensive. A more recent and less expensive option is the Voronoi iteration of [45], which has a cost $\mathrm{O}(nK)$ per iteration and works by reassigning points and updating the medoids, very similarly to what is done for K-means.
- *K-median* builds upon K-medoids but relies on a loss function which uses a distance rather than a squared distance as in (9.1), namely

$$\min_{(m_1,\dots,m_K)\in(\mathbb{R}^d)^K} \sum_{k=1}^{K} \sum_{x\in C_k} \mathtt{d}(x,m_k).$$

  Considering unsquared norms may be better for problems such as facility location (think of houses as data points and look for places where to position K firestations).
- *K-autoencoders*, introduced in [44], consider the loss function

$$\sum_{i=1}^{n} \min_{1\leqslant k\leqslant K} \left\|x_i - \Phi_{\theta_k}(x_i)\right\|^2,$$

  which is minimized with respect to the parameters $\theta_1,\dots,\theta_K$ for each autoencoder. This can therefore be seen as some extension of K-means where $\Phi_{\theta_k}(x_i)$ replaces the cluster empirical average. The cluster assignement is performed based on the index of the autoencoder for which the reconstruction loss is minimized, namely

$$C_k = \left\{ x_i \in \mathcal{X} \;\middle|\; k = \operatorname*{argmin}_{1\leqslant k'\leqslant K} \left\|x_i - \Phi_{\theta_{k'}}(x_i)\right\|^2 \right\}.$$

Choosing the number of clusters K. There is a monotonic decrease of the minimal value for the objective function (9.1) when K is increased, so that (as in PCA) it is not possible to use some form of cross-validation to identify the best value of K. There are two main ways to find an appropriate value for K:

- rely on probabilistic models (such as the Gaussian mixture models consider in Section 9.4) and use the tools of Bayesian model selection to determine K;
- look for a kink/elbow in the within cluster distances (see [25, Section 14.3.11]) or in silhouette coefficients (see [41, Section 21.3.7.3]).

9.3 Hierarchical clustering

There are two types of hierarchical clustering:

- *agglomerative*, in which case one starts from n clusters composed of single data points and merges them;
- *divisive*, in which case one starts from a single cluster with all data points, and splits it. This approach is not as well studied and less common (see [25, Section 14.3.12]).

A nice property of hierarchical clustering is its nested property, and the fact that one does not need to specify the desired number of clusters. Partitions of the data are obtained by transforming dendrograms representing the hierarchical structure using two possible termination criteria: cutting when a fixed number of clusters is attained, or when a certain upper bound in distance is reached.

Hierarchical agglomerative clustering. In order to make these statements more precise, we present agglomerative clustering in more detail. The key element to specify the method is a distance $\mathtt{D}$ between clusters, which allows to determine the closest clusters, which should be merged. The computational cost of the algorithm is a priori $\mathrm{O}(n^3)$ (as picking the two most similar clusters has a cost $\mathrm{O}(n^2)$, and this operation should be repeated $\mathrm{O}(n)$ times). The precise algorithm is the following. Starting from the n clusters $C_i^{(0)} = \{x_i\}$, iterate on $0 \leqslant t \leqslant n-2$:

- select among the remaining $n-t$ clusters the pair $\left(C_k^{(t)}, C_\ell^{(t)}\right)$ which minimizes the distance between clusters:
$$(k,\ell) \in \operatorname*{argmin}_{(a,b):1\leqslant a<b\leqslant n-t} \mathtt{D}\left(C_a^{(t)}, C_b^{(t)}\right);$$
- form a new partition $C_1^{(t+1)}, \dots, C_{n-t-1}^{(t+1)}$ by keeping the previous clusters $C_a^{(t)}$ for $a \in \{1, \dots, n-t\} \setminus \{k,\ell\}$, and adding the merged cluster $C_k^{(t)} \cup C_\ell^{(t)}$.

Choice of the distance. The key element in this algorithm is the choice of the distance $\mathtt{D}$. The result crucially depends on it. We present three choices here:

- *single link* is based on the distance between the two closest elements of C_k and C_ℓ:
$$\mathtt{D}(C_k, C_\ell) = \min_{\substack{x_i\in C_k\\ x_j\in C_\ell}} \mathtt{d}(x_i, x_j).$$
A possible issue with this choice is chaining, which corresponds to grouping together data points related by a series of close intermediate points, the overall cluster being however not very compact (somehow elongated, hence the name);
- *complete link* is based on the distance between the two farthest elements of C_k and C_ℓ:
$$\mathtt{D}(C_k, C_\ell) = \max_{\substack{x_i\in C_k\\ x_j\in C_\ell}} \mathtt{d}(x_i, x_j).$$
This metric emphasizes compact clusters but may lead to "closedness losses" (two elements of the same cluster can be separated because they are somewhat far away from each other);
- *average link* considers the average distance between the elements of C_k and C_ℓ:
$$\mathtt{D}(C_k, C_\ell) = \frac{1}{|C_k|\,|C_\ell|} \sum_{x_i\in C_k} \sum_{x_j\in C_\ell} \mathtt{d}(x_i, x_j).$$
It somewhat interpolates between the single and complete link, and hence allows to form relatively compact clusters with elements that are relatively close. It is however sensitive to monotonic transformations of the baseline distance $\mathtt{d}$.

There are many other possible choices, in particular Ward's distance (see Exercise 9.4 below). The main issue is the algorithmic complexity of the resulting algorithm. Hierarchical clustering is suitable for small data sets, or should be initialized with K-means with a large value of K, however sufficiently small compared to the number of data points (*i.e.* $1 \ll K \ll n$).

Exercise 9.4 (Ward's distance for hierarchical clustering). *Consider two disjoint clusters C and C', with means $\overline{x}_C$ and $\overline{x}_{C'}$ respectively, and denote by $\overline{x}_{C\cup C'}$ the mean of $C \cup C'$. The aim of this exercise is to prove that the Ward distance*

$$\mathtt{D}(C, C') := \frac{|C|\,|C'|}{|C| + |C'|} \left\|\overline{x}_C - \overline{x}_{C'}\right\|^2$$

is such that

$$\sum_{x\in C}\|x-\overline{x}_C\|^2+\sum_{x\in C'}\|x-\overline{x}_{C'}\|^2+\mathtt{D}(C,C')=\sum_{x\in C\cup C'}\|x-\overline{x}_{C\cup C'}\|^2.$$

(a) Interpret this equality.
(b) Show that $\sum_{x\in C}\|x-\overline{x}_{C\cup C'}\|^2=|C|\ \|\overline{x}_C-\overline{x}_{C\cup C'}\|^2+\sum_{x\in C}\|x-\overline{x}_C\|^2$.
(c) Prove that

$$\sum_{x\in C\cup C'}\|x-\overline{x}_{C\cup C'}\|^2-\sum_{x\in C}\|x-\overline{x}_C\|^2-\sum_{x\in C'}\|x-\overline{x}_{C'}\|^2 \\ =|C|\ \|\overline{x}_C-\overline{x}_{C\cup C'}\|^2+|C'|\ \|\overline{x}_{C'}-\overline{x}_{C\cup C'}\|^2.$$

(d) Express $\overline{x}_{C\cup C'}$ *as a convex combination of* $\overline{x}_C$ *and* $\overline{x}_{C'}$.
(e) Conclude.

Hierarchical clustering can be nicely visualized using dendrograms (see for instance the presentation in [25, Section 14.3.12]). In this representation, the data points are placed on the horizontal axis, at zero height. When two points are merged to form a cluster, say x_1 and x_2, then one draws a vertical line starting from x_1 and x_2, until the height corresponding to the distance $\mathtt{D}(\{x_1\},\{x_2\})$ where a horizontal line is drawn. This is repeated for other data points and when merging clusters, until a single cluster remains. In this representation, the height of each node in the resulting binary tree is proportional to the distance between the two daughter clusters. The tree can indeed be plotted as such due to the monotone increasing character of the distances between clusters.

9.4 Clustering with mixture models

Mixture models make use of a latent/hidden variable in the context of a generative model. Here, this latent variable is the cluster identity of a data point, the overall distribution of data points being a mixture of Gaussian distributions with different means and variances. This requires making an assumption on the form of the data distribution, in accordance with the "no free lunch theorem" (see Section 10.3).

Gaussian mixture model. The algorithm consists in determining the most likely value of the latent variable z_i for a given data point x_i. In fact, one determines the vector of discrete probabilities to belong to a cluster. The precise model for the density of the data distribution is

$$p_\theta(x)=\sum_{k=1}^K\pi_k\mathscr{G}_{\mu_k,\Sigma_k}(x),\qquad \mathscr{G}_{\mu,\Sigma}(x)=(2\pi)^{-d/2}\det(\Sigma)^{-1/2}\exp\left(-\frac{1}{2}(x-\mu)^\top\Sigma^{-1}(x-\mu)\right),$$

with parameters

$$\theta=(\pi_1,\dots,\pi_K,\mu_1,\dots,\mu_K,\Sigma_1,\dots,\Sigma_K)\in\Theta,$$

where

$$\Theta=\left\{\theta\in[0,1]^K\times\left(\mathbb{R}^d\right)^K\times\left(\mathbb{R}^{d\times d}\right)^K\ \middle|\ \sum_{k=1}^K\pi_k=1,\quad \Sigma_k=\Sigma_k^\top>0\right\}.$$

This models corresponds to having a prior distribution $(\pi_1,\dots,\pi_K)$ on the latent variable z_i, and a conditional distribution $p_\theta(x_i\,|\,z_i=k)=\mathscr{G}_{\mu_k,\Sigma_k}(x_i)$. The likelihood of a data point x is obtained by marginalizing out the latent variable, *i.e.*

$$p_\theta(x)=\sum_{k=1}^K p_\theta(z_i=k)p_\theta(x_i\,|\,z_i=k).$$

Bayes' rule is used to obtain the responsibility for each data point, *i.e.* the posterior membership probability:

$$r_{i,k} := p_\theta(z_i = k \,|\, x_i) = \frac{p_\theta(z_i = k) p_\theta(x_i \,|\, z_i = k)}{\sum_{k'=1}^{K} p_\theta(z_i = k') p_\theta(x_i \,|\, z_i = k')} = \frac{\pi_k \mathscr{G}_{\mu_k, \Sigma_k}(x_i)}{\sum_{k'=1}^{K} \pi_{k'} \mathscr{G}_{\mu_{k'}, \Sigma_{k'}}(x_i)}.$$

The latter formula can be seen as an application of the general Bayes formula to the case of Gaussian mixture models (as the previous equation can be rewritten as $p_\theta(z_i = k \,|\, x_i) = p_\theta(z_i = k) p_\theta(x_i \,|\, z_i = k) / p_\theta(x_i)$). Assignement can be made in various ways, for instance through the "hard assignement" rule

$$z_i \in \operatorname*{argmax}_{1 \leqslant k \leqslant K} r_{i,k}.$$

Estimation of the parameters of the model. The difficulty in this approach is that the vector of parameter θ needs to be estimated. This can be done using some maximum likelihood method, in which case one obtains

$$\widehat{\theta} \in \operatorname*{argmax}_{\theta \in \Theta} \{\log p_\theta(x_1, \ldots, x_n)\} = \operatorname*{argmax}_{\theta \in \Theta} \left\{ \sum_{i=1}^{n} \log \left(\sum_{k=1}^{K} \pi_k \mathscr{G}_{\mu_k, \Sigma_k}(x_i) \right) \right\}. \tag{9.3}$$

Gradient-like methods could be used to find (local) maxima. A more standard strategy in this context is however to rely on Expectation-Maximization (EM) algorithms, which ensure that the likelihood is non decreasing.

The precise EM algorithm is the following. Starting from an initial guess θ^0, iterate on $t \geqslant 0$:

- (*expectation step*) update the responsibilities as

$$r_{i,k}^{(t)} = \frac{\pi_k^{(t)} \mathscr{G}_{\mu_k^{(t)}, \Sigma_k^{(t)}}(x_i)}{\sum_{k'=1}^{K} \pi_{k'}^{(t)} \mathscr{G}_{\mu_{k'}^{(t)}, \Sigma_{k'}^{(t)}}(x_i)};$$

- (*maximization step*) update θ^{t+1} as

$$\begin{aligned} \pi_k^{(t+1)} &= \frac{1}{n} \sum_{i=1}^{n} r_{i,k}^{(t)} \in [0,1], \qquad \mu_k^{(t+1)} = \frac{\sum_{i=1}^{n} r_{i,k}^{(t)} x_i}{\sum_{i=1}^{n} r_{i,k}^{(t)}} \in \mathbb{R}^d, \\ \Sigma_k^{(t+1)} &= \frac{\sum_{i=1}^{n} r_{i,k}^{(t)} \left(x_i - \mu_k^{(t+1)}\right) \left(x_i - \mu_k^{(t+1)}\right)^\top}{\sum_{i=1}^{n} r_{i,k}^{(t)}} \in \mathbb{R}^{d \times d}. \end{aligned} \tag{9.4}$$

In the previous expressions, the data points x_i are considered as column vectors $\mathbb{R}^{d \times 1}$. This update corresponds to recomputing the first moments of the clusters based on the weights given by the updated responsibilities.

Motivation for the update rules in the EM algorithm. We first claim that the update rules (9.4) correspond to the maximization problem

$$\begin{aligned} &\max_{\theta \in \Theta} \left\{ \sum_{i=1}^{n} \sum_{k=1}^{K} r_{i,k}^{(t)} \left(\log \pi_k + \log \mathscr{G}_{\mu_k, \Sigma_k}(x_i) \right) \right\} \\ &\quad = \max_{\theta \in \Theta} \left\{ \sum_{i=1}^{n} \sum_{k=1}^{K} r_{i,k}^{(t)} \left(\log \pi_k - \frac{1}{2} (x_i - \mu_k)^\top \Sigma_k^{-1} (x_i - \mu_k) - \frac{1}{2} \log \det(\Sigma_k) \right) \right\}; \end{aligned} \tag{9.5}$$

and next motivate why this maximization problem is considered instead of (9.3). We assume that the maximization problem is well posed, with a maximizer $\theta^{t+1} \in \Theta$.

Exercise 9.5. *Prove that* θ^{t+1} *defined in* (9.4) *is the unique critical point of* (9.5).

Let us now discuss why the maximization problem (9.5) is considered, following the presentation in [41, Section 8.7.2]. We start by rewriting the log-likelihood of the data points by summing over the cluster assignements for conditional probabilities:

$$\mathscr{L}_n(\theta) = \sum_{i=1}^n \log p_\theta(x_i) = \sum_{i=1}^n \log \left(\sum_{k=1}^K p_\theta(x_i, z_i = k) \right) = \sum_{i=1}^n \log \left(\sum_{k=1}^K q_{i,k} \frac{p_\theta(x_i, z_i = k)}{q_{i,k}} \right),$$

where we introduced a discrete probability distribution $(q_{i,1}, \dots, q_{i,K})$ with non zero entries for all $1 \leqslant i \leqslant n$. Using Jensen's inequality and the concavity of the log,

$$\mathscr{L}_n(\theta) \geqslant \sum_{i=1}^n \sum_{k=1}^K q_{i,k} \log \left(\frac{p_\theta(x_i, z_i = k)}{q_{i,k}} \right) := \sum_{i=1}^n \mathcal{E}(\theta, q_i \,|\, x_i). \tag{9.6}$$

The right hand side of the previous inequality is called an "evidence lower bound" (ELBO), as it provides a lower bound to the negative log-likelihood. Moreover, the terms which appear in the ELBO can be rewritten as

$$\begin{aligned} \mathcal{E}(\theta, q_i \,|\, x_i) &= \sum_{k=1}^K q_{i,k} \log \left(\frac{p_\theta(z_i = k \,|\, x_i) p_\theta(x_i)}{q_{i,k}} \right) \\ &= \sum_{k=1}^K q_{i,k} \log \left(\frac{p_\theta(z_i = k \,|\, x_i)}{q_{i,k}} \right) + \left(\sum_{k=1}^K q_{i,k} \right) \log p_\theta(x_i) \\ &= -\mathrm{KL}(q_i \,|\, p_\theta(z_i \,|\, x_i)) + \log p_\theta(x_i), \end{aligned} \tag{9.7}$$

where

$$\mathrm{KL}(q \,|\, p) = \sum_{k=1}^K q_k \log \frac{q_k}{p_k} \geqslant 0$$

is the Kullback–Leibler entropy between two discrete probability distributions. The quantity (9.7) is maximized by setting $q_{i,k} = p_\theta(z_i = k \,|\, x_i)$, in view of Exercise 9.6 below. This corresponds to the expectation step. On the other hand, the maximization step is recovered by maximizing the lower bound (9.6) with respect to θ. Overall, denoting by

$$\mathscr{P}_K = \left\{ p \in [0,1]^K \;\middle|\; \sum_{k=1}^K p_k = 1 \right\}$$

the set of discrete probabilities over a finite set of cardinality K, the EM algorithm can then be rewritten as

$$\begin{aligned} q_i^{(t)} &= \operatorname*{argmax}_{q_i \in \mathscr{P}_K} \mathcal{E}\left(\theta^t, q_i \,\middle|\, x_i\right) = p_{\theta^t}(z_i \,|\, x_i) = r_i^{(t)}, \\ \theta^{t+1} &= \operatorname*{argmax}_{\theta \in \Theta} \left\{ \sum_{i=1}^n \mathcal{E}\left(\theta, q_i^{(t)} \,\middle|\, x_i\right) \right\} = \operatorname*{argmax}_{\theta \in \Theta} \left\{ \sum_{i=1}^n \sum_{k=1}^K r_{i,k}^{(t)} \log p_\theta(x_i, z_i = k) \right\}, \end{aligned}$$

the latter maximization problem being (9.5) since $p_\theta(x_i, z_i = k) = \pi_k \mathscr{G}_{\mu_k, \Sigma_k}$. In addition, by (9.7),

$$\sum_{i=1}^n \mathcal{E}\left(\theta^t, q_i^{(t)} \,\middle|\, x_i\right) = \sum_{i=1}^n \log p_{\theta^t}(x_i) = \mathscr{L}_n\left(\theta^t\right)$$

is the log-likelihood to be maximized. Therefore, by construction of the algorithm,

$$\mathscr{L}_n\left(\theta^t\right) \leqslant \sum_{i=1}^n \mathcal{E}\left(\theta^{t+1}, q_i^{(t)} \,\middle|\, x_i\right) \leqslant \sum_{i=1}^n \mathcal{E}\left(\theta^{t+1}, q_i^{(t+1)} \,\middle|\, x_i\right) = \mathscr{L}_n\left(\theta^{t+1}\right),$$

so that the log-likelihood is indeed non decreasing as claimed.

Exercise 9.6. *Consider two discrete probability distributions $q, p \in \mathscr{P}_K$. Prove that* $\mathrm{KL}(q \,|\, p) \geqslant 0$, *and* $\mathrm{KL}(q \,|\, p) = 0$ *if and only if $q = p$. It is useful to this end to first prove the identity*

$$\forall z \in [0, +\infty), \qquad z \log z - z + 1 \geqslant 0,$$

with equality if and only if $z = 1$.

(Soft) *K*-means. The cluster assignement

$$z_i = \operatorname*{argmax}_{1 \leqslant k \leqslant K} r_{i,k}^\infty,$$

with r_i^∞ the limiting responsibility obtained from the EM algorithm, allows to fall back to hard clustering. One can consider some soft clustering by using the vector $r_i^\infty \in [0,1]^K$, which provides the probabilities to be in one of the clusters. In any case, one benefit of Gaussian mixture clustering is that anisotropic shapes for the clusters are allowed thanks to the covariance matrices Σ_k^∞. The standard K-means algorithm corresponds to hard assignements r_i for which a single component is equal to 1 (at each step of the procedure), with parameters $\pi_k = 1/K$ and $\Sigma_k = \mathrm{Id}_d$, so that the only quantities to estimate are the cluster centers μ_k (see [41, Section 21.4.1.1]).

9.5 Density based clustering

The presentation in this section is inspired by [39, Section 13.1.3]. The idea behind density based clustering is that clusters are defined by regions of space with a higher density of data points; outliers being in regions of low density. These methods therefore allow to identify clusters and the associated points, but also outliers, which should not be assigned to any cluster.

The most famous density based clustering algorithm is DBSCAN [18] (but other methods exist, see for instance the review in [57, Section 2.2]). Its computational cost scales as $\mathrm{O}(n \log n)$ and it can therefore be used for large data sets. It relies on ε-neighborhoods of data points, defined as

$$\mathcal{N}_\varepsilon(x_i) = \{x \in \mathcal{D} \,|\, \mathtt{d}(x, x_i) < \varepsilon\}.$$

In addition to the parameter $\varepsilon > 0$, the method uses the integer $N_{\min}$, which is the minimal number of neighbors needed for a data point to be considered a "core point". This parameter should be chosen depending on the size of the smallest cluster one expects. A point is said to be "density reachable" if it is in the neighborhood of a core point.

The precise algorithm is the following: until all points have been visited,

- pick a data point x which has not been visited, determine $\mathcal{N}_\varepsilon(x)$ and mark the point as visited;
- if x is a core point (*i.e.* the cardinality of $\mathcal{N}_\varepsilon(x)$ is larger or equal to $N_{\min}$),
  - find the set C of all points that are density reachable from x;
  - C forms a cluster; mark all points in it as visited.

The algorithm then returns the cluster assignements $C_1, \ldots, C_K$. Note that K is not fixed in advance in this method. Note also that some points are not assigned to a cluster, and are therefore considered as noise or outliers. Each cluster contains at least one core point. Non-core points form the "edge" of a cluster. Overall, clusters are composed of one or several core points and all points (core or non-core) which are density reachable from these core points.

9.6 Spectral clustering

The presentation in this section is based on the review article [56]. Spectral clustering starts by constructing a similarity graph from the data points, by considering data points as vertices, linked by edges with weights $W_{ij} = s(x_i, x_j)$ for some similarity function $s : \mathcal{X}^2 \to \mathbb{R}_+$. A classical choice is

$$W_{ij} = \exp\left(-\frac{1}{2\sigma^2}\|x_i - x_j\|_2^2\right),$$

or more generally $\mathtt{d}(x_i, x_j)$ for some distance function. The aim of spectral clustering is to find a partition of the graph such that

- edges between different groups have low weights;
- edges within a group have large weights.

Graph cuts. A first attempt to partition the graph is to consider the first objective above, which leads to the so-called "mincut" problem:

$$\mathrm{Cut}(C_1, \dots, C_K) = \sum_{k=1}^{K} W\left(C_k, \overline{C}_k\right), \qquad W(A, B) = \sum_{\substack{a \in A \\ b \in B}} W_{ab},$$

where $\overline{C}_k = \mathcal{D} \setminus C_k$ is the complement of the data points C_k in the training data set. The mincut problem therefore minimizes the weight of connections between nodes of a given cluster and all other clusters. In practice, it often separates individual vertices from the rest of the graph. Think for instance of the case of a set evenly split into two subsets $C, \overline{C}$ of sizes $n/2$: in this case, $W(C, \overline{C})$ involves a summation over $n^2/4$ terms, while $W(\mathcal{D} \setminus \{x_i\}, \{x_i\})$ involves a summation over $n - 1$ terms only.

A simple fix is to normalize the cut in some way, by a term somewhat proportional to the size of the sets, so that the sums in the example above are of the same order of magnitude:

- unnormalized spectral clustering considers

$$\mathrm{RatioCut}(C_1, \dots, C_K) = \sum_{k=1}^{K} \frac{1}{|C_k|} W\left(C_k, \overline{C}_k\right);$$

- normalized spectral clustering considers

$$\mathrm{NCut}(C_1, \dots, C_K) = \sum_{k=1}^{K} \frac{1}{\mathrm{vol}(C_k)} W\left(C_k, \overline{C}_k\right), \qquad \mathrm{vol}(A) = \sum_{a \in A} d_a, \quad d_a = \sum_{i=1}^{n} W_{ai};$$

Normalized spectral clustering explicitly aims at maximizing the within cluster similarity $W(C_k, C_k)$ since

$$W(C_k, C_k) = \mathrm{vol}(C_k) - W\left(C_k, \overline{C}_k\right),$$

the first term being maximized by the objective function while the second term is minimized. On the other hand, unnormalized spectral clustering emphasizes larger clusters, but these may contain many connections of low weights. See [56, Section 8.5.1] for a more detailed discussion of this point.

Relaxed graph cuts. It is helpful in fact to solve relaxed versions of the minimization problems NCut and RatioCut: instead of assigning points to a cluster, *i.e.* finding $c_i \in \{0, 1\}^K$ with $c_{ik} = 1$ if and only if $x_i \in C_k$ (in particular, $c_{ik} = 0$ if $x_i \notin C_k$), one solves a simpler linear algebra problem; see [56, Sections 5.3 and 5.4] and Exercise 9.8 below for a precise discussion on how this relaxation is performed.

A useful matrix to consider for the relaxed problem is $L = D - W$, where D (the degree matrix) is a diagonal matrix with entries $D_{ii} = d_i$, the total weight of the edges connected to x_i. The following result is fundamental to the spectral approach to clustering (see [56, Proposition 2] and [41, Theorem 21.5.1]).

Proposition 9.1. *The multiplicity K of the eigenvalue 0 of L is equal to the number of connected components $C_1, \dots, C_K$ of the graph, and the associated eigenvectors are proportional to $\mathbf{1}_{C_1}, \dots, \mathbf{1}_{C_K}$.*

Exercise 9.7. *The aim of this exercise is to prove Exercise 9.1.*

(a) Prove that L is positive semidefinite, and that its spectrum is contained in $[0, +\infty)$.
(b) Show that

$$\sigma(L) = \bigcup_{k=1}^{K} \sigma(L_k).$$

where the matrices L_k are the block diagonal components of L (possibly upon reordering points).
(c) Conclude.

The above result suggests by a perturbative analysis that, when the connected components are weakly linked, the spectrum should consist of K small eigenvalues and then a spectral gap between the $(K+1)$-th and the K-th eigenvalues. This remark allows to find the relevant number of clusters K in practice by diagonalizing L and locating the spectral gap.

Moreover, the eigenvectors associated with these eigenvalues should be almost constant on the connected components (*i.e.* have values close to 1 on one of the connected components, and close to 0 on the others, when normalized). A practical algorithm to find these eigenvectors, and hence identify the clusters, is to perform K-means on the lines of the matrix

$$\mathcal{U} = \begin{pmatrix} | & | & & | \\ U_1 & U_2 & \dots & U_K \\ | & | & & | \end{pmatrix} \in \mathbb{R}^{n \times K},$$

where $(U_k)_{1 \leqslant k \leqslant K}$ are the first (normalized) K eigenvectors of L. This corresponds to the relaxed version of RatioCut. Some authors suggest to normalize L as $L_{\mathrm{sym}} = D^{-1/2} L D^{-1/2}$ in order to take into account that some clusters are more connected to others (see [41, Section 21.5.2]) and perform K-means clustering on the first K eigenvectors of L_{sym}. This corresponds to the relaxed version of NCut. There is no consensus on which approach to prefer, see the discussions in [56, Section 8.5.3].

Exercise 9.8. *We make precise in this exercise in which sense the spectral problem on the graph Laplacian can be understood as a relaxed version of graph cut problems.*

(a) We first consider RatioCut for $K = 2$, i.e. *consider the partition $(C, \overline{C})$. Introduce the vector $h \in \mathbb{R}^n$ with components*

$$h_i = \sqrt{\frac{|\overline{C}|}{|C|}} \mathbf{1}_{x_i \in C} - \sqrt{\frac{|C|}{|\overline{C}|}} \mathbf{1}_{x_i \in \overline{C}}.$$

Denoting by $\mathbf{1}_n \in \mathbb{R}^n$ the vector for which all components are equal to 1, prove that

$$h \cdot \mathbf{1}_n = 0, \qquad h^\top h = n, \qquad \mathrm{RatioCut}\left(C, \overline{C}\right) = \frac{1}{n} h^\top L h.$$

Deduce that

$$\mathrm{RatioCut}\left(C, \overline{C}\right) \geqslant \min_{\substack{\xi \in \mathbb{R}^n \setminus \{0\} \\ \xi \cdot \mathbf{1}_n = 0}} \frac{\xi^\top L \xi}{\xi^\top \xi}.$$

What is the minimizer of the right hand side?
(b) We next consider RatioCut for $K \geqslant 2$. Define the matrix $H \in \mathbb{R}^{n \times K}$ with entries

$$H_{ik} = \frac{1}{\sqrt{|C_k|}} \mathbf{1}_{x_i \in C_k}.$$

Prove that $H^\top H = \mathrm{Id}_K$ *and that*

$$\frac{1}{|C_k|}\mathrm{Cut}\left(C_k, \overline{C}_k\right) = \left(H^\top L H\right)_{kk}.$$

Deduce that

$$\mathrm{RatioCut}(C_1, \ldots, C_K) \geqslant \min_{H \in \mathbb{R}^{n\times K}} \left\{ \mathrm{Tr}\left(H^\top L H\right) \;\middle|\; H^\top H = \mathrm{Id}_K \right\}.$$

What is the minimizer of the right hand side?

(c) *We finally consider NCut. By proceeding as in the previous question, and introducing the matrix* $\widetilde{H} \in \mathbb{R}^{n\times K}$ *with entries*

$$\widetilde{H}_{ik} = \frac{1}{\sqrt{\mathrm{vol}(C_k)}} \mathbf{1}_{x_i \in C_k},$$

show that the relaxed version of NCut is

$$\min_{H \in \mathbb{R}^{n\times K}} \left\{ \mathrm{Tr}\left(H^\top L_{\mathrm{sym}} H\right) \;\middle|\; H^\top H = \mathrm{Id}_K \right\}.$$

10

Complements: Elements of statistical learning

This chapter gathers additional material which is mentioned at various places in the lecture notes: guarantees on empirical risk minimization in Section 10.1, elements on model selection in Section 10.2, and some key facts in Statistical Learning Theory in Section 10.3.

10.1 Empirical risk minimization and statistical learning theory

We present here a brief introduction to some theoretical aspects of empirical risk minimization for supervised learning, in particular estimates motivating that this approach provides good predictors. The material in this section is taken from [4, Sections 2.3 to 2.5] as well as selected parts of Chapter 4 of this reference, [41, Section 5.4] and [40, Sections 4.1 and 4.2].

10.1.1 Framework

Let us briefly recall here the setting of empirical risk minimization. We consider a parametrized family of predictors $f_\theta : \mathcal{X} \to \mathcal{Y}$ for $\theta \in \Theta$. In practice, predictors can be obtained from data by minimizing with respect to θ the empirical risk

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

where $\mathcal{D}_n = \{(x_1, y_1), \dots, (x_n, y_n)\}$ is a dataset of i.i.d. pairs (x_i, y_i) sampled with respect to some unknown distribution p_{data} (see for instance (2.1) and specific formulations such as (2.3), or (8.2), to give just two examples). The minimization of such empirical risks can sometimes be easy (as for linear regression problems, for which the optimal value of the parameter has an analytic expression; or for logistic regression, for which the empirical risk is convex), and can in any cases be considered for parameters of arbitrary dimensions. However, the minimization can also be quite difficult, for instance for non-convex empirical risks (as those encountered for supervised learning with neural networks), or when the gradient with respect to θ is complex or expensive to compute. Moreover, classification requires parametrizing functions with values in $\{0, 1\}$ (and/or convexifying the loss, as discussed in Section 3.1), and in general one needs capacity control to avoid overfitting.

10.1.2 Decomposition of the risk minimization

The aim of this section is to provide estimates on the quality of the predictor found by empirical risk minimization. We decompose to this end the excess risk into an estimation error and an approximation error. We consider for this a family $\mathcal{F}$ of predictor functions $\mathcal{X} \to \mathcal{Y}$ (for instance $\{f_\theta, \theta \in \Theta\}$ with Θ belonging to $\mathbb{R}^d$ or to some ball in order to have some capacity control). The predictor found by the empirical risk minimization is

$$\widehat{f} \in \operatorname*{argmin}_{f \in \mathcal{F}} \widehat{\mathcal{R}}_n(f). \tag{10.1}$$

The question is how large the excess risk $\mathcal{R}(\widehat{f}) - \mathcal{R}^\star$ is. We rewrite the latter quantity as

$$\underbrace{\mathcal{R}(\widehat{f}) - \mathcal{R}^\star}_{\text{excess risk}} = \underbrace{\mathcal{R}(\widehat{f}) - \inf_{f' \in \mathcal{F}} \mathcal{R}(f')}_{\text{estimation error}} + \underbrace{\inf_{f' \in \mathcal{F}} \mathcal{R}(f') - \mathcal{R}^\star}_{\text{approximation error}}. \tag{10.2}$$

Let us discuss the two errors on the right hand side of the previous equality:

- The *approximation error* is a deterministic quantity, which depends on $\mathcal{F}$ and the underlying (unknown) distribution p_{data}. It quantifies how well the Bayes predictor $f^\star$ can be represented in $\mathcal{F}$. The idea is that the larger $\mathcal{F}$ is, the smaller the approximation error is. Precise convergence rates depend on the regularity of $f^\star$ (which is a classical result in analysis and approximation theory).
- The *estimation error* is a random quantity, which depends implicitly on the data set through the minimizer $\widehat{f}$ of the empirical risk (see (10.1)). This error is not easy to quantify as such. A key idea here is to upper bound it in terms of the supremum over $f \in \mathcal{F}$ of the differences $|\mathcal{R}(f) - \widehat{\mathcal{R}}_n(f)|$, as made precise in Proposition 10.1.

Proposition 10.1 (Bound on the estimation error). *For any data set $\mathcal{D}_n$, it holds*

$$0 \leqslant \mathcal{R}(\widehat{f}) - \inf_{f \in \mathcal{F}} \mathcal{R}(f) \leqslant 2 \sup_{f \in \mathcal{F}} \left| \widehat{\mathcal{R}}_n(f) - \mathcal{R}(f) \right|.$$

The main interest of this result is that it is more convenient to work with $\widehat{\mathcal{R}}_n$ than $\widehat{f}$. Let us emphasize that the upper bound is a random quantity (depending on the data set). This random quantity usually decays with n (with $\widehat{\mathcal{R}}_n(f)$ converging almost surely to $\mathcal{R}(f)$ for f given).

Proof. Assume that $\inf_{f \in \mathcal{F}} \mathcal{R}(f) = \mathcal{R}(g_\mathcal{F})$ is attained for some $g_\mathcal{F} \in \mathcal{F}$ (otherwise the argument below needs to be modified as in the proof of [40, Proposition 4.1]). Then,

$$\begin{aligned}
\mathcal{R}(\widehat{f}) - \inf_{f \in \mathcal{F}} \mathcal{R}(f) &= \mathcal{R}(\widehat{f}) - \mathcal{R}(g_\mathcal{F}) \\
&= \left(\mathcal{R}(\widehat{f}) - \widehat{\mathcal{R}}_n(\widehat{f})\right) + \underbrace{\widehat{\mathcal{R}}_n(\widehat{f}) - \widehat{\mathcal{R}}_n(g_\mathcal{F})}_{\leqslant 0} + \left(\widehat{\mathcal{R}}_n(g_\mathcal{F}) - \mathcal{R}(g_\mathcal{F})\right) \\
&\leqslant 2 \sup_{f \in \mathcal{F}} \left| \mathcal{R}(f) - \widehat{\mathcal{R}}_n(f) \right|,
\end{aligned}$$

which is the claimed result. □

Let us give a few comments on the above result:

- the statistical dependence between $\widehat{f}$ and $\widehat{\mathcal{R}}_n$ is removed at the end of the proof by considering a uniform bound over all possible functions in $\mathcal{F}$. This uniform deviation bound typically grows as $\mathcal{F}$ becomes larger;
- it would be possible to take into account an additional optimization error (quantifying the fact that the minimizer of the empirical risk is obtained by a numerical procedure) by writing that $\widehat{\mathcal{R}}_n(\widehat{f}) - \widehat{\mathcal{R}}_n(g_\mathcal{F}) \leqslant \varepsilon$ instead of saying that this quantity is nonnegative.

In order to instantiate the bound on the estimation error provided by Proposition 10.1, we consider the simple setting when $\mathcal{F}$ is a set of finite cardinality $|\mathcal{F}| < +\infty$, and the elementary loss ℓ is bounded.

Proposition 10.2. *Suppose that $\mathcal{F}$ is finite, and the elementary loss ℓ is bounded: there exists $\mathsf{L} < +\infty$ such that*

$$\forall (y,z) \in \mathcal{Y}^2, \qquad 0 \leqslant \ell(y,z) \leqslant \mathsf{L} < +\infty.$$

Then, for any $\delta \in (0,1)$, the following bound holds with probability larger than $1-\delta$:

$$0 \leqslant \mathcal{R}(\widehat{f}) - \inf_{f \in \mathcal{F}} \mathcal{R}(f) \leqslant \mathsf{L}\sqrt{\frac{2}{n}\left(\log|\mathcal{F}| + \log\frac{2}{\delta}\right)}. \tag{10.3}$$

This result, which is based on Hoeffding's inequality, is proved in Section 10.1.3. Let us make a few comments on it:

- the error is "with high probability": the upper bound becomes larger as the probability $1-\delta$ comes closer to 1;
- the decay rate of the upper bound is $1/\sqrt{n}$, consistent with the scaling $\widehat{\mathcal{R}}(f) - \mathcal{R}(f) = \mathrm{O}(n^{-1/2})$ obtained from the central limit theorem;
- the size $|\mathcal{F}|$ of the set of functions for the approximation appears only logarithmically. The result can be extended to set which are not discrete, under some conditions, in which case the term $\log|\mathcal{F}|$ should be replaced by the dimension of the set.

The result on the estimation error of Propositions 10.1 and 10.2, combined with the decomposition of the excess risk (10.2), suggests that a trade-off should be reached in the complexity of $\mathcal{F}$: the approximation error is smaller for $\mathcal{F}$ larger, while the estimation error increases in this case; and conversely. The increase of the estimation error is related to the generalization capability of the model: a model with a small estimation error generalizes better.

A typical scenario, as the size of $\mathcal{F}$ (or the set Θ of parameters for the function f_θ in the set $\mathcal{F}$) increases, is the following:

- for small sizes of $\mathcal{F}$, the train and validation (test) errors are both large. This corresponds to the regime of *underfitting*, where the estimation error is small but the approximation error is large ("large bias but small variance");
- for large sizes of $\mathcal{F}$, the train error is small but the validation error is large. This corresponds to the regime of *overfitting*, where the approximation error is small but the estimation error is large since the predictor learnt from the train dataset fits too closely the training data and hence generalizes poorly ("small bias but large variance");

10.1.3 Proof of Proposition 10.2

A key tool in the proof of Proposition 10.2 is Hoeffding's inequality (proved in Exercise 10.4 below).

Proposition 10.3 (Hoeffding's inequality). *If $Z_1, \dots Z_n$ are independent random variables such that $Z_i \in [0,1]$ almost surely, then, for any $t \geqslant 0$,*

$$\mathbb{P}\left(\frac{1}{n}\sum_{i=1}^n Z_i - \frac{1}{n}\sum_{i=1}^n \mathbb{E}[Z_i] \geqslant t\right) \leqslant \mathrm{e}^{-2nt^2}. \tag{10.4}$$

Note that the random variables Z_i need not be identically distributed. The result above can be considered as a quantitative version of the Law of Large Numbers.

Exercise 10.1. *Under the same assumptions as in Proposition 10.3, prove that*

$$\mathbb{P}\left(\left|\frac{1}{n}\sum_{i=1}^n Z_i - \frac{1}{n}\sum_{i=1}^n \mathbb{E}[Z_i]\right| \geqslant t\right) \leqslant 2\mathrm{e}^{-2nt^2}.$$

Remark 10.1. *The bound* (10.4) *can be compared with the asymptotic bound given by the Central Limit Theorem when the random variables Z_i are i.i.d. Indeed, as $n \to +\infty$,*

$$\lim_{n\to+\infty} \mathbb{P}\left(\frac{1}{n}\sum_{i=1}^{n} Z_i - \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}[Z_i] \geqslant \frac{t}{\sqrt{n}}\right) = \sqrt{\frac{1}{2\pi\sigma^2}}\int_t^{+\infty} \mathrm{e}^{-z^2/(2\sigma^2)}\,dz, \tag{10.5}$$

for $\sigma^2 = \mathrm{Var}(Z_1)$. The integral in the right-hand side of the above equality can be upper bounded as

$$\begin{aligned}\int_t^{+\infty} \mathrm{e}^{-z^2/(2\sigma^2)}\,dz &= \mathrm{e}^{-t^2/(2\sigma^2)}\int_0^{+\infty} \mathrm{e}^{-y^2/(2\sigma^2)}\mathrm{e}^{-ty/\sigma^2}\,dy \leqslant \mathrm{e}^{-t^2/(2\sigma^2)}\int_0^{+\infty} \mathrm{e}^{-y^2/(2\sigma^2)}\,dy \\ &= \frac{\sqrt{2\pi\sigma^2}}{2}\mathrm{e}^{-t^2/(2\sigma^2)},\end{aligned}$$

so that the right-hand side of (10.5) *is smaller than $\mathrm{e}^{-t^2/(2\sigma^2)}/2$. Now, random variables with values in $[0,1]$ have a variance smaller than $1/4$ (see Exercise 10.2), so that $\mathrm{e}^{-t^2/(2\sigma^2)}/2 \leqslant \mathrm{e}^{-2t^2}/2$, which is itself smaller than the bound e^{-2t^2} obtained from* (10.4)*. The result* (10.5) *is however only an asymptotic result, whereas* (10.4) *holds for any $n \geqslant 1$.*

Exercise 10.2. *Consider a random variable Y which takes almost surely values in $[0,1]$. Prove that $\mathrm{Var}(Y) \leqslant 1/4$.*

Exercise 10.3. *Which bounds holds instead of* (10.4) *when $Z_i \in [a,b]$ almost surely with $-\infty < a < b < +\infty$?*

We are now in position to prove Proposition 10.2 with Hoeffding's inequality. For $t \geqslant 0$, using the upper bound on the estimation error provided by Proposition 10.1, then the union bound,

$$\begin{aligned}\mathbb{P}\left(\mathcal{R}(\widehat{f}) - \inf_{f\in\mathcal{F}}\mathcal{R}(f) \geqslant t\right) &\leqslant \mathbb{P}\left(2\sup_{f\in\mathcal{F}}\left|\widehat{\mathcal{R}}_n(f) - \mathcal{R}(f)\right| \geqslant t\right) \\ &\leqslant \sum_{f\in\mathcal{F}}\mathbb{P}\left(2\left|\widehat{\mathcal{R}}_n(f) - \mathcal{R}(f)\right| \geqslant t\right).\end{aligned}$$

For f given, it holds, in view of Exercises 10.3 and 10.1 and since $Z_i = \ell(f(x_i), y_i) \in [0, \mathsf{L}]$,

$$\mathbb{P}\left(2\left|\widehat{\mathcal{R}}_n(f) - \mathcal{R}(f)\right| \geqslant t\right) \leqslant 2\exp\left(-\frac{2n}{\mathsf{L}^2}\left(\frac{t}{2}\right)^2\right).$$

Therefore,

$$\mathbb{P}\left(\mathcal{R}(\widehat{f}) - \inf_{f\in\mathcal{F}}\mathcal{R}(f) \geqslant t\right) \leqslant 2|\mathcal{F}|\mathrm{e}^{-nt^2/(2\mathsf{L}^2)}.$$

We set $\delta = 2|\mathcal{F}|\mathrm{e}^{-nt^2/(2\mathsf{L}^2)}$, *i.e.*

$$t = \mathsf{L}\sqrt{\frac{2}{n}\log\left(\frac{2|\mathcal{F}|}{\delta}\right)},$$

from which the result follows.

Exercise 10.4 (Proof of Hoeffding's inequality). *Consider a random variable Z which almost surely has values in $[0,1]$, and introduce*

$$\varphi(s) = \log\mathbb{E}\left[\mathrm{e}^{s(Z-\mathbb{E}(Z))}\right].$$

We start by proving that $\mathrm{e}^{\varphi(s)} \leqslant \mathrm{e}^{s^2/8}$ for any $s \geqslant 0$, and then derive the claimed bound.

(1) Compute $\varphi'(s)$ and $\varphi''(s)$, and prove that $\varphi''(s) \geqslant 0$ for any $s \geqslant 0$.

(2) Show that $\varphi''(s) \leqslant 1/4$ by relying on Exercise 10.2, and deduce that

$$\mathbb{E}\left[\mathrm{e}^{s(Z-\mathbb{E}(Z))}\right] \leqslant \mathrm{e}^{s^2/8}.$$

(3) Use Markov's inequality $P(X \geqslant a) \leqslant \mathbb{E}(X)/a$ for a nonnegative random variable X and $a > 0$ to prove

$$\mathbb{P}\left(\frac{1}{n}\sum_{i=1}^{n} Z_i - \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}[Z_i] \geqslant t\right) \leqslant \mathrm{e}^{-st+s^2/(8n)}.$$

Conclude.

10.2 Model selection

This section discusses the mathematical foundations behind the techniques to estimate the expected risk and choose hyperparameters of models. More precisely, we rely on model selection, whose aim is to find the right balance between the estimation and approximation errors. We present the approach when looking for the best hyperparameters (*e.g.* the number of neighbors for KNN), but this can be readily generalized to choosing a regularization strength (which corresponds to the so-called structural risk minimization). The presentation is based on [50, Chapter 11] and [40, Sections 4.4 and 4.5].

10.2.1 Validation

The principle of cross-validation is introduced in Section 1.3.2, when studying KNN. It corresponds to using some part of the training set as validation set to select hyperparameters. To formalize the discussion, introduce a dataset $\mathcal{D} = \{(x_1, y_1), \dots, (x_n, y_n)\}$ is a dataset of i.i.d. pairs (x_i, y_i) sampled with respect to some unknown probability distribution. This dataset is randomly decomposed into a training data set $\mathcal{D}_{\text{train}}$ of size $(1-\alpha)n$ and a validation set $\mathcal{D}_{\text{val}}$ of size αn, for some fraction $\alpha \in [0,1]$ (chosen such that αn is an integer; a typical value would be $\alpha = 0.2$). This is equivalent to independently sampling a training and validation set of i.i.d. data points, of respective sizes $(1-\alpha)n$ and αn.

Proposition 10.4. *Assume that the elementary loss function is bounded, namely $0 \leqslant \ell(y,z) \leqslant \mathsf{L} < +\infty$ for any $y, z \in \mathcal{Y}$. Then,*

$$\mathbb{P}\left(\left|\mathcal{R}(\widehat{f}_{\mathcal{D}_{\text{train}}}) - \widehat{\mathcal{R}}_{\text{val}}(\widehat{f}_{\mathcal{D}_{\text{train}}})\right| \geqslant \varepsilon\right) \leqslant 2\mathrm{e}^{-2\alpha n\varepsilon^2/\mathsf{L}^2}, \tag{10.6}$$

where $\widehat{f}_{\mathcal{D}_{\text{train}}}$ is a minimizer of the training loss (defined on $\mathcal{D}_{\text{train}}$). Therefore, with probability larger that $1-\delta$ over the choice of the train and validation sets,

$$\left|\mathcal{R}(\widehat{f}_{\mathcal{D}_{\text{train}}}) - \widehat{\mathcal{R}}_{\text{val}}(\widehat{f}_{\mathcal{D}_{\text{train}}})\right| \leqslant \mathsf{L}\sqrt{\frac{1}{2\alpha n}\log\left(\frac{2}{\delta}\right)}. \tag{10.7}$$

Remark 10.2. *Note that the bound* (10.7) *is tighter than bounds such as* (10.3) *(in the sense that it involves a single term with $\log(2/\delta)$) because a "fresh" sample, independent of the training set, is considered.*

Proof. The first inequality is obtained by fixing $\widehat{f}_{\mathcal{D}_{\text{train}}}$ (which is determined by the training set) and then using Hoeffding's inequality (10.4) with respect to realizations of the validation set. The choice

$$\delta = 2\mathrm{e}^{-2\alpha n\varepsilon^2/\mathsf{L}^2},$$

that is

$$\varepsilon = \mathsf{L}\sqrt{\frac{1}{2\alpha n}\log\left(\frac{2}{\delta}\right)},$$

directly leads to the second inequality. □

10.2.2 Model selection with validation

Consider the training of models on a training set $\mathcal{D}_{\text{train}}$ of size $(1-\alpha)n$, for r different choices of hyperparameters (*e.g.* order of polynomials for polynomial regression, topology of neural networks, etc), henceforth leading to associated predictors $\widehat{f}_1, \dots, \widehat{f}_r$. A natural question is to choose a single predictor out of these r candidates, to be used on the test set.

A standard procedure is to characterize the performance of the prediction on a validation set $\mathcal{D}_{\text{val}}$ of size αn; more precisely, to choose the predictor which minimizes the validation error:

$$\min_{1 \leqslant i \leqslant r} \mathcal{R}_{\text{val}}(\widehat{f}_i),$$

with

$$\mathcal{R}_{\text{val}}(\widehat{f}_i) = \frac{1}{\alpha n} \sum_{(x_j, y_j) \in \mathcal{D}_{\text{val}}} \ell\left(\widehat{f}_i(x_j), y_j\right).$$

This is similar to learning from a finite set $\mathcal{F}$, but this set $\mathcal{F}$ is not fixed ahead in time as it depends on the training dataset $\mathcal{D}_{\text{train}}$. However, the sets $\mathcal{D}_{\text{train}}$ and $\mathcal{D}_{\text{val}}$ are independent, so that $\mathcal{D}_{\text{val}}$ is independent of the set of predictors being considered. We can therefore use the same techniques as the ones used to obtain learning guarantees for finite sets $\mathcal{F}$.

Theorem 10.1. *Consider an arbitrary set of predictors $\mathcal{F} = \{f_1, \dots, f_r\}$ (possibly depending on $\mathcal{D}_{\text{train}}$), and assume that the elementary loss function is bounded, namely $0 \leqslant \ell(y,z) \leqslant \mathsf{L} < +\infty$ for any $y, z \in \mathcal{Y}$. Consider a validation set $\mathcal{D}_{\text{val}}$ of size αn, sampled independently of $\mathcal{F}$. Then, with probability at least $1-\delta$ over the choice of $\mathcal{D}_{\text{val}}$,*

$$\forall h \in \mathcal{F}, \qquad |\mathcal{R}(h) - \mathcal{R}_{\text{val}}(h)| \leqslant \mathsf{L}\sqrt{\frac{1}{2\alpha n} \log\left(\frac{2|\mathcal{F}|}{\delta}\right)}.$$

Proof. The estimate (10.6) for a given element $h \in \mathcal{F}$, combined with the union bound, leads to

$$\mathbb{P}\left(\sup_{h \in \mathcal{F}} \left|\mathcal{R}(h) - \widehat{\mathcal{R}}_{\text{val}}(h)\right| \geqslant \varepsilon\right) \leqslant \sum_{h \in \mathcal{F}} \mathbb{P}\left(\left|\mathcal{R}(h) - \widehat{\mathcal{R}}_{\text{val}}(h)\right| \geqslant \varepsilon\right) \leqslant 2|\mathcal{F}| \mathrm{e}^{-2\alpha n \varepsilon^2 / \mathsf{L}^2},$$

from which the proof is concluded similarly to the proof of Proposition 10.4. □

The interpretation of Theorem 10.1 is that the error on the validation set approximates the true error as long as $\mathcal{F}$ is not too large. If one tries too many methods or values of the parameters (*i.e.* the number r of predictors $\widehat{f}_1, \dots, \widehat{f}_r$ is too large), then there is no good guarantee on the performance of the predictor. This corresponds to a situation of overfitting, where the training loss is typically small while the test loss is large.

In practice, one should avoid testing too many parameters – both because it is dangerous from the above discussion on the interpretation of Theorem 10.1, and also because it is time consuming. To this end, one should

- start with a rough coverage of parameter values (for instance using a coarse logscale for positive parameters);
- refine in a second step the parameter grid in the most relevant regions to fine tune the choice of the parameters;
- potentially, it may be a good idea not to optimize all hyperparameters and models at the same time, and proceed sequentially (for instance, set the hyperparameters of the training procedure in order to have an efficient training for extreme values of the other hyperparameters, and then look for optimal values of these other hyperparameters).

Remark 10.3 (Learning curves). *The behavior of the training and validation losses as a function of the size of the training set is a useful criterion to determine how to improve the learning when the validation error is large, in particular when the training error is small but the validation*

error is large. In essence, does the issue come from n being too small (large estimation error) or the class $\mathcal{F}$ being too small (large approximation error)? When the approximation error is large ($\mathcal{F}$ is too small), the validation error does not decrease much as n increases. On the other hand, when the approximation error is small ($\mathcal{F}$ is large enough), the validation error can be large for n small but then decreases as n is increased since the estimation error decreases, and one expects that the difference between the train and validation errors vanishes as $n \to +\infty$.

10.3 Statistical learning theory

The goal of learning theory is to provide guarantees of performance of unseen data. From a technical viewpoint, this can be described as relating the in-sample and out-of-sample errors. Recall that we work in these lecture notes in the statistical framework: the data points are assumed to be i.i.d. with respect to some unknown distribution $p_{\mathrm{data}}(dx\,dy)$. This corresponds to considering a random set $\mathcal{D}_{\mathrm{train}} = \{(x_1, y_1), \dots, (x_n, y_n)\} := \mathscr{D}_n(p_{\mathrm{data}})$. In this context, an algorithm $\mathscr{A}$ is a mapping which associates to a dataset $\mathscr{D}_n(p_{\mathrm{data}})$ a function $\mathcal{X} \to \mathcal{Y}$. The associated expected risk is $\mathcal{R}_{p_{\mathrm{data}}}(f)$ with $f = \mathscr{A}(\mathscr{D}_n(p_{\mathrm{data}}))$. The aim is to find $\mathscr{A}$ such that

$$\mathcal{R}_{p_{\mathrm{data}}}\left(\mathscr{A}(\mathscr{D}_n(p_{\mathrm{data}}))\right) - \mathcal{R}^\star \tag{10.8}$$

is small. The challenge is that p_{data} is unknown and should be considered as arbitrary (so that ideally one should make minimalistic assumptions on it); and that the the risk is random since $\mathscr{D}_n(p_{\mathrm{data}})$ is random.

Metrics of performance. To quantify the performance, there are two main metrics:

- minimize the *average error*, namely the expectation of (10.8) with respect to realizations of $\mathscr{D}_n(p_{\mathrm{data}})$:
$$\mathbb{E}\left[\mathcal{R}_{p_{\mathrm{data}}}\left(\mathscr{A}(\mathscr{D}_n(p_{\mathrm{data}}))\right)\right] - \mathcal{R}^\star; \tag{10.9}$$
- in the *probably approximately correct* (PAC) framework, one wants to ensure that (10.8) is smaller than ε with probability larger than $1 - \delta$:
$$\mathbb{P}\left[\mathcal{R}_{p_{\mathrm{data}}}\left(\mathscr{A}(\mathscr{D}_n(p_{\mathrm{data}}))\right) - \mathcal{R}^\star \leqslant \varepsilon\right] \geqslant 1 - \delta.$$
Typically, one fixes $\delta \in (0, 1)$ and seeks to minimize $\varepsilon > 0$, or fixes $\varepsilon > 0$ and seeks to minimize $\delta \in (0, 1)$.

An algorithm is consistent in expectation if (10.9) converges to 0 as $n \to +\infty$. It is PAC consistent if, for any $\varepsilon > 0$, the inequality (10.9) holds for any $n \geqslant 1$ with $\delta_n \to 0$ as $n \to +\infty$.

Consistency. Consistency can also be assessed over classes of problems, with an analysis that can be asymptotic or not. An algorithm is said to be universally consistent if it is consistent in expectation for all distributions p_{data} (but the rate of convergence will generically depend on p_{data} in view of the "no free lunch theorem"; see Theorem 10.2 below). Consistency can also be established for classes of distributions with some regularity properties (for instance with compact support, or leading to a Bayes predictor which is Lipschitz, etc). For such a given class $\mathscr{P}$, this amounts to finding an algorithm $\mathscr{A}$ such that

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

is as small as possible.

One can also consider minimax risks, which correspond to taking infima over $\mathscr{A}$ in the previous metrics of convergence. Upper bounds on the minimax risk can be derived from the results obtained by studying one particular algorithm. Lower bounds are more difficult to establish.

No free lunch theorems. The spirit of various results going under the name "no free lunch" theorem is that there is no algorithm that works optimally for all distributions – *i.e.* learning is not possible without assumptions. We present here one possible result, which shows that, for any algorithm and any fixed number of samples $n \geqslant 1$, there is a probability distribution which makes the algorithm useless – *i.e.* no better than guessing at random ("chance level").

Theorem 10.2 (No free lunch). *Consider binary classification with the elementary loss function $\ell(y,z) = \mathbf{1}_{y \neq z}$ and a space of inputs $\mathcal{X}$ of infinite cardinality ($|\mathcal{X}| = +\infty$). Denote by $\mathscr{P}$ the set of probability measures on $\mathcal{X} \times \{0,1\}$. Then, for any learning algorithm $\mathscr{A}$ and any integer $n \geqslant 1$,*

$$\sup_{p \in \mathscr{P}} \mathbb{E}\left[\mathcal{R}_p\left(\mathscr{A}(\mathscr{D}_n(p))\right)\right] - \mathcal{R}^\star \geqslant \frac{1}{2}.$$

There are versions of such results with some uniformity in n, for instance statements such as: for any decreasing sequence $(a_n)_{n \geqslant 1}$ with $a_n \to 0$ as $n \to +\infty$, there exists $p_{\mathrm{data}} \in \mathscr{P}$ such that $\mathbb{E}\left[\mathcal{R}_{p_{\mathrm{data}}}\left(\mathscr{A}(\mathscr{D}_n(p_{\mathrm{data}}))\right)\right] - \mathcal{R}^\star \geqslant a_n$.

Proof. We write the result for $\mathcal{X} = \mathbb{N}$. Fix $k \in \mathbb{N}$ (this number will ultimately be sent to infinity). The proof proceeds in two steps: we first construct a probability distribution supported on k elements of $\mathbb{N}$, with $k \gg n$ such that the knowledge of the n labels does not imply doing well on all k elements; and next choose the parameters of the distribution (characterized by the vector r below) by comparing the performance of the algorithm to the one of random guesses.

For the first step, we introduce a vector of labels $r \in \{0,1\}^k$ and define a joint distribution p_r on (x,y) by

$$\forall j \in \{1,\dots,k\}, \quad \mathbb{P}(x=j, y=r_j) = \frac{1}{k}, \qquad \forall j \in \{k+1,\dots,n\}, \quad \mathbb{P}(x=j,\{0,1\}) = 0.$$

This means that x is uniformly distributed over $\{1,\dots,k\}$ with labels r_x deterministically obtained from x. Since the relationship from inputs to labels is deterministic, $\mathcal{R}^\star = 0$.

We next denote by

$$S(r) = \mathbb{E}\left[\mathcal{R}_{p_r}\left(\widehat{f}_{\mathscr{D}_n}\right)\right]$$

the average (over realizations of the data set) of the expected risk, with the predictor

$$\widehat{f}_{\mathscr{D}_n} = \mathscr{A}(\mathscr{D}_n(p_r)).$$

In order to maximize the quantity S with respect to $r \in \{0,1\}^k$, we consider in fact a distribution q of possible elements in $\{0,1\}^k$ (for instance the uniform distribution). The maximum over all elements is larger that the average with respect to q. More precisely,

$$\begin{aligned}
\max_{r \in \{0,1\}^k} S(r) &\geqslant \mathbb{E}_{r \sim q}[S(r)] = \mathbb{P}\left(\widehat{f}_{\mathscr{D}_n}(x) \neq r_x\right) \\
&= \mathbb{E}\left[\mathbb{P}\left(\widehat{f}_{\mathscr{D}_n}(x) \neq r_x \,\middle|\, (x_1,r_1),\dots,(x_n,r_n)\right)\right] \\
&\geqslant \mathbb{E}\left[\mathbb{P}\left(\widehat{f}_{\mathscr{D}_n}(x) \neq r_x \text{ and } x \in \{x_1,\dots,x_n\} \,\middle|\, (x_1,r_1),\dots,(x_n,r_n)\right)\right] \\
&= \frac{1}{2}\mathbb{E}\left[\mathbb{P}\left(x \in \{x_1,\dots,x_n\} \,|\, (x_1,r_1),\dots,(x_n,r_n)\right)\right] \\
&= \frac{1}{2}\prod_{i=1}^n \mathbb{P}(x_i \neq x \,|\, x) = \frac{1}{2}\left(1 - \frac{1}{k}\right)^n.
\end{aligned}$$

where the second line is obtained by conditioning on $\mathscr{D}_n$ and r, so that the conditional probability is over realizations of x; while in the third line the lower bound corresponds to realizations of x not in the data set, for which the label r_x was not observed, and so it has probability $1/2$ to be 0 or 1 when averaging over $r \sim q$. Note that the random variables $x_1,\dots,x_n$ (realizations of the data set $\mathscr{D}_n$), r (realizations of the labels) and x are independent. The desired lower bound is finally obtained in the limit $k \to +\infty$. □

11

Implementing and debuging machine learning programs

We refer for instance to [23, Chapter 11] for various practical advice on how to implement and debug machine learning programs. When predictions on the test set are poor, several options should be considered:

- increase or decrease the model capacity, depending on whether underfitting or overfitting is the issue. This can be achieved by playing with the parameters of the model (for instance increase or decrease the size of a neural network) and/or the regularization used;
- improve the quality of the optimization of the parameters of the model if the training error is large;
- debug the software implementation by looking at the worst errors, for instance wrong predictions obtained with a good confidence (think of a binary classification problem, where the probability to observe a class label would be close to 1 for an incorrect label);
- gather more data and/or clean it up. This can be assessed and quantified with learning curves, which report the predictive performance on the training and test sets as a function of the number of data points, possibly with a logarithmic scale on the number of data points;
- carefully select the hyperparameters of the method with refined grid searches and cross validations.

References

[1] Scikit-learn user guide. https://scikit-learn.org/stable/user_guide.html.

[2] G. Alain et Y. Bengio, What regularized auto-encoders learn from the data-generating distribution, *J. Mach. Learn. Res* **15**(110) (2014) 3743–3773.

[3] E. Alpaydin, *Introduction to Machine Learning*, 3 ed., Adaptive Computation and Machine Learning (MIT Press, 2014).

[4] F. Bach, *Learning Theory from First Principles* (MIT Press, 2023). To appear.

[5] P. Baldi et K. Hornik, Neural networks and principal component analysis: Learning from examples without local minima, *Neural Networks* **2** (1989) 53–58.

[6] D. Barber, *Bayesian Reasoning and Machine Learning* (Cambridge University Press, 2012).

[7] G. Biau et E. Scornet, A random forest guided tour, *TEST* **25** (2016) 197–227.

[8] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)* (Springer-Verlag, 2006).

[9] C. M. Bishop et H. Bishop, *Deep Learning - Foundations and Concepts* (Springer, 2024).

[10] H. Bourlard et Y. Kamp, Auto-association by multilayer perceptrons and singular value decomposition, *Biol. Cybern.* **59**(4-5) (1988) 291–294.

[11] C.-C. Chang et C.-J. Lin, LIBSVM: A library for support vector machines, *ACM Transactions on Intelligent Systems and Technology* **2** (2011) 1–27. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[12] T. Chen et C. Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2016), KDD '16, Association for Computing Machinery, p. 785–794.

[13] G. Cybenko, Approximation by superpositions of a sigmoidal function, *Math. Control Signal Systems* **2** (1989) 303–314.

[14] A. Belotto da Silva et M. Gazeau, A general system of differential equations to model first-order adaptive algorithms, *Journal of Machine Learning Research* **21** (2020) 1–42.

[15] B. Dai et D. Wipf. Diagnosing and enhancing VAE models. In *International Conference on Learning Representations* (2019).

[16] T. Duchamp et W. Stuetzle, Extremal properties of principal curves in the plane, *Ann. Statist.* **24**(4) (1996) 1511–1520.

[17] A. F. Duque, S. Morin, G. Wolf, et K. Moon. Extendable and invertible manifold learning with geometry regularized autoencoders. In *2020 IEEE International Conference on Big Data (Big Data), Atlanta, GA, USA* (2020), pp. 5027–5036.

[18] M. Ester, H.-P. Kriegel, J. Sander, et X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining* (1996), KDD'96, AAAI Press, p. 226–231.

[19] S. Gerber, Saddlepoints in unsupervised least squares, *arXiv preprint* **2104.05000** (2021).

[20] S. Gerber, T. Tasdizen, et R. Whitaker. Dimensionality reduction and principal surfaces via kernel map manifolds. In *2009 IEEE 12th International Conference on Computer Vision* (2009), pp. 529–536.

[21] S. Gerber et R. Whitaker, Regularization-free principal curve estimation, *J. Mach. Learn. Res.* **14** (2013) 1285–1302.

[22] P. Ghosh, M. S. M. Sajjadi, A. Vergari, M. Black, et B. Scholkopf. From variational to deterministic autoencoders. In *International Conference on Learning Representations* (2020).

[23] Ian Goodfellow, Yoshua Bengio, et Aaron Courville, *Deep Learning* (MIT Press, 2016). `http://www.deeplearningbook.org`.

[24] T. Hastie et W. Stuetzle, Principal curves, *J. Amer. Statist. Assoc.* **84**(406) (1989) 502–516.

[25] T. Hastie, R. Tibshirani, et J. Friedman, *The Elements of Statistical Learning*, 2nd ed., Springer Series in Statistics (Springer, 2009).

[26] G. E. Hinton et R. R. Salakhutdinov, Reducing the dimensionality of data with neural networks, *Science* **313**(5786) (2006) 504–507.

[27] K. Hornik, M. Stinchcombe, et H. White, Multilayer feedforward networks are universal approximators, *Neural Networks* **2**(5) (1989) 359–366.

[28] K. Iyer, R. Bhalodia, et S. Elhabian, RENs: Relevance encoding networks, *arXiv preprint* **2205.13061** (2022).

[29] K. Jia, L. Sun, S. Gao, Z. Song, et B. E. Shi, Laplacian auto-encoders: An explicit learning of nonlinear data manifold, *Neurocomputing* **160** (2015) 250–260.

[30] L. Kaufman et P. Rousseeuw. Clustering by means of Medoids. In *Statistical Data Analysis Based on the L1-norm and Related Methods* (1987), Y. Dodge, Ed., North-Holland, pp. 405–416.

[31] D.P. Kingma et J. Ba, Adam: A method for stochastic optimization, *3rd International Conference for Learning Representations* (2014).

[32] D. P. Kingma et M. Welling. Auto-encoding variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings* (2014), Y. Bengio and Y. LeCun, Eds.

[33] D. P. Kingma et M. Welling, An introduction to variational autoencoders, *Foundations and Trends® in Machine Learning* **12**(4) (2019) 307–392.

[34] M. A. Kramer, Nonlinear principal component analysis using autoassociative neural networks, *AIChE Journal* **37**(2) (1991) 233–243.

[35] Y. Lee, S. Yoon, M. Son, et F. C. Park. Regularized autoencoders for isometric representation learning. In *International Conference on Learning Representations* (2022).

[36] T. Lelièvre, T. Pigeon, G. Stoltz, et W. Zhang, Analyzing multimodal probability measures with autoencoders, *J. Phys. Chem. B* (2024).

[37] M. Leshno, V. Y. Lin, A. Pinkus, et S. Schocken, Multilayer feedforward networks with a nonpolynomial activation function can approximate any function, *Neural Networks* **6**(6) (1993) 861–867.

[38] G. Louppe, *Understanding Random Forests: From Theory to Practice* (PhD thesis, Université de Liège, 2014).

[39] Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexandre G.R. Day, Clint Richardson, Charles K. Fisher, et David J. Schwab, A high-bias, low-variance introduction to machine learning for physicists, *Physics Reports* **810** (2019) 1–124.

[40] M. Mohri, A. Rostamizadeh, et A. Talwalkar, *Foundations of Machine Learning*, 2 ed. (MIT Press, 2018).

[41] K. P. Murphy, *Probabilistic Machine Learning: An introduction* (MIT Press, 2022).

[42] A. Ng. Sparse autoencoder. In *CS294A Lecture notes* (2011), volume 72, pp. 1–19.

[43] M. A. Nielsen, *Neural Networks and Deep Learning* (Determination Press, 2015).

[44] Y. Opochinsky, S.E. Chazan, S Gannot, et J. Goldberger. k-Autoencoders deep clustering. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (2020), pp. 4037–4041.

[45] H.-S. Park et C.-H. Jun. A simple and fast algorithm for K-medoids clustering. In *Expert Systems with Applications* (2009), volume 36.2, pp. 3336–3341.

[46] S. Rifai, P. Vincent, X. Muller, X. Glorot, et Y. Bengio. Contractive auto-encoders: Explicit invariance during feature extraction. In *ICML* (2011), L. Getoor and T. Scheffer, Eds., Omnipress, pp. 833–840.

[47] H. Robbins et S. Monro, A stochastic approximation method, *Ann. Math. Statist.* **22**(3) (1951) 400–407.

[48] B. Schölkopf, A. Smola, et K.-R. Müller, Nonlinear component analysis as a kernel eigenvalue problem, *Neural Comput.* **10**(5) (1998) 1299–1319.

[49] I. Sekkat et G. Stoltz, Mini-batching error and adaptive Langevin dynamics, *Journal of Machine Learning Research* **24**(329) (2023) 1–58.

[50] S. Shalev-Shwartz et S. Ben-David, *Understanding Machine Learning: From Theory to Algorithms* (Cambridge University Press, 2014).

[51] Y. Sun, H. Mao, Q. Guo, et Y. Zhang, Learning a good representation with unsymmetrical auto-encoder, *Neural Comput. Appl.* **27** (2016) 1361–1367.

[52] I. Sutskever, J. Martens, G. Dahl, et G. Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning* (Atlanta, Georgia, USA, 17–19 Jun 2013), S. Dasgupta and D. McAllester, Eds., volume 28 de *Proceedings of Machine Learning Research*, PMLR, pp. 1139–1147.

[53] R. Tibshirani, Principal curves revisited, *Stat. Comput.* **2** (1992) 183–190.

[54] R. Tibshirani, Regression shrinkage and selection via the Lasso, *Journal of the Royal Statistical Society. Series B (Methodological)* **58**(1) (1996) 267–288.

[55] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, et P.-A. Manzagol, Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion, *J. Mach. Learn. Res* **11**(110) (2010) 3371–3408.

[56] U. von Luxburg, A tutorial on spectral clustering, *Stat. Comput.* **17** (2007) 395–416.

[57] X. Xu, H. Hou, et S. Ding, Semi-supervised deep density clustering, *Applied Soft Computing* **148** (2023) 110903.

[58] A. Zhang, Z. C. Lipton, M. Li, et A. J. Smola, *Dive into Deep Learning* (Cambridge University Press, 2023).

[59] J. H. Zhao, P. L. H. Yu, et Q. Jiang, ML estimation for factor analysis: EM or non-EM?, *Stat. Comput.* **18** (2008) 109–123.