
Introduction to Machine Learning

Alessandro Rudi, Umut Şimşekli

Class notes by Antoine Groudiev



Last modified 2nd June 2024

Contents

| | | |
|----------|---|-----------|
| 1 | An overview of Machine Learning | 4 |
| 2 | Linear Least Squares Regression | 4 |
| 2.1 | Introduction | 4 |
| 2.2 | General setup and notations for supervised learning | 4 |
| 2.3 | Ordinary Least Squares Estimator (OLS) | 6 |
| 2.3.1 | Definition and closed form | 6 |
| 2.3.2 | Geometric interpretation | 6 |
| 2.3.3 | Numerical resolution | 7 |
| 2.3.4 | Nonlinear problem: polynomial, spline, and kernel regression | 7 |
| 2.4 | Statistical analysis | 8 |
| 2.4.1 | Stochastic assumptions and bias/variance decomposition | 8 |
| 2.4.2 | Statistical properties of OLS | 9 |
| 2.4.3 | Gaussian noise model | 10 |
| 2.5 | Ridge regression | 10 |
| 2.5.1 | Handling non-injective design matrices | 10 |
| 2.5.2 | Ridge regression | 10 |
| 2.5.3 | Comparison to the OLS | 11 |
| 3 | Logistic regression and convex analysis | 11 |
| 3.1 | Logistic regression | 11 |
| 3.1.1 | Motivation | 12 |
| 3.1.2 | Loss function | 12 |
| 3.1.3 | Computation of the estimator | 13 |
| 3.1.4 | Regularization | 14 |
| 3.2 | Convex analysis | 14 |
| 3.2.1 | Convexity and minimization problems | 14 |
| 3.2.2 | Convex sets | 14 |
| 3.2.3 | Convex functions | 16 |
| 3.2.4 | Unconstrained optimization problems | 17 |
| 4 | Convex analysis and convex optimization | 18 |
| 4.1 | Constrained optimization problems | 18 |
| 4.1.1 | Lagrangian duality | 19 |
| 4.1.2 | Link between primal and dual problems | 20 |
| 4.1.3 | Strong duality | 20 |
| 4.1.4 | Karush-Kuhn-Tucker optimality conditions | 21 |
| 4.2 | Optimization algorithms for unconstrained convex optimization | 22 |
| 4.2.1 | Gradient Descent | 22 |
| 4.2.2 | Gradient Descent for strongly convex functions | 23 |
| 4.2.3 | Stochastic Gradient Descent | 24 |
| 5 | Kernels | 25 |
| 5.1 | Introduction to kernels | 25 |
| 5.2 | Representer theorem | 26 |
| 5.2.1 | Theorem statement | 26 |
| 5.2.2 | Finite dimensional representation of the learning problem | 27 |
| 5.3 | Properties of kernels | 28 |

| | | |
|-----------|---|-----------|
| 6 | Elements of Statistical Machine Learning | 29 |
| 6.1 | Introduction | 29 |
| 6.2 | Empirical Risk Minimization | 30 |
| 6.3 | Preliminary results | 30 |
| 6.3.1 | Results on random variables | 30 |
| 6.3.2 | Bernstein's inequality | 31 |
| 6.4 | Consistency and Learning rates for Empirical Risk Minimization | 31 |
| 6.4.1 | Introduction | 31 |
| 6.4.2 | Bounding the variance term: PAC bounds | 32 |
| 6.4.3 | Bounds in probability | 33 |
| 6.5 | Bounding the Bias | 34 |
| 7 | k-Nearest Neighbours | 34 |
| 7.1 | Introduction | 34 |
| 7.1.1 | Goal | 34 |
| 7.1.2 | Intuition of the base algorithm | 34 |
| 7.1.3 | Assumptions and notations | 35 |
| 7.2 | Classifiers and estimators | 35 |
| 7.2.1 | The optimal Bayes classifier | 35 |
| 7.2.2 | Plug-in estimator | 36 |
| 7.3 | The k -nearest neighbors classifier (k -NN) | 37 |
| 7.3.1 | The nearest neighbor classifier (1-NN) | 37 |
| 7.3.2 | Inconsistency of the k -NN classifier for fixed k | 37 |
| 7.3.3 | Consistency of nearest neighbors making $k \rightarrow +\infty$ | 37 |
| 8 | Unsupervised Learning | 37 |
| 8.1 | K -means | 37 |
| 8.1.1 | Distortion | 37 |
| 8.1.2 | Algorithm | 38 |
| 8.1.3 | Convergence and initialization: K -means++ | 38 |
| 8.1.4 | Choice of K | 39 |
| 8.1.5 | Other problems | 39 |
| 8.2 | Dimensionality reduction: principal component analysis | 40 |
| 8.2.1 | Analysis view | 40 |
| 8.2.2 | In practice | 41 |
| 9 | Probabilistic models and Maximum Likelihood | 41 |
| 9.1 | Introduction | 41 |
| 9.2 | Maximum likelihood estimation | 42 |
| 9.2.1 | Definition and example | 42 |
| 9.2.2 | Link with empirical risk minimization | 43 |
| 9.2.3 | Link with Kullback-Leibler divergence | 44 |
| 9.2.4 | Conditional modeling | 45 |
| 9.2.5 | Linear regression as conditional modeling | 45 |
| 9.3 | Maximum a posteriori | 46 |
| 10 | Posterior sampling using MCMC and SGLD | 48 |
| 10.1 | Introduction | 48 |
| 10.1.1 | Bayesian probabilistic model | 48 |
| 10.1.2 | Level of uncertainty | 48 |
| 10.2 | Markov Chain Monte-Carlo (MCMC) | 49 |

| | | |
|-----------|---|-----------|
| 10.2.1 | Monte-Carlo estimator | 49 |
| 10.2.2 | Sampling with Markov processes | 49 |
| 10.3 | Stochastic Gradient Langevin Dynamics | 50 |
| 10.3.1 | Notation | 50 |
| 10.3.2 | The Langevin equation | 50 |
| 10.3.3 | Numerical simulation | 51 |
| 10.3.4 | Stochastic Gradient Langevin Dynamics | 52 |
| 11 | Neural Networks | 53 |
| 11.1 | Introduction | 53 |
| 11.2 | A single neuron | 54 |
| 11.2.1 | Background | 54 |
| 11.2.2 | Logistic regression using sigmoid activation and cross-entropy loss | 54 |
| 11.2.3 | Gradient and SGD | 54 |
| 11.3 | One hidden layer | 56 |
| 11.3.1 | Motivation | 56 |
| 11.3.2 | Definition of hidden layers | 56 |
| 11.3.3 | Gradient through backpropagation | 56 |
| 11.3.4 | Approximation properties | 57 |
| 11.3.5 | Link with kernel methods | 57 |
| 11.4 | Multiple hidden layers | 58 |
| 12 | Appendix | 58 |
| 12.1 | Complexity of linear algebra computations | 58 |

Abstract

This document is Antoine Groudiev's class notes while following the class *Introduction to Machine Learning* (Apprentissage Statistique) at the Computer Science Department of ENS Ulm. It is freely inspired by the class notes written by Francis Bach, Pierre Gaillard, Alessandro Rudi, and Umut Şimşekli.

1 An overview of Machine Learning

2 Linear Least Squares Regression

2.1 Introduction

In this chapter, we will study the simple but still widely used problem of *Linear Least Square Regression*. We are given a set of points, which we assume to be sampled from some distribution: there exists some function which generated these points, and we want to retrieve or at least approximate this unknown function. To do so, we will naturally look for the function which best fits the points; nevertheless, assuming that it is unlikely that the function is overly-complicated, we will only approximate it using *linear function*. Finally, to choose which linear function “fits best” the data, we will introduce the mean square error, which we will minimize to find our linear approximation function.

Formally, our objective is to find a function f such that it explains well the distribution $(y_i)_{1 \leq i \leq n}$ as a function of $(x_i)_{1 \leq i \leq n}$, that is $y_i \sim f(x_i)$. To do this, we can choose a *function space* \mathcal{F} and solve the empirical risk minimization problem:

$$\hat{f}_n \in \arg \min_{f \in \mathcal{F}} \hat{R}_n(f) := \arg \min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n (y_i - f(x_i))^2$$

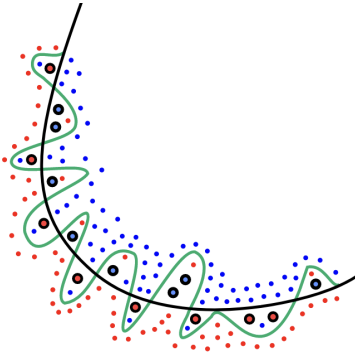


Figure 2.1: Example of overfitting

Care must be taken when selecting the function space to avoid overfitting¹: for example, if we were to choose $\mathcal{F} := \mathbb{R}_d[X]$ for some $d \in \mathbb{N}$, it is in our best interest to keep d small to avoid getting a function f which fits perfectly the points but diverges between them. Although the empirical mean square error \hat{R}_n decreases when the function space \mathcal{F} becomes larger (i.e. larger polynomial degrees), the \hat{f}_n estimator loses its predictive power. \hat{f}_n will not necessarily perform well on new data. In what follows, we will consider the linear function space, containing functions of the form $f : x \mapsto ax + b$, which is the simplest.

2.2 General setup and notations for supervised learning

Definition (Training data set). The *training data set*, often denoted $D_n := \{ (x_i, y_i) \mid i \in \llbracket 1, n \rrbracket \}$, is the set of some observations $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$. We will often make the assumption that the observations (x_i, y_i) are realizations of i.i.d. random variables from a distribution ν .

The distribution ν is unknown to the statistician; it's a matter of learning it from the data.

¹We say that the function *overfits* the data when it corresponds too closely to the specific set of data (i.e. on the training data), such that it fails to fit additional data (i.e. test data).

Definition (Learning algorithm). A *learning algorithm* – or *learning rule* – \mathcal{A} is a function that associates to training data D_n a prediction function \hat{f}_n :

$$\mathcal{A} : \bigcup_{n \in \mathbb{N}} (\mathcal{X} \times \mathcal{Y})^n \longrightarrow \mathcal{Y}^{\mathcal{X}}$$

$$D_n \longmapsto \hat{f}_n$$

The estimated function \hat{f}_n is constructed to predict a new output y from a new input x , where (x, y) is a pair of *test data*, i.e. not necessarily observed in the training data. The function \hat{f}_n is an estimator because it depends on the data D_n and not on unobserved parameter, such as the distribution ν . If D_n is random, it is also a random function.

Definition (Squared Loss Risk). Given an estimator \hat{f}_n , we define its risk:

$$\mathcal{R}(\hat{f}_n) := \mathbb{E} \left[(Y - \hat{f}_n(X))^2 \mid D_n \right] \quad \text{where } (X, Y) \sim \nu \quad (2.2.1)$$

This is also called the *generalization error*, as it measures how well the estimator performs on other inputs and outputs of the dataset.

In practice, the statistician cannot compute the risk, since one cannot access the distribution ν . Therefore, a common method in supervised machine learning is to replace the risk (defined using the distribution ν through the expectation \mathbb{E}) by the empirical risk (defined using the training data set).

Definition (Squared Loss Empirical risk). Given an estimator \hat{f}_n and a data set $D_n = \{ (x_i, y_i) \mid i \in \llbracket 1, n \rrbracket \}$, we define its *empirical risk*:

$$\hat{\mathcal{R}}_n(\hat{f}_n) := \frac{1}{n} \sum_{i=1}^n (y_i - \hat{f}_n(x_i))^2 \quad (2.2.2)$$

However, one must be careful about overfitting, the case where $\hat{\mathcal{R}}_n(f)$ is much lower than $\mathcal{R}(f)$, as discussed previously. In this chapter, we will study the performance of the least square estimator in the case of the linear model.

Definition (Linear model). When $\mathcal{X} = \mathbb{R}^d$ and $\mathcal{Y} = \mathbb{R}$, the simplest interesting function space is the set of affine functions. To ease the notation, we assume that the first components of the inputs is 1 so that it is sufficient to consider linear functions. Therefore, the function space is:

$$\mathcal{F} := \left\{ x \mapsto \theta^\top x \mid \theta \in \mathbb{R}^d \right\} \quad (2.2.3)$$

i.e. linear functions parametrized by $\theta \in \mathbb{R}^d$.

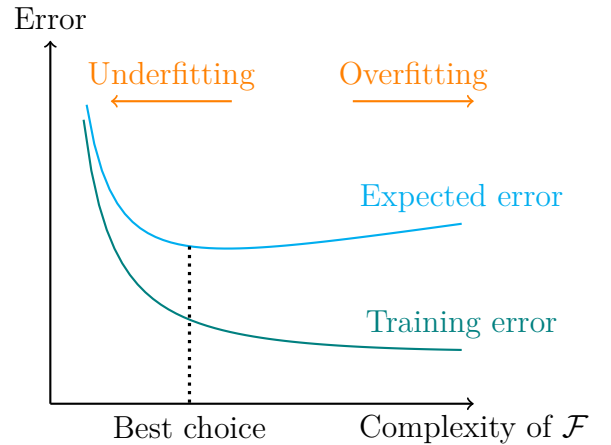


Figure 2.2: Overfitting and underfitting

Remark. Choosing a linear model, the empirical risk minimization corresponds to the problem of minimizing the following quantity over $\theta \in \mathbb{R}^d$:

$$\hat{\mathcal{R}}_n(\theta) := \frac{1}{n} \sum_{i=1}^n (y_i - \theta^\top x_i)^2$$

This expression can be rewritten using matrix notation. We let $Y = (Y_1, \dots, Y_n)^\top \in \mathbb{R}^n$ be the vector of outputs and $X \in \mathbb{R}^{n \times d}$ the matrix of inputs, whose rows are x_i^\top . X is called the design matrix. The empirical risk is therefore given by:

$$\hat{\mathcal{R}}_n(\theta) = \frac{1}{n} \|Y - X\theta\|_2^2$$

2.3 Ordinary Least Squares Estimator (OLS)

2.3.1 Definition and closed form

In the following, we assume that the design matrix is injective.² In particular, $d \leq n$, otherwise this is not possible.

Definition (Ordinary Least Squares). If X is injective, the minimizer of the empirical risk (2.2.2) is called the *Ordinary Least Squares (OLS) estimator*. Said otherwise, it is the vector $\hat{\theta} \in \mathbb{R}^d$ minimizing $\hat{\mathcal{R}}_n$:

$$\hat{\theta} := \arg \min_{\theta \in \mathbb{R}^d} \hat{\mathcal{R}}_n(\theta) = \arg \min_{\theta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n (y_i - \theta^\top x_i)^2 \quad (2.3.1)$$

Property 2.1 (Closed form solution of the OLS estimator). If X is injective, the OLS estimator exists and is unique. Moreover, it is given by:

$$\hat{\theta} = (X^\top X)^{-1} X^\top Y \quad (2.3.2)$$

Proof. Since $\hat{\mathcal{R}}_n$ is coercive³ and continuous, it admits at least a minimizer. Furthermore, we have:

$$\hat{\mathcal{R}}_n(\theta) := \frac{1}{n} \|Y - X\theta\|_2^2 = \frac{1}{n} (\theta^\top (X^\top X) \theta - 2\theta^\top X^\top Y + \|Y\|^2)$$

Since $\hat{\mathcal{R}}$ is differentiable any minimizer cancels its gradient:

$$\nabla \hat{\mathcal{R}}_n(\hat{\theta}) = \frac{1}{n} (\hat{\theta}^\top (X^\top X) + (X^\top X) \hat{\theta} - 2X^\top Y) = \frac{2}{n} ((X^\top X) \hat{\theta} - X^\top Y)$$

where the last equality holds because $X^\top X \in \mathbb{R}^{d \times d}$ is symmetric. Since X is injective, $X^\top X$ is invertible⁴. Therefore, a solution of $\nabla \hat{\mathcal{R}}_n(\hat{\theta}) = 0$ satisfies:

$$\hat{\theta} = (X^\top X)^{-1} X^\top Y$$

Finally, this unique solution is indeed a minimum since its Hessian is definite positive:

$$\nabla^2 \hat{\mathcal{R}}_n(\hat{\theta}) = \frac{2}{n} (X^\top X)$$

□

2.3.2 Geometric interpretation

The linear model aims modeling the output vector $Y \in \mathbb{R}^n$ by a linear combination of the form $X\theta \in \mathbb{R}^n$. The image of X is the solution space, denoted:

$$\text{Im}(X) = \{ X\theta \in \mathbb{R}^n \mid \theta \in \mathbb{R}^d \}$$

This is the vector subspace of \mathbb{R}^n generated by the $d \leq n$ columns of the design matrix. As $\text{rg}(X) = d$, it is of dimension d .

By minimizing $\|Y - X\theta\|$, we thus look for the element of $\text{Im}(X)$ closest to Y . This is the orthogonal projection of Y on $\text{Im}(X)$, denoted \hat{Y} . By definition of the OLS and by Property 2.1, we have:

$$\hat{Y} := X\hat{\theta} = X(X^\top X)^{-1} X^\top Y$$

In particular, $P_X := X(X^\top X)^{-1} X^\top$ is the projection matrix on $\text{Im}(X)$.

²Said otherwise, the rank of X is d .

³ $\|\hat{\mathcal{R}}_n(\theta)\| \xrightarrow{\|\theta\| \rightarrow +\infty} +\infty$

⁴It is even positive definite.

2.3.3 Numerical resolution

The closed form formula (2.3.2) of the OLS is useful in analyzing it; however, calculating it naively can be prohibitively expensive. For example, when d is large, one prefers to avoid inverting the design matrix $X^\top X$ which costs $O(d^3)$ ⁵, and can be very unstable when the matrix is badly conditioned. The following methods are usually preferred.

QR factorization To improve stability, QR decomposition can be used. Since $\hat{\theta}$ is the solution of the equation:

$$(X^\top X)\hat{\theta} = X^\top Y$$

we write $X \in \mathbb{R}^{n \times d}$ as $X = QR$ where $Q \in \mathbb{R}^{n \times d}$ is an orthogonal matrix⁶ and $R \in \mathbb{R}^{d \times d}$ is upper triangular. Upper triangular matrices are very useful for solving linear systems. Substituting in the previous equation, we get:

$$\begin{aligned} R^\top (Q^\top Q) R \hat{\theta} &= R^\top Q^\top Y \iff R^\top R \hat{\theta} = R^\top Q^\top Y \\ &\iff R \hat{\theta} = Q^\top Y \end{aligned}$$

All that remains is to solve a linear system with a triangular upper matrix, which is easy.

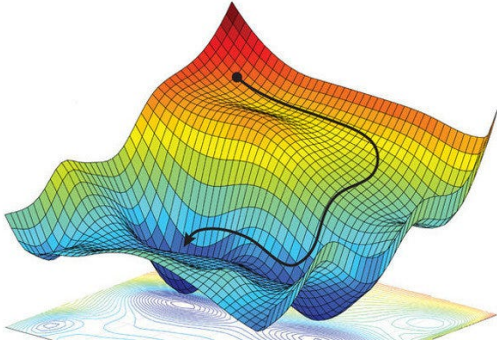


Figure 2.3: (Non-convex) Gradient descent

Gradient descent We can completely bypass the need of matrix inversion or factorization using gradient descent. It consists in solving the minimization problem step by step by approaching the minimum through gradient steps. For example, we initialize $\theta_0 := 0^7$, then update it using the following recurrence formula:

$$\begin{aligned} \theta_{i+1} &:= \theta_i - \eta \cdot \nabla \hat{\mathcal{R}}_n(\theta_i) \\ &= \theta_i - \eta \cdot \frac{2}{n} \left((X^\top X)\theta_i - Y^\top X \right) \end{aligned}$$

where $\eta > 0$ is a learning parameter called *learning rate*. We see that if the algorithm converges, then it converges to a point cancelling the gradient, thus to the (unique) OLS solution. For the algorithm to converge, the learning rate η must be well calibrated. This will be seen in more details in the following chapter about gradient descent.

If the data set is too big, i.e. when $n \gg 1$, loading all the data to make the gradient calculation $\nabla \hat{\mathcal{R}}(\theta_i)$ can be prohibitively expensive too. The common solution to this is to use *Stochastic Gradient Descent*, where gradient calculations for one step are made only on estimates of $\nabla \hat{\mathcal{R}}(\theta_i)$, calculated on a random subset of the data.

2.3.4 Nonlinear problem: polynomial, spline, and kernel regression

The assumption that the observations y_i can be explained as a linear combination of the explanatory variables $x_{i,j}$ may seem strong. However, the previous linear framework can be applied to transformations of the variables $x_{i,j}$. For example, by adding the powers of the variables $x_{i,j}^k$ or their products $x_{i,j} \cdot x_{i',j'}$, this allows comparison to polynomial spaces. Doing a

⁵Using the Gauss-Jordan method

⁶That is $QQ^\top = I_n$

⁷In practice, θ_0 is often initialized to some random vector to avoid singularities.

linear regression on polynomial transformations of variables is equivalent to doing a polynomial regression.

Of course, other bases and transformations exist: for instance, *spline bases* are piecewise polynomials with constraints on the edgets. This is the model used for example by EDF to predict electricity consumption as a function of variables such as the time of the day, the day of the week, temperature or cloud cover. In general, we can consider transformations $\phi : \mathcal{X} \rightarrow \mathbb{R}^d$ and try to explain the outputs y_i with functions of the form $\theta \rightarrow \phi(x_i)^\top \theta$. Another form of regression that we will discuss in the following is therefore *kernel regression*, which allows computing efficiently the estimator even when ϕ maps to an infinite dimensional space.

2.4 Statistical analysis

In this section, we try to show some guarantees for the OLS estimator, such as a bound the excess risk of the OLS, which we will define later on.

2.4.1 Stochastic assumptions and bias/variance decomposition

To provide guarantees on the performance of OLS, we require assumptions about how the data is generated. In this section, we consider a stochastic framework that will allow us to statistically analyze OLS.

Assumption: linear model We assume that there exists a vector $\theta^* \in \mathbb{R}^d$ such that for all $1 \leq i \leq n$,

$$Y_i = x_i^\top \theta^* + Z_i \quad (2.4.1)$$

where $Z = (Z_1, \dots, Z_n)^\top \in \mathbb{R}^n$ is a vector of *errors*, also called *noise*. The Z_i are assumed to be centered independent variables of variance σ^2 , i.e. $\mathbb{E}[Z_i] = 0$ and $\mathbb{V}[Z_i] = \sigma^2$. The assumption (2.4.1) can be rewritten in matrix form:

$$Y = X\theta^* + Z \quad (2.4.2)$$

where $Y = (Y_1, \dots, Y_n)^\top \in \mathbb{R}^n$, $X = (x_1, \dots, x_n)^\top \in \mathbb{R}^{n \times d}$, and $Z = (Z_1, \dots, Z_n)^\top \in \mathbb{R}^n$.

Remark. We write Y_i and Z_i using capital letters to remind ourselves that they are random variables. The noise Z comes from the fact that in practice, the observations Y_i never completely fit the linear forecast. They are due to noise or unobserved explanatory variables. As before, we assume that the first vector of the explanatory variable is the constant vector, i.e. $\forall i, x_{i,1} = 1$

Analysis settings Two settings of analysis can be chosen:

- In the *fixed desing* setting, the design matrix X is not random but deterministic and the features x_1, \dots, x_n are fixed. The expectations are thus only with respect to the Z_i and the Y_i , and the goal is to minimize:

$$\mathcal{R}_X(\theta) = \mathbb{E} \left[\frac{1}{n} \sum_{i=1}^n (Y_i - x_i^\top \theta)^2 \right]$$

for new random observations Y_i , but on the same inputs.

- In the *random design* setting, both the inputs and the ouputs are random. This is the most standard setting in supervised machine learning. The goal is therefore to minimize the risk (2.2.1) – also called the generalization error.

In this chapter, we will consider the fixed design setting, because it eases both the notation and the calculations – we will only need some simple linear algebra.

Before analyzing the statistical properties of OLS, we state a general result under the linear model assumption, which illustrates the tradeoff between estimation and approximation – or bias and variance.

Property 2.2 (Risk decomposition). Under the linear model assumption with fixed design, the following holds:

$$\forall \theta \in \mathbb{R}^d, \quad \mathbb{E}[\mathcal{R}_X(\theta) - \mathcal{R}_X(\theta^*)] = \|\theta - \theta^*\|_\Sigma^2$$

where $\Sigma := \frac{1}{n} X^\top X \in \mathbb{R}^{d \times d}$ and $\|\alpha\|_\Sigma^2 := \alpha^\top \Sigma \alpha$. Furthermore, if θ is a random variable – when it depends on a random dataset – we have:

$$\mathbb{E}[\mathcal{R}_X(\theta)] - \mathcal{R}(\theta^*) = \underbrace{\|\mathbb{E}[\theta] - \theta^*\|_\Sigma^2}_{\text{Bias}} + \underbrace{\mathbb{E}[\|\theta - \mathbb{E}[\theta]\|_\Sigma^2]}_{\text{Variance}}$$

Proof. □

Remark. *It is worth to note that the optimal risk satisfies:*

$$\mathcal{R}_X(\theta^*) = \mathbb{E} \left[\frac{1}{n} \sum_{i=1}^n (Y_i - x_i^\top \theta^*)^2 \right] = \mathbb{E} \left[\frac{1}{n} \sum_{i=1}^n Z_i^2 \right] = \frac{1}{n} \sum_{i=1}^n \mathbb{E}[Z_i^2] = \sigma^2$$

2.4.2 Statistical properties of OLS

We can now show some guarantees for the OLS estimator.

Property 2.3. Under the linear model assumption with fixed design, the OLS estimator $\hat{\theta}$ defined by (2.3.1) satisfies:

$$\mathbb{E}[\hat{\theta}] = \theta^* \quad \text{and} \quad \mathbb{V}[\hat{\theta}] = \frac{\sigma^2}{n} \Sigma^{-1}$$

i.e. it is unbiased, and its variance is some $O(n^{-1})$. Furthermore, we can even show that it satisfies the Gauss-Markov property: it is optimal among all unbiased estimators of θ , in the sense that it has a minimal variance-covariance matrix.

Proof. □

Definition (Excess risk). The excess risk of an estimator $\hat{\theta}$ is defined by:

$$\mathcal{R}(\hat{\theta}) := \mathbb{E}[\mathcal{R}_X(\hat{\theta})] - \mathcal{R}(\theta^*) \tag{2.4.3}$$

It allows to measure the risk induced by the use of an estimator instead of the optimal vector, without taking into account the inherent risk generated by the noise, which cannot be reduced.

Corollary (Excess risk of OLS). Under the linear model assumption with fixed design, the excess risk of the OLS satisfies:

$$\mathcal{R}(\hat{\theta}) := \mathbb{E}[\mathcal{R}_X(\hat{\theta})] - \mathcal{R}(\theta^*) = \frac{\sigma^2 d}{n}$$

Proof. □

2.4.3 Gaussian noise model

A special case that is often considered is Gaussian noise, i.e. $Z_i \sim \mathcal{N}(0, \sigma^2)$, the normal distribution of expectation 0 and variance σ^2 . This choice comes not only from the fact that it allows to compute many additional statistical properties on $\hat{\theta}$ and to perform tests, such as confidence intervals or significance of variables. In practice, it is also motivated by the central limit theorem, and the fact that noise is often an addition of many phenomena not explained by the linear combination of the explanatory variables.

Property 2.4. In the linear model with Gaussian noise, the maximum likelihood estimators of θ and σ satisfy respectively:

$$\hat{\theta}_{MV} = (X^\top X)^{-1} X^\top Y \quad \text{and} \quad \hat{\sigma}_{MV}^2 = \frac{\|Y - X\hat{\theta}\|^2}{n}$$

We therefore find the least squares estimator obtained by minimizing the empirical risk. The variance estimator is biased. We will see more about maximum likelihood estimators in next chapters.

2.5 Ridge regression

2.5.1 Handling non-injective design matrices

If X is not injective, the matrix $X^\top X$ is no longer invertible and the OLS optimization problem admits several solutions. The problem is said to be *poorly posed* or *unidentifiable*. Since the variance of $\hat{\theta}$ depends on the conditioning of the matrix $(X^\top X)^{-1}$, the more columns of it are likely to be dependent, the less stable $\hat{\theta}$ will be. Several solutions allow dealing with the case where $\text{rg}(X) < D$.

Explicit complexity control reduces the $\text{Im}(X)$ solution space: this can be done by removing columns from the X matrix until it becomes injective (for example, by reducing the degree of polynomials). One can also set identifiability constraints of the form $\theta \in V$, a vector subspace of \mathbb{R}^d such that any element $y \in \text{Im}(X)$ has a unique antecedent $\theta \in V$ with $y = X\theta$. For example, we could choose $V = \ker(X)^\perp$.

Implicit complexity control regularizes the empirical risk minimization problem. The most common approach is to regularize by adding $\|\theta\|_2^2$ (Ridge regression) or $\|\theta\|_1$ (Lasso regression).

2.5.2 Ridge regression

Definition. For a regularization parameter λ , the Ridge regression estimator is defined as

$$\hat{\theta}_\lambda \in \arg \min \left\{ \frac{1}{n} \|Y - X\theta\|_2^2 + \lambda \|\theta\|_2^2 \mid \theta \in \mathbb{R}^d \right\} \quad (2.5.1)$$

The regularization parameter $\lambda > 0$ regulates the trade-off between the variance of $\hat{\theta}$ and its bias.

Property 2.5. The Ridge regression estimator is unique and satisfies:

$$\hat{\theta}_\lambda = (X^\top X + n\lambda I_n)^{-1} X^\top Y$$

Proof. Similar to the one of the OLS and left as an exercise. We can see that there is no longer the problem of inverting $X^\top X$ since the Ridge regression replaces $(X^\top X)^{-1}$ by $(X^\top X + n\lambda I_n)^{-1}$ in the OLS solution. \square

Property 2.6 (Risk of Ridge regression). Under the linear model assumption, the Ridge regression estimator satisfies:

$$\mathbb{E}[\mathcal{R}_X(\hat{\theta}_\lambda)] - \mathcal{R}_X(\theta^*) = \sum_{j=1}^d (\theta_j^*)^2 \frac{\lambda_j}{(1 + \lambda_j/\lambda)^2} + \frac{\sigma^2}{n} \sum_{j=1}^d \frac{\lambda_j^2}{(\lambda_j + \lambda)^2}$$

where λ_j is the j -th eigenvalue of $\Sigma = \frac{1}{n}X^\top X$. In particular, the choice of:

$$\lambda^* = \frac{\sigma \sqrt{\text{Tr}(\Sigma)}}{\|\theta^*\|_2 \sqrt{n}}$$

yields the following excess risk:

$$\mathbb{E}[\mathcal{R}_X(\hat{\theta}_{\lambda^*})] - \mathcal{R}_X(\theta^*) \leq \frac{\sigma \sqrt{2 \text{Tr}(\Sigma) \|\theta^*\|_2}}{\sqrt{n}}$$

Proof. Follows from the bias-variance decomposition, and is left as an exercise. \square

Remark. As $\lambda \rightarrow 0$, its excess risk converges to the one of OLS. The first term corresponds to the bias of the Ridge estimator. Thus, on the downside, the Ridge estimator is biased in contrast to the OLS. But on the positive side, its variance does not involve the inverse of Σ but of $\Sigma + \lambda I_d$ instead, which is better conditioned. It has therefore a lower variance. The parameter λ controls the trade-off.

2.5.3 Comparaison to the OLS

We can compare the excess risk bound obtained by $\hat{\theta}_{\lambda^*}$ with the one of the OLS, which was $\sigma^2 d/n$.

- The OLS convergence is in $O(n^{-1})$ while the convergence of $O(n^{-1/2})$, which is slower
- The OLS dependency on the noise is in σ^2 while Ridge's is in σ , which is better
- Since $\text{Tr}(\Sigma) \leq \max_{1 \leq i \leq n} \|x_i\|^2$, if the input norms are bounded by R , the excess risk of Ridge does not depend on the dimension d , which can even be infinite. It is called a *dimension free* bound.

The calibration of the regularization parameter is therefore essential in practice. It can for example be done analytically as in the proposition – but often, some quantities such as σ^2 and $\|\theta^*\|$ are unknown. In practice, one resorts to train/validation set or *cross-validation*.

3 Logistic regression and convex analysis

This chapter will introduce logistic regression, a widely used classification algorithm. Unlike linear regression, there is no closed-form solution and one needs to solve it using iterative convex optimization algorithms.

3.1 Logistic regression

We consider the binary classification problem: given inputs in \mathbb{R}^d , we want to predict outputs in $\{0, 1\}$. We are given a training set $D_n = \{(X_i, Y_i) \mid i \in \llbracket 1, n \rrbracket\}$, where the data points (X_i, Y_i) are i.i.d. random variables following a distribution ν in $\mathcal{X} \times \mathcal{Y} = \mathbb{R}^d \times \{0, 1\}$

3.1.1 Motivation

We would like to use an algorithm similar to linear regression introduced in the previous chapter. However, since the outputs Y_i are binary and belong to $\{0, 1\}$, a discrete set, we cannot predict them using a linear transformation of the inputs X_i . We will thus classify the data based on a classification rule of the form

$$f : \mathbb{R}^d \longrightarrow \mathbb{R}$$

which will then be passed through a function with specification $\mathbb{R} \rightarrow \{0, 1\}$. The final estimator will therefore be:

$$\mathbb{1}_{\mathbb{R}_+} \circ f$$

meaning that the estimator will predict $Y_i = +1$ exactly when $f(X_i) \geq 0$.

More precisely, we will consider linear functions f of the form:

$$f_\beta = x \longmapsto x^\top \beta$$

This assumes that the data is *linearly separable*, meaning that it can be well-explained by a linear separation.

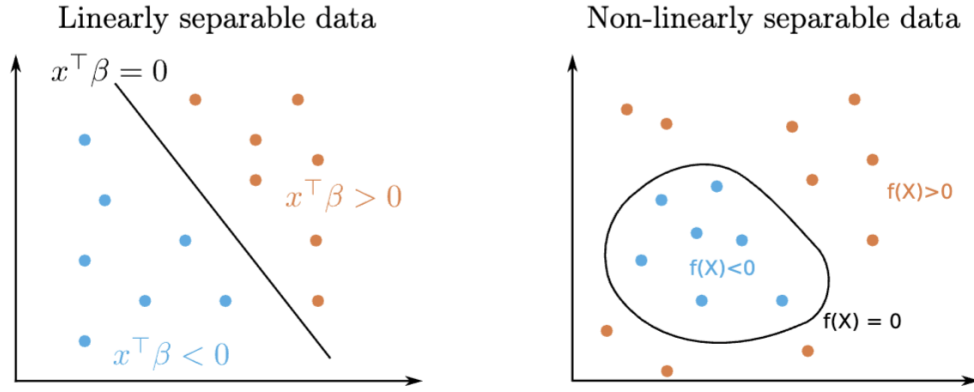


Figure 3.1: Data that can or cannot be well-explained by a linear separation

If the data does not seem to be linearly separable, we can use similar tricks as the one introduced for linear regression (polynomial regression, kernel regression, splines, ...). We will dive into more details in an upcoming chapter about kernels.

3.1.2 Loss function

To minimize the empirical risk, it remains to choose a loss function to assess the performance of a prediction.

Definition (Loss function). A *loss function* ℓ is a function of the form

$$\ell : \mathcal{Y} \times \mathcal{Y}' \longrightarrow \mathbb{R}_+$$

such that for $(y, y') \in \mathcal{Y} \times \mathcal{Y}'$, $\ell(y, y')$ intuitively quantifies the mistake when predicting y' instead of y .

Definition (Empirical Risk associated to a Loss function). Given an estimator \hat{f}_n , a data set $D_n = \{(x_i, y_i) \mid i \in \llbracket 1, n \rrbracket\}$ and a loss function ℓ , we define the *empirical risk associated to ℓ* by:

$$\hat{\mathcal{R}}_n(\hat{f}_n) := \frac{1}{n} \sum_{i=1}^n \ell(y_i, \hat{f}_n(x_i)) \quad (3.1.1)$$

Definition (Squared loss). We define the *squared loss* $\ell_2 : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}_+$ used previously in linear regression by:

$$\ell_2(y, y') := (y - y')^2 \quad (3.1.2)$$

Using the squared loss ℓ_2 in (3.1.1), we obtain the same result as the original definition (2.2.2).

Definition (Binary loss). A natural loss is the *binary loss*, also known as the *0-1 loss*. It takes 1 as a value if there is a mistake (i.e. $f(X_i) \neq Y_i$) and 0 otherwise:

$$\ell_1(X_i, Y_i) = \delta_{Y_i \neq \mathbf{1}_{\mathbb{R}_+}(f(X_i))} \quad (3.1.3)$$

Using ℓ_1 , the empirical risk becomes:

$$\hat{\mathcal{R}}_n(\beta) = \frac{1}{n} \sum_{i=1}^n \delta_{Y_i \neq \mathbf{1}_{\mathbb{R}_+}(X_i^\top \beta)}$$

This loss function is however not convex in β ; therefore, the problem of minimizing $\hat{\mathcal{R}}_n$ is extremely hard to solve. The idea of logistic regression consists in replacing the binary loss with another similar loss function, which is convex in β . This is the case of both the *Hinge loss* and the *logistic loss* which we will now introduce.

Definition (Hinge loss). The *Hinge loss* $\ell_H : \mathbb{R} \times \{-1, 1\} \rightarrow \mathbb{R}_+$ is such that it values to 0 when both arguments have the same sign and that $|y| \geq 1$ (confident prediction), and increases linearly when their signs differ or when $|y| < 1$ (prediction not confident enough):

$$\ell_H(y, y') := \max(0, 1 - yy') \quad (3.1.4)$$

Definition (Logistic loss). The *logistic loss* $\ell_l : \mathbb{R} \times \{0, 1\} \rightarrow \mathbb{R}_+$ is defined by:

$$\ell_l(y, y') := y' \log(1 - e^{-y}) + (1 - y') \log(1 + e^y) \quad (3.1.5)$$

The advantage of the logistic loss with respect to the Hinge loss is that it has a probabilistic interpretation, by modeling $\mathbb{P}(Y = 1|X)$, where (X, Y) is a couple of random variables following the law (X_i, Y_i) . We will see more on this in the lecture on Maximum Likelihood.

Definition (Logistic regression estimator). The logistic regression estimator is the solution of the following minimization problem:

$$\hat{\beta}_{(\text{logi.})} := \arg \min_{\beta \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n \ell_l(X_i^\top \beta, Y_i) \quad (3.1.6)$$

3.1.3 Computation of the estimator

Similarly to OLS, we may try to analytically solve the minimization problem to find $\hat{\beta}_{(\text{logi.})}$. This could be done by cancelling the gradient of the empirical risk. Note that:

$$\frac{\partial \ell_l(y, y')}{\partial y} = \sigma(y) - y'$$

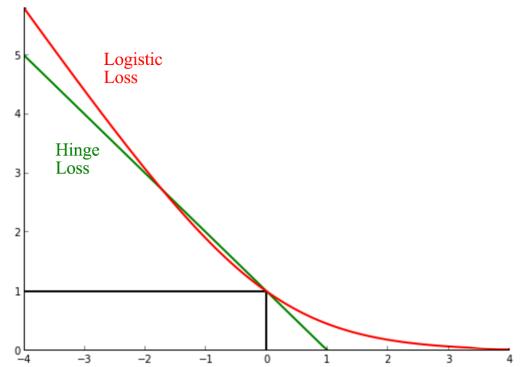


Figure 3.2: Binary, Hinge, and logistic loss for $y' = 1$

where σ is the logistic function

$$\sigma = z \mapsto \frac{1}{1 + e^{-z}}.$$

Therefore,

$$\nabla \hat{\mathcal{R}}_n(\beta) = \frac{1}{n} \sum_{i=1}^n X_i(\sigma(X_i^\top \beta) - Y_i) = \frac{1}{n} X(Y - \sigma(X\beta))$$

where $\sigma(X\beta)_i := \sigma(X_i^\top \beta)$. The problem is that the equation $\nabla \hat{\mathcal{R}}_n(\beta) = 0$ had no closed-form solution. Therefore, it needs to be solved through iterative algorithms (gradient descent, Newton's method, ...). Fortunately, this is possible, since the logistic loss is convex in its first argument. Indeed:

$$\frac{\partial^2 \ell_l(y, y')}{\partial y^2} = \sigma(y)\sigma(-y) > 0$$

Furthermore, the loss being strictly convex, the solution is even unique. In this chapter and in the next one, we will see tools and methods to solve convex optimization problems.

3.1.4 Regularization

Similarly to linear regression, logistic regression may over-fit the data (especially when $p > n$). In this case, one needs to add a regularization term, such as $\lambda \|\beta\|_2^2$ to the logistic loss.

3.2 Convex analysis

3.2.1 Convexity and minimization problems

We will now see notions of convex analysis to solve convex optimization problems, such as logistic regression. This chapter will introduce convex analysis – the properties of convex functions and convex optimization problems, and the next chapter will approach convex optimization algorithms (gradient descent, Newton's method, stochastic gradient descent, ...).

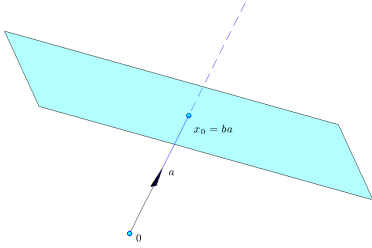


Figure 3.3: Hyperplane

Convexity is a crucial notion in many fields of mathematics and computer science. In machine learning, convexity creates well-defined problems with efficient solutions. A typical example is the problem of *empirical risk minimization*:

$$\hat{f}_n \in \arg \min_{f \in \mathcal{F}} \frac{1}{n} \sum_{i=1}^n \ell(f(X_i), Y_i) + \lambda \Omega(f)$$

where $D_n = \{ (X_i, Y_i) \mid i \in \llbracket 1, n \rrbracket \}$ is the data set, \mathcal{F} is a *convex* set of predictors $f : \mathcal{X} \rightarrow \mathbb{R}$, for all $y' \in \mathcal{Y}$, $y \mapsto \ell(y, y')$ is a convex loss function, and Ω is a convex penalty (such as $\|\cdot\|_2$, $\|\cdot\|_1$, ...).

Convexity will be useful to analyze both statistical properties of the solution \hat{f}_n and its generalization error:

$$\mathcal{R}(\hat{f}_n) := \mathbb{E}[\ell(f(X), Y) | D_n]$$

but also to derive efficient algorithms to solve the forementioned minimization problem and find \hat{f}_n .

3.2.2 Convex sets

In what follows, we will only consider finite dimensional Euclidean spaces (typically \mathbb{R}^d).

Definition (Convex set). A set $K \subseteq \mathbb{R}^d$ is convex if and only if:

$$\forall x, y \in K, \forall \alpha \in [0, 1], \quad \alpha x + (1 - \alpha)y \in K$$

Said otherwise, K is stable by barycentration, or, for all points $x, y \in K$, $[x, y] \subseteq K$.

Example. The following sets are convex:

- Hyperplans: $K = \left\{ x \in \mathbb{R}^d \mid a^\top x = b, a \neq 0, b \in \mathbb{R} \right\}$
- Half spaces: $K = \left\{ x \in \mathbb{R}^d \mid a^\top x \geq b, a \neq 0, b \in \mathbb{R} \right\}$
- Affine subspaces: $K = \left\{ x \in \mathbb{R}^d \mid Ax = b, A \in \mathcal{M}_d(\mathbb{R}), b \in \mathbb{R} \right\}$
- Balls: $\left\{ x \in \mathbb{R}^d \mid \|x\| \leq R \right\}$
- Cones: $K = \left\{ (x, r) \in \mathbb{R}^{d+1} \mid \|x\| \leq r \right\}$
- Convex polytopes: intersections of half spaces

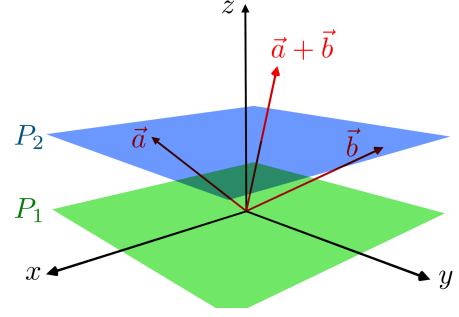


Figure 3.4: Affine space

We now announce useful properties of convex sets.

Property 3.1 (Stability by intersection). Convexity is stable by intersection. If $(K_i)_{i \in I}$ is a collection – non-necessarily countable – of convex sets, then:

$$\bigcap_{i \in I} K_i \quad \text{is convex}$$

Property 3.2 (Stability by affine transformation). Convexity is stable by affine transformation. If $K \subseteq \mathbb{R}^d$ is a convex set, then for all $\lambda \in \mathbb{R}$ and $\beta \in \mathbb{R}^d$,

$$\lambda \cdot K + \beta := \{ \lambda \cdot x + \beta \mid x \in K \} \quad \text{is convex}$$

Property 3.3 (Convex separation). If C, D are disjoint convex sets (i.e. $C \cap D = \emptyset$), then there exists a hyperplane which separates C and D :

$$\exists a \neq 0, b \in \mathbb{R}, \quad C \subseteq \left\{ x \in \mathbb{R}^d \mid a^\top x \geq b \right\}, D \subseteq \left\{ x \in \mathbb{R}^d \mid a^\top x \leq b \right\}$$

Moreover, the inequalities are strict if C and D are compact.

When a set is not convex, a trick is to use its convex hull to show some properties on it.

Definition (Convex Hull). Let $A \in \mathbb{R}^d$. The *Convex Hull* of A , denoted $\text{Conv}(A)$, is the smallest convex set that contains A . In other words,

$$\text{Conv}(A) := \bigcap \left\{ B \in \mathbb{R}^d \mid A \subseteq B, B \text{ convex} \right\}$$

$$= \left\{ \sum_{i=1}^p \alpha_i z_i \mid p \geq 1, \alpha \in \mathbb{R}_+^p, (z_1, \dots, z_p) \in A^p, \sum_{i=1}^p \alpha_i = 1 \right\}$$

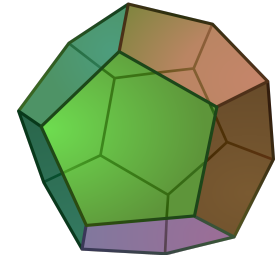


Figure 3.6: Polytope

3.2.3 Convex functions

Definition (Convex function). A function $f : D \subseteq \mathbb{R}^d \rightarrow \mathbb{R}$ where D is convex is a *convex function* when:

$$\forall x, y \in D, \forall \alpha \in [0, 1], \quad f(\alpha x + (1 - \alpha)y) \leq \alpha f(x) + (1 - \alpha)f(y) \quad (3.2.1)$$

Definition (Strictly convex function). A function $f : D \subseteq \mathbb{R}^d \rightarrow \mathbb{R}$ where D is convex is a *strictly convex function* when:

$$\forall x, y \in D, \forall \alpha \in [0, 1], \quad f(\alpha x + (1 - \alpha)y) < \alpha f(x) + (1 - \alpha)f(y) \quad (3.2.2)$$

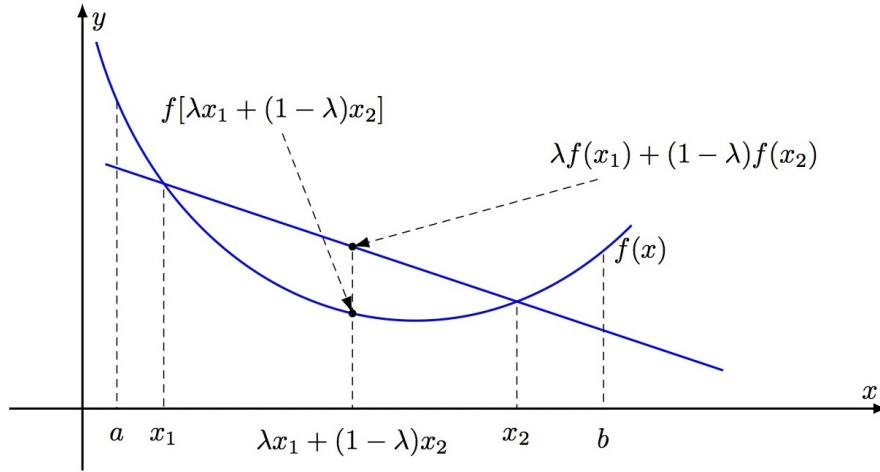


Figure 3.7: Geometric interpretation of convexity

Definition. A function $f : D \subseteq \mathbb{R}^d \rightarrow \mathbb{R}$ where D is convex is μ -*strictly convex* when

$$x \mapsto f(x) - \frac{\mu}{2} \|x\|^2 \quad \text{is convex}$$

Example (Convex functions). All the following functions are convex:

In dimension $d = 1$:

- $x \mapsto x$
- $x \mapsto x^2$
- $x \mapsto -\log(x)$
- $x \mapsto \log(1 + e^{-x})$
- $x \mapsto |x|^p$ for $p \geq 1$
- $x \mapsto -x^p$ for $p < 1$

Higher dimensions $d \geq 1$:

- Linear functions $x \mapsto aa^\top x$
- Quadratic functions $x \mapsto x^\top Qx$ for Q semidefinite symmetric positive matrix
- Norms
- $x \mapsto \max \{ x_i \mid i \in \llbracket 1, d \rrbracket \}$
- $x \mapsto \log \left(\sum_{i=1}^d e^{x_i} \right)$

In practice, the functions which we will consider will be \mathcal{C}^1 and even \mathcal{C}^2 . In this case, much simpler characterizations are given for convex functions.

Property 3.4 (Differentiable convex functions). If f is \mathcal{C}^1 ,

$$f \text{ convex} \iff \forall x, y \in D, f(x) \geq f(y) + f'(y)(x - y)$$

Property 3.5 (Twice differentiable convex functions). If f is twice differentiable,

f convex $\iff \forall x \in D$, its Hessian is semi-definite positive ($f''(x) \geq 0$)

Property 3.6 (Operations which preserve convexity). If $(f_i)_{i \in I}$ is a family of convex functions, then,

$$x \mapsto \sup_{i \in I} f_i(x) \quad \text{is convex}$$

$$\forall \alpha_i \in \mathbb{R}_+, x \mapsto \sum_{i \in I} \alpha_i f_i(x) \quad \text{is convex}$$

Furthermore, if f is convex on $C \times D$, then

$$y \mapsto \inf_{x \in C} f(x, y) \quad \text{is convex on } D$$

Property 3.7 (Convexity and continuity). If f is convex on D , then f is continuous on $\overset{\circ}{D}$. Furthermore, the epigraph of f ,

$$\{(x, t) \in D \times \mathbb{R} \mid f(x) \leq t\} \quad \text{is convex.}$$

Property 3.8 (Jensen's inequality). For f convex, $x_1, \dots, x_n \in D$ and $\alpha_1, \dots, \alpha_n \in \mathbb{R}_+$ such that $\sum_{i=1}^n \alpha_i = 1$. Then:

$$f\left(\sum_{i=1}^n \alpha_i x_i\right) \leq \sum_{i=1}^n \alpha_i f(x_i)$$

Property 3.9 (Jensen's integral inequality). For f convex, $p(x) \geq 0$ over $S \subseteq D$ such that $\int_S p(x) dx = 1$, then:

$$f\left(\int_S p(x) dx\right) \leq \int_S p(x) f(x) dx$$

Property 3.10 (Jensen's expectation inequality). For f convex, and X a random variable such that $X \in D$ almost surely and with $\mathbb{E}[X] < +\infty$, then:

$$f(\mathbb{E}[X]) \leq \mathbb{E}[f(X)]$$

3.2.4 Unconstrained optimization problems

Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be a convex function. We consider the following minimization problem:

$$\inf_{x \in \mathbb{R}^d} f(x)$$

Note that the notation $\min_x f(x)$ is only used when the minimum is reached. If no point achieves the minimum, we use the notation $\inf_x f(x)$.

Three cases are possible, depending on $\inf_x f(x)$:

- $\inf_{x \in \mathbb{R}^d} f(x) = -\infty$ – there is no minimum.
- $\inf_{x \in \mathbb{R}^d} f(x) > -\infty$ and the minimum is not reached.
- $\inf_{x \in \mathbb{R}^d} f(x) > -\infty$ and the minimum is reached and equals $\min_{x \in \mathbb{R}^d} f(x)$.

Definition (Local minimum). Let $f : D \rightarrow \mathbb{R}$ and $x \in D$. x is a local minimum if and only if there exists an open set $V \subseteq D$ such that $x \in V$ and $f(x) = \min_{x' \in V} f(x')$.

Property 3.11. If f is convex, any local minimum of f is a global minimum.

Property 3.12. If f is strictly convex, it has at most one minimum.

Property 3.13. If f is convex and \mathcal{C}^1 , then x is a minimum of f on \mathbb{R}^d if and only if $f'(x) = 0$.

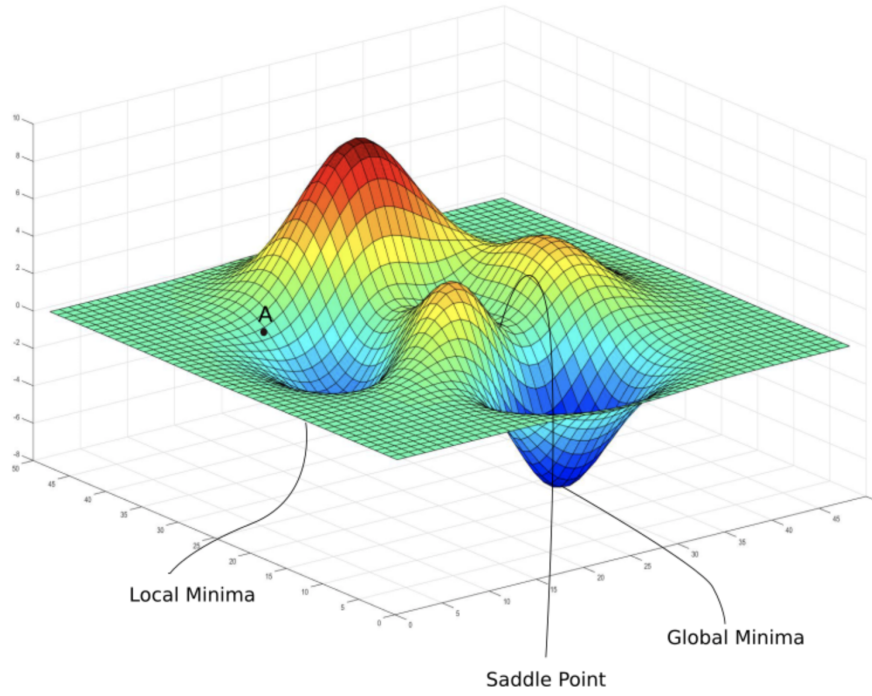


Figure 3.8: Local and global minima of a non-convex function

These three properties justify the approach of canceling the gradient to efficiently find a solution to a minimization problem. Property 3.11 guarantees that the result of some gradient descent is a global minimum, and not just some dead-end for the descent path. Strict convexity guarantees that every converging gradient descent will end up at the same minimum.

In the next chapter, we will continue the analysis of this frequent minimization problem and develop formal tools and algorithm to solve it efficiently.

4 Convex analysis and convex optimization

4.1 Constrained optimization problems

Let $f : D \mapsto \mathbb{R}^d$ convex and $C \subseteq D$ convex. We consider the constrained minimization problem

$$\inf_{x \in C} f(x)$$

where C is the constraint set. It is often defined as the intersection of sets of the form $\{h_i(x) = 0 \mid i \in I\}$ (hyperplanes) and $\{g_j(x) \leq 0 \mid j \in J\}$ (half-spaces).

Example (Minimization of a linear function over a compact). Let $A \subseteq \mathbb{R}^d$ be a compact, non-necessarily convex set, and $a \in \mathbb{R}^d \setminus \{0\}$. We can reformulate the non-convex minimization problem on A as a constrained convex optimization problem on $\text{Conv}(A)$:

$$\min \{a^\top x \mid x \in A\} = \min \{a^\top x \mid x \in \text{Conv}(A)\}$$

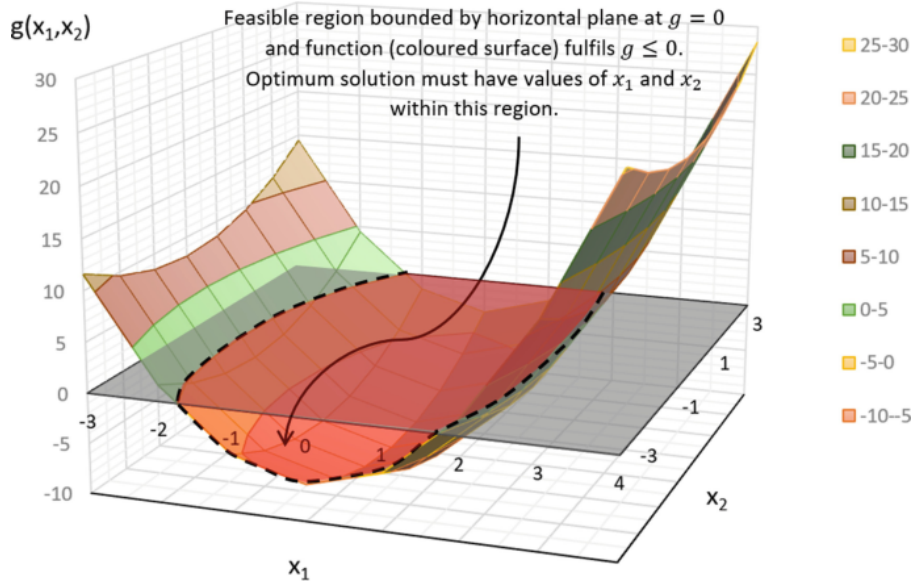


Figure 4.1: Constrained optimization: D^* is the red and orange part below the g plane

4.1.1 Lagrangian duality

A useful notion to solve constrained problems is Lagrangian duality. Assume that we are interested in the following constrained optimization problem:

$$\min_{x \in D} f(x) \quad \text{such that} \quad \begin{cases} h_i(x) = 0 & \text{for } i \in \llbracket 1, m \rrbracket \\ g_j(x) \leq 0 & \text{for } j \in \llbracket 1, r \rrbracket \end{cases} \quad (\text{P})$$

We denote by $D^* \subseteq D$ the set of points that satisfy the constraints:

$$D^* := \{x \in D \mid \forall i \in \llbracket 1, m \rrbracket, h_i(x) = 0 \wedge \forall j \in \llbracket 1, r \rrbracket, g_j(x) \leq 0\}$$

Note that the equality constraints $h_i(x) = 0$ can be rewritten as inequalities:

$$h_i(x) \leq 0 \wedge -h_i(x) \leq 0$$

Unlike unconstrained optimization problems, canceling the gradient does not necessarily provide a solution for constrained optimization problems. The basic idea of Lagrangian duality is therefore to take the constraint D^* into account in the minimization problem by augmenting the objective function with a weighted sum of the constraint functions.

Definition (Lagrangian). The *Lagrangian* associated to the optimization problem (P) is the function

$$\mathcal{L} : D \times \mathbb{R}^m \times \mathbb{R}_+^r \longrightarrow \mathbb{R}$$

defines by:

$$\mathcal{L}(x, \lambda, \mu) = f(x) + \lambda^\top h(x) + \mu^\top g(x)$$

Definition (Primal function). We define the *primal function* associated to (P)

$$\bar{f} : D \longrightarrow \mathbb{R} \cup \{+\infty\}$$

by:

$$\bar{f} : x \longmapsto \sup_{\lambda \in \mathbb{R}^m, \mu \in \mathbb{R}_+^r} \mathcal{L}(x, \lambda, \mu) = \begin{cases} f(x) & \text{if } x \in D^* \\ +\infty & \text{otherwise} \end{cases}$$

Definition (Primal problem). With these definitions, we observe that the optimization problem (P) can be re-written as a constraints-free problem of minimization of the primal function.

$$\begin{aligned} \inf_{x \in D^*} &= \inf_{x \in D} \bar{f}(x) \\ &= \inf_{x \in D} \sup_{\lambda \in \mathbb{R}^m, \mu \in \mathbb{R}_+^r} \mathcal{L}(x, \lambda, \mu) \end{aligned}$$

Definition (Dual problem). The *Dual problem* is obtained by exchanging inf and sup in the primal problem:

$$\sup_{\lambda \in \mathbb{R}^m, \mu \in \mathbb{R}_+^r} f^*(\lambda, \mu) := \sup_{\lambda \in \mathbb{R}^m, \mu \in \mathbb{R}_+^r} \inf_{x \in D} \mathcal{L}(x, \lambda, \mu)$$

where

$$f^* : (\lambda, \mu) \longmapsto \inf_{x \in D} \mathcal{L}(x, \lambda, \mu)$$

is the *dual function*. If f is convex, this function is concave. Note that the dual of the dual is the primal.

The admissibility domain of the primal is $D^* := \{x \in D \mid \bar{f}(x) < +\infty\}$, and the admissibility domain of the dual is $C^* := \{(\lambda, \mu) \in \mathbb{R}^m \times \mathbb{R}_+^r \mid f^*(\lambda, \mu) > -\infty\}$. There is no solution to the optimization problem when $D^* = \emptyset$. The problem is unbounded when $C^* = \emptyset$.

4.1.2 Link between primal and dual problems

The primal and dual problems are not necessarily identical, but have a strong relationship. For any (λ, μ) , $f^*(\lambda, \mu)$ provides a lower bound on the solution of (P). The dual problem finds the best lower bound.

Property 4.1 (Weak duality principle).

$$d^* := \sup_{\lambda \in \mathbb{R}^m, \mu \in \mathbb{R}_+^r} \inf_{x \in D} \mathcal{L}(x, \lambda, \mu) \leq \inf_{x \in D} \sup_{\lambda \in \mathbb{R}^m, \mu \in \mathbb{R}_+^r} \mathcal{L}(x, \lambda, \mu)$$

The solution of the dual problem is therefore always smaller than the solution of the primal. A good mnemonic is to remember this inequality as “the largest dwarf is always smaller than the smallest giant”.

Definition (Dual gap). The *dual gap* of the optimization problem is the difference between the primal and dual solutions:

$$p^* - d^* \geq 0$$

Definition (Strong duality). There is *strong duality* when $p^* = d^*$. In this case, the two problems are equivalent – they share the same solutions. The existence of the solutions are related with the existence of saddle point of the Lagrangian. Note that strong duality does not always hold.

4.1.3 Strong duality

Sometimes, the dual problem is easier to solve than the primal problem. It is then useful to know if there is strong duality.

Definition (Strictly feasible point). A point $x_0 \in D$ is *strictly feasible* when:

$$\exists x_0 \in D, \quad \begin{cases} h_i(x_0) = 0 & \forall i \in \llbracket 1, m \rrbracket \\ g_j(x_0) < 0 & \forall j \in \llbracket 1, r \rrbracket \end{cases}$$

Theorem (Slater's condition). If f and D are convex, h_i are affine and g_j are convex, then the existence of a strictly feasible point implies strong duality.

Example. Let $D = \mathbb{R}_+^d$ and consider the following linear programming minimization problem:

$$\min_{x \in D, Ax=b} c^\top x$$

where $A \in \mathcal{M}_{m,d}(\mathbb{R})$ and $b \in \mathbb{R}^m$. The constraints can be written as $Ax - b = 0$. Therefore, the Lagrangian is:

$$\mathcal{L} : (x, \lambda) \in \mathbb{R}_+^d \times \mathbb{R}^m \longrightarrow c^\top x + \lambda^\top (b - Ax)$$

The primal problem can be rewritten using the Lagrangian:

$$\begin{aligned} \min_{x \in D, Ax=b} c^\top x &= \min_{x \in D} \sup_{\lambda \in \mathbb{R}^m} c^\top x + \lambda^\top (b - Ax) \\ &= \min_{x \in D} \sup_{\lambda \in \mathbb{R}^m} b^\top \lambda + x^\top (c - A^\top \lambda) \end{aligned}$$

The problem is convex since the objective function is convex, and the equality constraints are affine. By Slater's condition, we can swap the min and the sup, giving:

$$\begin{aligned} \min_{x \in D, Ax=b} c^\top x &= \sup_{\lambda \in \mathbb{R}^m} \min_{x \in D} c^\top x + \lambda^\top (b - Ax) \\ &= \sup_{\lambda \in \mathbb{R}^m, A^\top \lambda \leq c} b^\top \lambda \end{aligned}$$

This is the dual formulation of the problem.

4.1.4 Karush-Kuhn-Tucker optimality conditions

We will now see conditions playing the same role as gradient cancelling for unconstrained optimization problems. These conditions will be useful to find equations to compute analytically the solutions of the minimization problem.

Assume that the function f , h_i , and g_i are all differentiable. Let x^* , and (λ^*, μ^*) be any primal and dual solutions, and assume that there is strong duality.

Property 4.2 (KKT1). Since x^* minimizes $\mathcal{L}(x, \lambda^*, \mu^*)$ over x , its gradient must be canceled at x^* :

$$\nabla f(x^*) + \sum_{i=1}^m \lambda_i^* \nabla h_i(x^*) + \sum_{j=1}^r \mu_j^* \nabla g_j(x^*) = 0 \quad (\text{KKT1})$$

Property 4.3 (KKT2). Since $x^* \in D^*$ and $(\lambda^*, \mu^*) \in C^*$ are feasible we have:

$$\begin{aligned} \forall i \in \llbracket 1, m \rrbracket, \quad h_i(x^*) &= 0 \\ \forall i \in \llbracket 1, r \rrbracket, \quad g_i(x^*) &\leq 0 \\ \forall j \in \llbracket 1, r \rrbracket, \quad \mu_j^* &\geq 0 \end{aligned} \quad (\text{KKT2})$$

Property 4.4 (KKT3). The *complementary condition* holds, i.e.:

$$\forall j \in \llbracket 1, r \rrbracket, \quad \mu_j^* g_j(x^*) = 0 \quad (\text{KKT3})$$

Proof. By contradiction, if the complementary condition does not hold, we could improve μ^* by setting $\mu_j^* = 0$, since $g_j(x^*) \leq 0$ and (λ^*, μ^*) maximizes $\mathcal{L}(x^*, \lambda, \mu)$. \square

These conditions are called the Karush-Kuhn-Tucker (KKT) conditions. When the primal problem is convex, these conditions are also sufficient.

Theorem (Karush–Kuhn–Tucker theorem). If there is strong duality, then:

$$(\text{KKT}) \text{ conditions are satisfied} \iff \begin{cases} x^* & \text{is a solution to the primal problem} \\ (\lambda^*, \mu^*) & \text{is a solution of the dual problem} \end{cases}$$

The KKT conditions play an important role in optimization. In some cases, it is possible to solve them analytically. Many optimization methods are conceived for solving the KKT conditions.

4.2 Optimization algorithms for unconstrained convex optimization

In this section we will see two widely used optimization algorithms for the problem of unconstrained optimization, *Gradient descent* and *Stochastic Gradient descent*.

Since the goal of minimizing a function $f : \mathbb{R}^d \rightarrow \mathbb{R}$ for $d \in \mathbb{N}$ is to find the point x for which the function has minimum value, the fundamental idea behind gradient descent consists in starting from a given $x_0 \in \mathbb{R}^d$ and finding the next point following a descent direction iteratively. In particular, we will consider f to be a convex function.

4.2.1 Gradient Descent

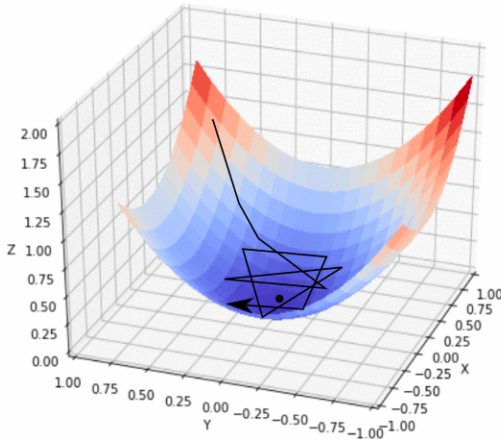


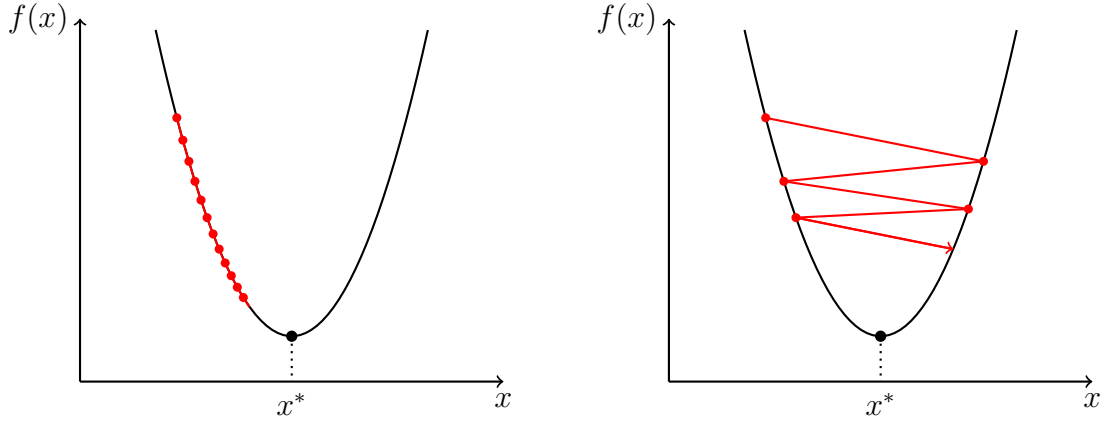
Figure 4.2: Convex gradient descent

When f is differentiable, the gradient of f in x , denoted $\nabla f(x)$, determines the direction of maximum increase of the function in a suitable neighborhood of x – and so $-\nabla f(x)$ determines the direction of maximum decrease of the function. The gradient descent algorithm then reads as follows:

$$\begin{cases} x_0 & \text{given} \\ x_{t+1} := x_t - \gamma_t \nabla f(x_t) & \forall t \in \llbracket 1, T \rrbracket \end{cases}$$

where γ_n is denoted as *step-size* and is small enough such that $-\gamma_t \nabla f(x_t)$ is still a decrease direction in the neighborhood of x_t .

The choice of γ_t is crucial for the optimization algorithm. If γ_t is too big, it makes the algorithm unstable and possibly diverge, since it follows the direction $-\nabla f(x_t)$ out of the region where it is a descent direction. On the other hand, if γ_t is too small, the chosen direction is a descent direction, but each step is very short, leading to a larger number of steps required to arrive to the minimum solution – with a big impact on the total computational complexity.



Too small: converges very slowly

Too big: overshoots and diverges

Figure 4.3: Choice of step size

We will prove in the next theorem that there exists a step-size that guarantees the convergence of the solution of the gradient descent algorithm to the minimizer of f , and we characterize how fast gradient descent achieves it.

4.2.2 Gradient Descent for strongly convex functions

Definition (L -Lipschitz continuous gradients). For $L > 0$, f has L -Lipschitz continuous gradients when:

$$\forall x, y \in \mathbb{R}^d, \quad \|\nabla f(x) - \nabla f(y)\| \leq L\|y - x\|$$

Lemma 4.1. Let $L > 0$ and $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be a convex function with L -Lipschitz continuous gradients, then:

$$f(y) \leq f(x) + \nabla f(x)^\top (y - x) + L\|y - x\|^2 \quad (4.2.1)$$

Proof.

□

Remark. Using a different argument, it is possible to prove the tighter result:

$$f(y) \leq f(x) + \nabla f(x)^\top (y - x) + \frac{L}{2}\|y - x\|^2 \quad (4.2.2)$$

Lemma 4.2 (Gradient descent is a descent algorithm with $\gamma_t \in]0, 1/L[$). Let f be convex with L -Lipschitz gradient, let $x_0 \in \mathbb{R}^d$ and $\gamma_t > 0$, then:

$$\forall t \in \mathbb{N}^*, \quad f(x_t) \leq f(x_{t-1}) - \gamma_t(1 - L \cdot \gamma_t) \cdot \|\nabla f(x_{t-1})\|^2$$

In particular, if $\gamma_t \in]0, 1/L[$, we have that $\gamma_t(1 - L \cdot \gamma_t) > 0$. Therefore, for all $t \in \mathbb{N}^*$:

$$f(x_t) < f(x_{t-1})$$

whenever x_{t-1} is not a global optimum, otherwise $x_t = x_{t-1}$ and $f(x_t) = f(x_{t-1})$ (a fixpoint is reached).

Proof.

□

Recall that when f is μ -stricly convex, we have:

$$f(y) \geq f(x) + \nabla f(x)^\top (y - x) + \frac{\mu}{2} \|y - x\|^2$$

This will be used in the next theorem.

Theorem. Let $f : \mathbb{R}^d \rightarrow \mathbb{R}$ be a μ -stricly convex function with L -Lipschitz continuous gradients. Let $\gamma \in]0, 1/2L[$, and choose a constant step-size $\gamma_t := \gamma$ for $t \in \mathbb{N}$. Denote by x^* the global optimum of f . Finally, let $x_0 \in \mathbb{R}^d$ and $T \in \mathbb{N}$. Then:

$$\|x_T - x^*\|^2 \leq (1 - \gamma\mu)^T \|x_0 - x^*\|^2$$

Proof. □

Remark. Using the tighter inequality (4.2.2), the previous result can be extended to $\gamma \in]0, 1/L[$.

Note that since:

$$(1 - \gamma\mu)^T \leq e^{-\gamma\mu T}$$

the choice of $T = \frac{\gamma\mu}{\log}(\|x_0 - x^*\|^2/\varepsilon)$ gives:

$$\|x_T - x^*\|^2 \leq \varepsilon$$

Corollary. Under the same hypothesis on f , γ , we have:

$$\|x_T - x^*\| \xrightarrow{T \rightarrow +\infty} 0 \tag{4.2.3}$$

Therefore, in this case, the Gradient Descent algorithm converges.

4.2.3 Stochastic Gradient Descent

In this section, we will introduce Stochastic Gradient Descent. In some cases, especially when n is big, computing the actual gradient of the empirical risk function $\hat{\mathcal{R}}_n$ is extremely costly. The approach of SGD is therefore to replace the exact gradient by a stochastic approximation of it, in practice calculating it from a random subset of the training data. This enables faster iterations, at the cost of a slower convergence rate.

The algorithm is useful to find the global minimum of conex functions of the form

$$f(x) = \mathbb{E}_\theta [g(x, \theta)], \quad \text{where} \quad \mathbb{E}_\theta [g(x, \theta)] = \int_\Omega g(x, \theta) \, \mathrm{d}p(\theta)$$

where p is a probability distribution over Ω and $g : \mathbb{R}^d \times \Omega \rightarrow \mathbb{R}$.

Example (Empirical Risk Minimization). We consider f of the form:

$$f(x) = \frac{1}{n} \sum_{i=1}^n L(x, \theta_i), \quad \text{where} \quad \begin{cases} \theta_i = (z_i, y_i) \\ L(x, \theta_i) = \ell(x^\top z_i, y_i) \end{cases}$$

In this context, we can assume that due to the size of Ω (in the previous example, n), we are not able to evaluate the integral above, but we assume that we are able to sample some θ_t from p and to compute the gradient $\nabla_x g(x, \theta_t)$.

The algorithm is similar to classical gradient descent, but the iteration step is changed to:

$$x_t = x_{t-1} - \gamma_t \nabla_x g(x, \theta_t) \quad \text{where} \quad \theta_t \sim p$$

As previously, γ_t is a sequence of step sizes and x_0 is given; moreover, specifically for stochastic GD, θ_t is independently and identically distributed according to p .

Note that, by linearity of the integral $\nabla f(x) = \mathbb{E}_\theta [\nabla_x g(x, \theta)]$, thus:

$$\begin{aligned}\mathbb{E}_\theta[x_t] &= x_{t-1} - \gamma_t \nabla_x g(x_{t-1}, \theta_t) \\ &= x_{t-1} - \gamma_t \nabla f(x_{t-1})\end{aligned}$$

Intuitively, the SGD seems to behave in expectation like gradient descent. Let's analyze this property in more details with the following theorem.

Definition (Variance of the estimator of the gradient). We define the *variance of the estimator of the gradient* as:

$$\sigma^2(x) := \mathbb{E}_\theta [\|\nabla f(x) - \nabla_x g(x, \theta)\|^2]$$

Theorem (Expectation bound for SGD). Assuming that $f : \mathbb{R}^d \rightarrow \mathbb{R}$ is μ -strictly convex and with L -Lipschitz continuous gradients, let $\gamma_t = \gamma$ for $t \in \mathbb{N}$, with $\gamma \in]0, 1/2L[$. Assume that there exists σ_*^2 such that

$$\forall x \in \mathbb{R}^d, \quad \sigma^2(x) \leq \sigma_*^2$$

Under such assumptions, we have that:

$$\mathbb{E}_{\theta_1, \dots, \theta_T} [\|x_T - x^*\|^2] \leq (1 - \mu\gamma)^T \|x_0 - x^*\|^2 + \frac{\gamma}{\mu} \sigma^2$$

Proof.

□

Corollary (Convergence of SGD). Under the same assumptions:

$$\mathbb{E}_{\theta_1, \dots, \theta_T} [\|x_T - x^*\|^2] \xrightarrow{T \rightarrow +\infty} 0$$

Hence, stochastic gradient descent converges in expectation towards the global minimum.

5 Kernels

5.1 Introduction to kernels

In this course, we often focused on prediction methods which are *linear*, that is, the input data are vectors and the prediction function is linear (e.g. $f(x) = w^\top x$ for $w \in \mathbb{R}^d$). In this situation with given data (x_i, y_i) , the vector w can be obtained by minimizing

$$\hat{L}(w) = \frac{1}{n} \sum_{i=1}^n l(y_i, w^\top x_i) + \lambda \Omega(w)$$

Classical examples are logistic regression or least-squares regression. These methods look at first sight of limited practical significance, because input data may not be vectors, and relevant prediction functions may not be linear.

The goal of kernel methods is therefore to go beyond these limitations while keeping the good aspects. The underlying principle is to replace x by a function $\varphi(x) \in \mathbb{R}^d$, *explicitly* or *implicitly*, and consider linear predictions in $\Phi(x)$, i.e. $f(x) = w^\top \varphi(x)$. We call $\varphi(x)$ the *feature* associated to x .

Example (Linear regression). In the case of linear regression, $\varphi(x) = x$ for $x \in \mathbb{R}^d$. As expected, this gives us linear models:

$$f(x) = w^\top x = \sum_{j=1}^d w_j x_j$$

Example (Polynomial regression of degree r). With $x \in \mathbb{R}$, we have $\varphi(x) \in \mathbb{R}^{r+1}$ defined by:

$$\varphi(x) = (1, x, x^2, \dots, x^r)$$

Therefore, the prediction functions will be general polynomials of degree at most r :

$$f(x) = w^\top \varphi(x) = \sum_{j=0}^r (\varphi(x))_j = \sum_{j=1}^r w_j x^j$$

Example (Polynomial multivariate regression of degree r). We consider $x \in \mathbb{R}^d$ and

$$\varphi(x) = (x_1^{\alpha_1}, \dots, x_d^{\alpha_d}) \quad \text{where} \quad \sum_{i=1}^d \alpha_i = r$$

In this situation, $p = \binom{d+r-1}{r}$ might be too big for an explicit representation to be feasible.

Example (Generic set of functions). Let $\phi_1, \phi_r : \mathbb{R}^d \rightarrow \mathbb{R}$ be a set of functions of interest (e.g. a subset of the Fourier basis); we define $\varphi(x) = (\phi_1(x), \dots, \phi_r(x))$ to have:

$$f(x) = w^\top \varphi(x) = \sum_{j=1}^r w_j \phi_j(x)$$

5.2 Representer theorem

5.2.1 Theorem statement

Given a dataset x_1, \dots, x_n , let φ be a feature map, such that we are able to compute the observed feature maps $\varphi(x_1), \dots, \varphi(x_n)$. We can ask ourselves if there exists an easier representation for w in terms of these observed feature maps, i.e. we want to know if it is possible to characterize the minimum \hat{w} of:

$$\hat{L}(w) = \frac{1}{n} \sum_{i=1}^n l(y_i, w^\top \varphi(x_i)) + \lambda w^\top w$$

in the form of $\hat{w} = \sum_{i=1}^n \alpha_i \varphi(x_i)$, with $\alpha_i \in \mathbb{R}$. The following theorem guarantees such characterization under basic properties of \hat{L} .

Theorem (Representer theorem). Let $\varphi : \mathcal{X} \rightarrow \mathbb{R}^d$. Let $(x_1, \dots, x_n) \in \mathcal{X}^n$ and assume that $\Psi : \mathbb{R}^{n+1} \rightarrow \mathbb{R}$ is strictly increasing with respect to the last variable. Then, the minimum of

$$\hat{L}(w) := \Psi(w^\top \varphi(x_1), \dots, w^\top \varphi(x_n), w^\top w)$$

is obtained for

$$w = \sum_{i=1}^n \alpha_i \varphi(x_i)$$

for some $\alpha \in \mathbb{R}^n$.

Proof.

□

Corollary. For $\lambda > 0$,

$$\min_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^n \ell(y_i, w^\top \varphi(x_i)) + \frac{\lambda}{2} w^\top w$$

is obtained for

$$w = \sum_{i=1}^n \alpha_i \varphi(x_i).$$

Note that there is no assumption on ℓ , and in particular, no convexity assumption. This result is extendable to Hilbert spaces (RKHS⁸), as we will see in the next section.

5.2.2 Finite dimensional representation of the learning problem

Using the representer theorem, we know that the minimum of \hat{L} is of the form $w = \sum_{i=1}^n \alpha_i \varphi(x_i)$; we can therefore directly optimize this characterization, i.e. we can then write:

$$\min_{w \in \mathbb{R}^r} \frac{1}{n} \sum_{i=1}^n \ell(y_i, w^\top \varphi(x_i)) + \frac{\lambda}{2} w^\top w = \min_{\alpha \in \mathbb{R}^n} \frac{1}{n} \sum_{i=1}^n \ell(y_i, (K\alpha)_i) + \frac{\lambda}{2} \alpha^\top K \alpha$$

where $K \in \mathcal{M}_n(\mathbb{R})$ with values

$$K_{i,j} = \varphi(x_i)^\top \varphi(x_j).$$

Indeed,

$$\varphi(x_i)^\top w = \sum_{j=1}^n \alpha_j \varphi(x_i)^\top \varphi(x_j) = (K\alpha)_i$$

moreover,

$$\|w\|^2 = w^\top w = \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j \varphi(x_i)^\top \varphi(x_j) = \alpha^\top K \alpha$$

We finally have a closed form representation for the function evaluation. Defining the *kernel function* $k(x, x') := \varphi(x)^\top \varphi(x')$, we have:

$$f(x) = w^\top \varphi(x) = \sum_{i=1}^n \alpha_i \varphi(x_i)^\top \varphi(x) = \sum_{i=1}^n \alpha_i k(x_i, x).$$

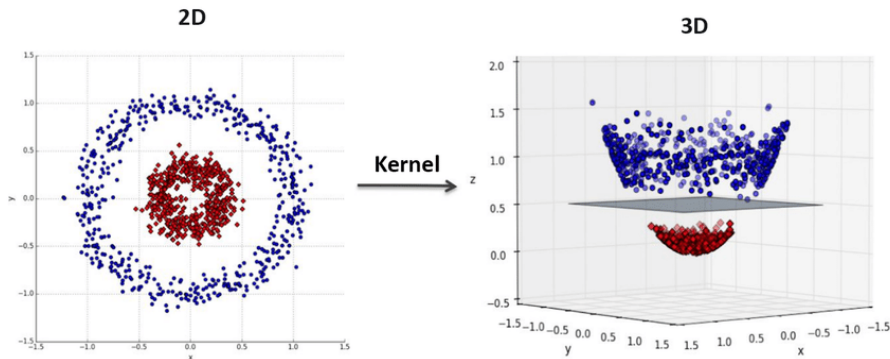


Figure 5.1: Use of a kernel to linearly separate data

⁸Reproducing kernel Hilbert space, a specific kind of Hilbert space which is often used in machine learning.

Remark (Kernel trick). *The whole learning problem can be written in terms of the kernel k ; indeed, f depends only on k , \hat{L} depends on K with $K_{i,j} = k(x_i, x_j)$. Therefore, we have the so-called kernel trick, i.e. we do not need to compute explicitly the features φ to be able to represent and solve the learning problem, we just need to be able to compute their inner product.*

Example (Power of the kernel trick with infinite dimensional feature maps).

Consider $\mathcal{X} = [-1, 1]$ and the feature map

$$\varphi(x) = (1, x, x^2, \dots).$$

The resulting model space would have the form

$$f(x) = \sum_{j=0}^{+\infty} w_j x^j$$

with $\sum_{j=0}^{+\infty} w_j^2 < +\infty$. This model space is the set of analytic function on \mathcal{X} , which is a very rich space. In particular, it is dense in the space of continuous functions. However, it is not possible to compute $\varphi(x)$ explicitly since it is infinite dimensional. The kernel trick provides an elegant way to compute the solution of the learning problem in closed form; indeed, the inner product can be computed in closed form in $O(1)$:

$$k(x, x') = \varphi(x)^\top \varphi(x') = \sum_{j=0}^{+\infty} x^j x'^j = \frac{1}{1 - xx'}$$

Therefore, the kernel trick allow to replace \mathbb{R}^d by \mathbb{R}^n , which is interesting when d is very large. Furthermore, it allows to separate the representation problem (design a kernel on a set \mathcal{X}), algorithms, and analysis (which only use the kernel matrix K).

5.3 Properties of kernels

Since the learning problem is completely defined in terms of the kernel function, the explicit knowledge of the feature map is not required anymore. In particular, given a function $k : X \times X \rightarrow \mathbb{R}$, to use it in a learning problem, we need to be sure that it is a *positive definite kernel*, i.e. that there exists a feature map φ such that

$$\forall x, x' \in X, \quad k(x, x') = \varphi(x)^\top \varphi(x')$$

Kernel functions admits many characterizations, which we will now present.

Property 5.1 (Characterization in terms of positive-definiteness). k is a positive definite kernel if and only if the kernel matrix k is positive semi-definite (i.e. all its eigenvalues are non-negative).

Theorem (Aronszajn). k is a positive definite kernel if and only if there exists a Hilbert space \mathcal{F} , and $\varphi : X \rightarrow \mathcal{F}$ such that

$$\forall x, y \in X, \quad k(x, y) = \langle \varphi(x), \varphi(y) \rangle$$

If such objects exist, \mathcal{F} is called the *feature space* and φ the *feature map*.

Property 5.2. The sum and product of kernels are kernels.

Example (Linear kernel). The linear kernel corresponds to $\varphi = x \mapsto x$:

$$k(x, y) = x^\top y$$

Example (Polynomial kernel). The kernel $k(x, y) = (x^\top y)^r$ can be expanded as:

$$k(x, y) = \left(\sum_{i=1}^d x_i y_i \right)^r = \sum_{\alpha_1 + \dots + \alpha_p = r} \binom{r}{\alpha_1, \dots, \alpha_p} \underbrace{(x_1 y_1)^{\alpha_1} \dots (x_p y_p)^{\alpha_p}}_{(x_1^{\alpha_1} \dots x_p^{\alpha_p})(y_1^{\alpha_1} \dots y_p^{\alpha_p})}$$

Example (Translation-invariant kernels on a bounded interval).

Example (Translation-invariant kernels on \mathbb{R}^d).

6 Elements of Statistical Machine Learning

In this class we will introduce the main elements of PAC Learning. Probably Approximately Correct Learning is a theoretical and mathematical framework for analysing machine learning algorithms. Such framework allows, for instance, to evaluate the complexity of a problem in the context of supervised learning. It was introduced by Leslie Valiant in 1984.

6.1 Introduction

Recall that a *learning algorithm* – or *learning rule* – is a function \mathcal{A} that maps a training set D_n to an estimator $\hat{f}_n : \mathcal{X} \rightarrow \mathcal{Y}$:

$$\mathcal{A} : \bigcup_{n \in \mathbb{N}} (\mathcal{X} \times \mathcal{Y})^n \mapsto \mathcal{Y}^{\mathcal{X}}$$

We denote $\hat{f}_n := \mathcal{A}(D_n)$ the estimator, which is a random variable in $\mathcal{Y}^{\mathcal{X}}$, since it depends on the dataset D_n .

Remark. Sometimes, the prediction set can differ from the actual output set. For instance, binary classification in $\{0, 1\}$ can be replaced by predictions in $[0, 1]$. We used this technique previously by introducing Hinge and Logistic losses to extend the binary loss and obtain a convex optimization problem.

Definition (Fundamental problem of Supervised Learning). Given a dataset D_n , a loss function ℓ , the goal of supervised learning is to estimate f^* that satisfies:

$$f^* \in \arg \min_{f: \mathcal{X} \rightarrow \mathcal{Y}} R(f)$$

Recall the *excess risk*, defined in (2.4.3) by:

$$\mathcal{R}(\hat{f}_n) := \mathcal{R}(\hat{f}_n) - \mathcal{R}(f^*)$$

It measures how close given predictor \hat{f}_n is to the best possible f^* , in terms of expected risk \mathcal{R} , i.e. in terms of *average error on new examples*. Both $\mathcal{R}(\hat{f}_n)$ and $\mathcal{R}(\hat{f}_n)$ are random variables, since \hat{f}_n depends on the dataset D_n .

Definition (Consistency). Let $\delta \in]0, 1]$. The algorithm \mathcal{A} is *consistent* (i.e. it is a proper learning algorithm) when:

$$\lim_{n \rightarrow +\infty} \mathbb{E}_{D_n}[\mathcal{R}(\hat{f}_n)] = 0$$

Definition (Strong consistency). An algorithm \mathcal{A} is strongly consistent when the equation above holds with probability 1:

$$\lim_{n \rightarrow +\infty} \mathcal{R}(\hat{f}_n) = 0$$

Not all algorithms learn at the same rate: for a given $n \in \mathbb{N}^*$, some algorithms might have more precise predictions than others. We can define more quantitative versions of the requirements above, which are useful to characterize how precise are the predictions.

Definition (Learning rates). The sequence $(e_n)_{n \in \mathbb{N}} \in \mathbb{R}_+^{\mathbb{N}}$ is a learning rate in expectation if:

$$\forall n \in \mathbb{N}, \quad \mathbb{E}_{D_n}[\mathcal{R}(\hat{f}_n)] \leq e_n$$

Given $\delta \in]0, 1]$, a sequence $(p_{n,\delta})_{n \in \mathbb{N}} \in [0, 1]^{\mathbb{N}}$ is a learning rate in probability if:

$$\forall n \in \mathbb{N}, \quad \mathbb{P}_{D_n}(\mathcal{R}(\hat{f}_n) > p_{n,\delta}) \leq \delta$$

6.2 Empirical Risk Minimization

A classical way to estimate f^* is via *empirical risk minimization*. This is the approach we used in the specific examples of linear and logistic regressions. We will now define it more formally.

Let \mathcal{F} be a set of functions called *function space* containing some candidate estimators of choice. The estimator is defined as:

$$\hat{f}_n := \arg \min_{f \in \mathcal{F}} \hat{\mathcal{R}}_n(f) \quad \text{where} \quad \hat{\mathcal{R}}_n(f) = \frac{1}{n} \sum_{i=1}^n \ell(y_i, f(x_i))$$

As before, $\hat{\mathcal{R}}_n$ is the empirical risk, which measures the average error performed by f on the training set. The intuition is that $\hat{\mathcal{R}}_n(f)$ approximates $\mathcal{R}(f)$ – the expected error – increasingly better when n goes to infinity. A crucial question we need to address is to understand under which conditions *empirical risk minimization* is a learning algorithm and has learning rates.

6.3 Preliminary results

We will first recall some results that will be useful for the proof.

6.3.1 Results on random variables

Lemma 6.1 (Union bound). Let \mathcal{F} be a finite set indexing a family of sets $(A_f)_{f \in \mathcal{F}}$, then:

$$\mathbb{P}\left(\bigcup_{f \in \mathcal{F}} A_f\right) \leq \sum_{f \in \mathcal{F}} \mathbb{P}(A_f)$$

Lemma 6.2 (Supremum of random variables). Let $t > 0$ and \mathcal{F} be a finite set indexing real random variables $(u_f)_{f \in \mathcal{F}}$, we have:

$$\mathbb{P}(\sup_{f \in \mathcal{F}} |u_f| > t) \leq \sum_{f \in \mathcal{F}} \mathbb{P}(|u_f| > t)$$

Property 6.1. Let X be a random variable and let f, g be real functions, with $t \in \mathbb{R}$ such that $f(X) \geq g(X) > t$ almost surely, then:

$$\mathbb{P}(g(X) > t) \leq \mathbb{P}(f(X) > t)$$

6.3.2 Bernstein's inequality

Lemma 6.3 (Exponential moments of bounded random variables). Let u be a random variable such that $|u| \leq B$, for $B > 0$ almost surely and $\mathbb{E}[u] = 0$. Define $\sigma^2 := \mathbb{E}[u^2]$. Let $0 \leq \theta < B^{-1}$, then:

$$\mathbb{E}[e^{\theta u}] \leq \exp\left(\frac{\theta^2 \sigma^2}{2(1 - \theta B)}\right)$$

Lemma 6.4 (Bernstein inequality for random variables). Let U_1, \dots, U_n be independently and identically distributed random variables, such that $\mathbb{E}[u] = 0$ and $|u| \leq B$ almost surely, for $B > 0$. Define $\sigma^2 := \mathbb{E}[u^2]$. Let $t > 0$. The following holds:

$$\mathbb{P}\left(\frac{1}{n} \sum_{i=1}^n U_i > t\right) \leq \exp\left(-\frac{t^2 n}{2(\sigma^2 + Bt)}\right) \quad (6.3.1)$$

Corollary. Under the same assumption as the previous lemmas,

$$\mathbb{P}\left(\left|\frac{1}{n} \sum_{i=1}^n U_i\right| > t\right) \leq 2 \exp\left(-\frac{t^2 n}{2(\sigma^2 + Bt)}\right)$$

6.4 Consistency and Learning rates for Empirical Risk Minimization

6.4.1 Introduction

The goal of this section is to prove consistency and learning rates for empirical risk minimization. A key step in this process is to decompose the excess risk as:

$$\mathcal{R}(\hat{f}_n) := \mathbb{E}[\mathcal{R}(\hat{f}_n)] - \mathcal{R}(f^*) = \underbrace{\left(\mathbb{E}[\mathcal{R}(\hat{f}_n)] - \inf_{f \in \mathcal{F}} \mathcal{R}(f)\right)}_{\text{Estimation error}} + \underbrace{\left(\inf_{f \in \mathcal{F}} \mathcal{R}(f) - \mathcal{R}(f^*)\right)}_{\text{Approximation error}}$$

The leftmost term is called the *variance term* or *estimation error*. It depends on D_n , \mathcal{F} , \hat{f}_n , and can be bounded with mild or no assumption on the data distribution ν . We are going to prove such results in this chapter.

The rightmost term is called the *bias term* or *approximation error*, and depends on f^* and \mathcal{F} , but not on \hat{f}_n or D_n . To control it, we must make some assumptions on ν . While it is possible to prove its consistency without assumptions, hypothesis are needed to guarantee rates of convergence.

The picture below shows the effect of the capacity $|\mathcal{F}|$ on the variance term, bias term and total excess risk, for a fixed dataset. In particular, one can observe two important regimes: *underfitting* and *overfitting*.

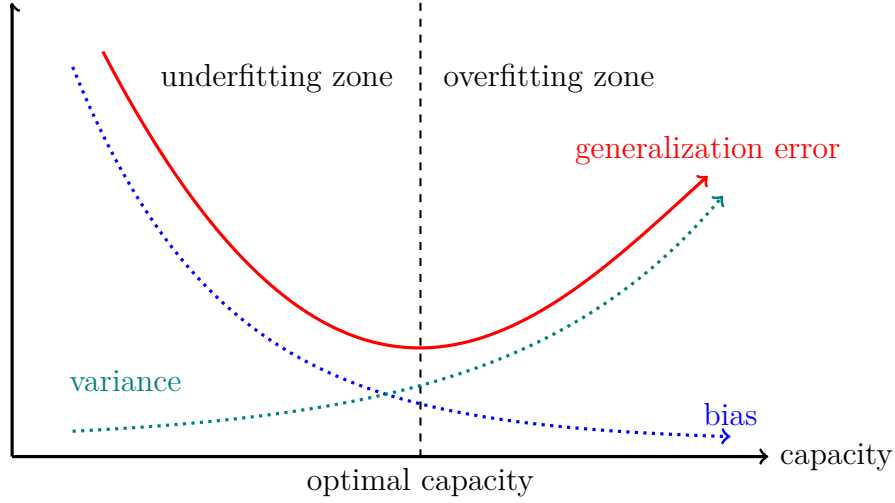


Figure 6.1: Effect of the capacity on bias and variance

The regime is in a regime of *underfitting* when the capacity of the hypothesis space is too small, i.e. the chosen functions are not able to approximate well f^* , but at the same time there are only few functions in \mathcal{F} , the variance is therefore small.

On the other side, when we select a very large hypothesis space, there likely exists a function $f \in \mathcal{F}$ very close to f^* , inducing a small bias – but at the same time, since $|\mathcal{F}|$ is big, the variance is large.

Obviously, there are cases that are not represented by the graph above, consider for example the lucky case $\mathcal{F} = \{f^*\}$, or more generally $\mathcal{F} = \{f^*\} \cup \mathcal{F}_0$.

6.4.2 Bounding the variance term: PAC bounds

As explained above, the estimator \hat{f}_n is a random variable. A way to deal with this randomness is to consider the expectation of $\mathcal{R}(\hat{f}_n)$, but this is limited: it only makes statements about the risk on average. A more precise control over the excess risk can be stated in terms of a probabilistic statement: a PAC – Probably Approximately Correct – bound.

Definition $((\varepsilon, \delta)$ -PAC). We say that \hat{f}_n is ε -accurate with confidence $1 - \delta$ of (ε, δ) -PAC when:

$$\mathbb{P}_{D_n} \left(\mathcal{R}(\hat{f}_n) - \inf_{f \in \mathcal{F}} \mathcal{R}(f) > \varepsilon \right) < \delta \quad (6.4.1)$$

Note that:

$$\mathcal{R}(\hat{f}_n) - \inf_{f \in \mathcal{F}} \mathcal{R}(f) = \underbrace{\left[\mathcal{R}(\hat{f}_n) - \inf_{f \in \mathcal{F}} \hat{\mathcal{R}}_n(f) \right]}_{\rho_1} + \underbrace{\left[\inf_{f \in \mathcal{F}} \hat{\mathcal{R}}_n(f) - \inf_{f \in \mathcal{F}} \mathcal{R}(f) \right]}_{\rho_2}$$

By the definition of \hat{f}_n , which is the minimum of $\hat{\mathcal{R}}_n(\hat{f}_n)$ over \mathcal{F} , then for the first term ρ_1 we have:

$$\rho_1 := \mathcal{R}(\hat{f}_n) - \inf_{f \in \mathcal{F}} \hat{\mathcal{R}}_n(f) = \mathcal{R}(\hat{f}_n) - \hat{\mathcal{R}}_n(\hat{f}_n) \leq \sup_{f \in \mathcal{F}} |\mathcal{R}(f) - \hat{\mathcal{R}}_n(f)|$$

For the second term ρ_2 , using the inverse triangular inequality, we have:

$$\rho_2 := \inf_{f \in \mathcal{F}} \hat{\mathcal{R}}_n(f) - \inf_{f \in \mathcal{F}} \mathcal{R}(f) \leq \sup_{f \in \mathcal{F}} |\mathcal{R}(f) - \hat{\mathcal{R}}_n(f)|$$

Therefore,

$$\mathcal{R}(\hat{f}_n) - \inf_{f \in \mathcal{F}} \mathcal{R}(f) = \rho_1 + \rho_2 \leq 2 \sup_{f \in \mathcal{F}} |\mathcal{R}(f) - \hat{\mathcal{R}}_n(f)|$$

6.4.3 Bounds in probability

In the following, we assume that \mathcal{F} is finite and ℓ is bounded, that is:

$$|\mathcal{F}| < +\infty \quad \text{and} \quad \exists M, \forall y, y' \in \mathcal{Y}, \quad |\ell(y, y')| \leq M$$

Under these assumptions, for all $t > 0$, we have by Property 6.1:

$$\begin{aligned} \mathbb{P} \left(\mathcal{R}(\hat{f}_n) - \inf_{f \in \mathcal{F}} \mathcal{R}(f) > 2t \right) &\leq \mathbb{P} \left(2 \sup_{f \in \mathcal{F}} |\mathcal{R}(f) - \hat{\mathcal{R}}_n(f)| > 2t \right) \\ &\leq \sum_{f \in \mathcal{F}} \mathbb{P} \left(2 |\mathcal{R}(f) - \hat{\mathcal{R}}_n(f)| > 2t \right) \\ &= \sum_{f \in \mathcal{F}} \mathbb{P} \left(|\mathcal{R}(f) - \hat{\mathcal{R}}_n(f)| > t \right) \end{aligned}$$

where the last step is due to Lemma 6.2. To further bound the inequality above, we need to study the probability of the even $|\mathcal{R}(f) - \hat{\mathcal{R}}_n(f)| > t$. Given $f \in \mathcal{F}$, let:

$$\forall i \in \llbracket 1, n \rrbracket, \quad v_i := \ell(y_i, f(x_i)) - \mathcal{R}(f)$$

Therefore, we have:

$$|\hat{\mathcal{R}}_n(f) - \mathcal{R}(f)| = \left| \frac{1}{n} \sum_{i=1}^n v_i \right|$$

Notice that $\mathbb{E}[v_i] = 0$ and that the $(v_i)_i$ are independent and identically distributed. Moreover, using the assumption that ℓ is bounded, we have that $|v_i| \leq M$ almost surely and that $\mathbb{E}[v_i^2] \leq M^2$. Therefore, applying the Bernstein inequality, we have:

$$\mathbb{P} \left(\left| \frac{1}{n} \sum_{i=1}^n v_i \right| > t \right) \leq 2 \exp \left(- \frac{t^2 n}{2(M^2 + Mt)} \right)$$

Therefore,

$$\begin{aligned} \mathbb{P}_{D_n} \left(\mathcal{R}(\hat{f}_n) - \inf_{f \in \mathcal{F}} \mathcal{R}(f) > 2t \right) &\leq \mathbb{P}_{D_n} \left(2 \sup_{f \in \mathcal{F}} |\mathcal{R}(f) - \hat{\mathcal{R}}_n(f)| > 2t \right) \\ &\leq \sum_{f \in \mathcal{F}} \mathbb{P}_{D_n} \left(2 \sup_{f \in \mathcal{F}} |\mathcal{R}(f) - \hat{\mathcal{R}}_n(f)| > 2t \right) \\ &\leq \sum_{f \in \mathcal{F}} 2 \exp \left(- \frac{t^2 n}{2(M^2 + Mt)} \right) = 2|\mathcal{F}| \exp \left(- \frac{t^2 n}{2(M^2 + Mt)} \right) \end{aligned}$$

Now, note that when $t \leq M$, then $\exp \left(- \frac{t^2 n}{2(M^2 + Mt)} \right) \leq \exp \left(- \frac{t^2 n}{M^2} \right)$. Let $\delta \in]0, 1[$, and when $n \leq \log \frac{2|\mathcal{F}|}{\delta}$, select $t = \sqrt{\frac{M^2 \log \frac{2|\mathcal{F}|}{\delta}}{n}}$. Thus, we have $t \leq M$ and the following (t, δ) -PAC bound holds:

$$\mathbb{P}_{D_n} \left(\mathcal{R}(\hat{f}_n) - \inf_{f \in \mathcal{F}} \mathcal{R}(f) > \sqrt{\frac{4M^2 \log \frac{2|\mathcal{F}|}{\delta}}{n}} \right) \leq \delta$$

Equivalently, the following holds with probability at least $1 - \delta$:

$$\mathcal{R}(\hat{f}_n) - \inf_{f \in \mathcal{F}} \mathcal{R}(f) \leq \sqrt{\frac{4M^2 \log \frac{2|\mathcal{F}|}{\delta}}{n}}$$

6.5 Bounding the Bias

Bias term depends on the specific properties of the function f^* that we try to learn, and the function space \mathcal{F} that we have chosen. Assumptions must be done on these two objects, to quantify the bias term.

Example. Let $\mathcal{X} = \mathbb{R}, \mathcal{Y} = \mathbb{R}$. We assume that the loss function ℓ satisfies the following property for some $C > 0$:

$$\forall y, y', y'', \quad |\ell(y', y) - \ell(y'', y)| \leq C|y' - y''|$$

Furthermore, we assume that the function $f^* : [-1, 1] \rightarrow \mathbb{R}$ can be decomposed as a power series:

$$f^*(x) = \sum_{k=0}^{+\infty} \beta_k x^k$$

for a sequence $(\beta_k)_{k \in \mathbb{N}}$ such that $S := \sum_{k=0}^{+\infty} \beta_k < +\infty$. Let $R > 0, p \in \mathbb{N}$, and define:

$$\mathcal{F} := \left\{ x \mapsto \sum_{k=1}^p \alpha_k x^k \mid \alpha_k \in [-R, R] \right\}$$

Denote by f_p the function $f_p : x \mapsto \sum_{k=1}^p \beta_k x^k$. When $R \geq S$, $f_p \in \mathcal{F}$, then:

$$\begin{aligned} \inf_{f \in \mathcal{F}} \mathcal{R}(f) - \mathcal{R}(f^*) &\leq \mathcal{R}(f_p) - \mathcal{R}(f^*) = \mathbb{E}[\ell(f_p(x), y) - \ell(f^*(x), y)] \\ &\leq C \cdot \mathbb{E}[|f_p(x) - f^*(x)|] \\ &\leq C \cdot \mathbb{E}\left[\sum_{k=p+1}^{+\infty} \beta_k x^k\right] \\ &\leq C \cdot \sum_{k=p+1}^{+\infty} |\beta_k| \end{aligned}$$

7 k -Nearest Neighbours

7.1 Introduction

7.1.1 Goal

We would like to classify objects, described with vectors $x \in \mathbb{R}^d$, among $L + 1$ classes $\mathcal{Y} := \{0, \dots, L\}$, automatically. To do so, we have at hand a labelled data set of n data points $(x_i, y_i) \in \mathbb{R}^d \times \mathcal{Y}$ for $i \in \llbracket 1, n \rrbracket$, which we assume to be the realizations of i.i.d. random variables (X_i, Y_i) following a distribution ν . The goal of this lesson is to build a classifier, i.e. a function

$$g : \mathbb{R}^d \longrightarrow \mathcal{Y}$$

which minimizes the probability of mistakes:

$$\mathbb{P}_{(X,Y) \sim \nu}(g(X) \neq Y) = \mathcal{R}(g) := \mathbb{E}_{(X,Y) \sim \nu}[\mathbf{1}_{g(X) \neq Y}]$$

7.1.2 Intuition of the base algorithm

The k -nearest neighbours classifier works as follows. Given a new input $x \in \mathbb{R}^d$, the classifier analyzes the k nearest points x_i in the data set $D_n = \{(x_i, y_i) \mid i \in \llbracket 1, n \rrbracket\}$ and predicts a majority vote among them.

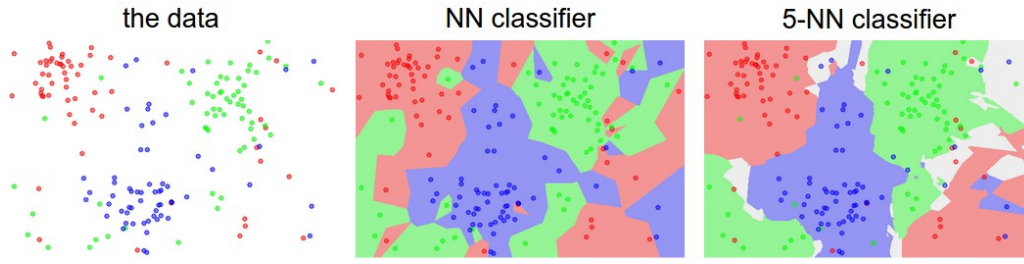


Figure 7.1: k -NN for different values of k

The k -nearest neighbor classifier is quite popular because it is simple to code and to understand; it has nice theoretical guarantess as soon as k is appropriately chosen and performs reasonably well in low dimensional spaces.

7.1.3 Assumptions and notations

In the following, unless stated otherwise, \mathbb{E} and \mathbb{P} are according to $(X, Y) \sim \nu$.

For simplicity, we will asumme in the following that we are in the binary case: $L = 1$ (two classes), $\mathcal{Y} = \{0, 1\}$. For each $l \in \mathcal{Y}$, we denote by μ_l the law of X given Y under ν and p_l the marginal distribution of Y :

$$\begin{cases} \mu_l = (X|Y = l) \\ p_l = \mathbb{P}(Y = l) \end{cases}$$

We assume that μ_l is absolutely continuous with respect to Lebesgue measure on \mathbb{R}^d . We denote by f_l its density, and for each $x \in \mathbb{R}^d$, we denote by

$$\eta(x) := \mathbb{P}(Y = 1|X = x)$$

Lemma 7.1. For any classifier $g : \mathbb{R}^d \rightarrow \mathcal{Y}$:

$$\mathcal{R}(g) = \mathbb{E}[\eta(X) \cdot \mathbb{1}_{g(X)=0} + (1 - \eta(X)) \cdot \mathbb{1}_{g(X)=1}]$$

This is a simple decomposition of the risk of the classifier between the cases when $Y = 1$ or $Y = 0$.

Proof.

$$\begin{aligned} \mathcal{R}(g) &= \mathbb{E}[\mathbb{1}_{g(X) \neq Y}] \\ &= \mathbb{E}[\mathbb{E}[\mathbb{1}_{g(X) \neq Y} | X]] \\ &= \mathbb{E}[\mathbb{E}[\mathbb{1}_{g(X) \neq 1} | X, Y = 1] \cdot \mathbb{P}(Y = 1|X) + \mathbb{E}[\mathbb{1}_{g(X) \neq 0} | X, Y = 0] \cdot \mathbb{P}(Y = 0|X)] \\ &= \mathbb{E}[\mathbb{E}[\mathbb{1}_{g(X) \neq 1} | X, Y = 1] \cdot \eta(X) + \mathbb{E}[\mathbb{1}_{g(X) \neq 0} | X, Y = 0] \cdot (1 - \eta(X))] \\ &= \mathbb{E}[\mathbb{1}_{g(X)=0} \cdot \eta(X) + \mathbb{1}_{g(X)=1} \cdot (1 - \eta(X))] \end{aligned}$$

□

7.2 Classifiers and estimators

7.2.1 The optimal Bayes classifier

Notice that a random classifier sampling $g(X) = 0$ and $g(X) = 1$ with probability $1/2$ has an expected risk $\mathcal{R}(g) = 1/2$. Our goal is therefore to build a non-trivial classifier that outperforms

this expected error. If the function η was known, one could define the Bayes classifier as follows:

$$g^*(X) = \begin{cases} 1 & \text{if } \eta(X) \geq 1/2 \\ 0 & \text{otherwise} \end{cases}$$

Definition (Consistency). We say that an estimator \hat{g}_n is *consistent* when:

$$\mathbb{E}_{(X_i, Y_i) \sim \nu} [\mathcal{R}(\hat{g}_n)] \xrightarrow{n \rightarrow +\infty} \mathcal{R}^*$$

Lemma 7.2. The risk of the Bayes classifier is

$$\mathcal{R}^* := \mathcal{R}(g^*) = \mathbb{E}[\min(\eta(X), 1 - \eta(X))]$$

Furthermore, for any classifier g , we have:

$$\mathcal{R}(g) - \mathcal{R}^* = \mathbb{E} \left[|2\eta(X) - 1| \cdot \mathbf{1}_{g(X) \neq g^*(X)} \right] \geq 0$$

This implies that the Bayes classifier is optimal, meaning that it minimizes its risk:

$$\mathcal{R}^* = \min_{g: \mathbb{R}^d \rightarrow \{0,1\}} \mathcal{R}(g)$$

Proof. □

Therefore, if η was known, one could compute the optimal classifier g^* . In practice, η is unknown, and one should therefore estimate it.

7.2.2 Plug-in estimator

Definition (Estimator). An estimator $\hat{\eta}_n$ of η is a function of the observation $D_n = (X_i, Y_i)_{i \in \llbracket 1, n \rrbracket}$ of the form:

$$\hat{\eta}_n : \mathbb{R}^d \longrightarrow [0, 1]$$

where $\hat{\eta}_n$ does depend on the data set D_n .

Definition (Plug-in estimator). Given an estimator $\hat{\eta}_n$ of η , we can build the *plug-in estimator* as follows:

$$\hat{g}_n(x) := \begin{cases} 1 & \text{if } \hat{\eta}_n(x) \geq 1/2 \\ 0 & \text{otherwise} \end{cases} \quad (7.2.1)$$

The intuition is that if $\hat{\eta}_n$ is close enough to η , then \hat{g}_n is also close to g^* and $\mathcal{R}\hat{g}_n$ will be close to \mathcal{R}^* . This is formalized by the following Lemma:

Lemma 7.3. Given a plug-in estimator \hat{g}_n , we have:

$$0 \leq \mathcal{R}(\hat{g}_n) - \mathcal{R}^* \leq 2 \cdot \mathbb{E} \left[|\eta(X) - \hat{\eta}_n(X)| \middle| D_n \right]$$

Proof. □

This shows that for $\hat{\eta}_n = \eta$, $\hat{g}_n = g^*$, the optimal Bayes classifier. Furthermore, if $\hat{\eta}_n \simeq \eta$, then $\hat{g}_n \simeq g^*$. Therefore, if we could build from the data an estimator $\hat{\eta}_n$ of η such that

$$\forall x \in \mathbb{R}^d, \quad \mathbb{E} \left[|\eta(X) - \hat{\eta}_n(X)| \middle| D_n \right] \xrightarrow[n \rightarrow +\infty]{0}$$

then the associated plug-in classifier would be consistent. The reverse is not true: estimating η is harder than estimating g^* , since η is real-valued while g^* can only take two values. We will show that the k -nearest neighbors is consistent if the number of neighbors grows appropriately (k is a function of n). This is not the case when the number of neighbors is fixed (k independent of n).

7.3 The k -nearest neighbors classifier (k -NN)

The k -NN classifier classifies a new input x with the majority class among its k -nearest neighbors, as can be seen in Figure 7.1. Formally, we denote by $\bar{X}_i(x)$ the i -th nearest neighbor of $x \in \mathbb{R}^d$ among the inputs X_i (using the Euclidean distance – that is ℓ_2 or $\|\cdot\|_2$). The $\bar{X}_i(x)$ are such that:

$$\|x - \bar{X}_1(x)\|_2 \leq \|x - \bar{X}_2(x)\|_2 \leq \dots \leq \|x - \bar{X}_n(x)\|_2$$

We denote by $\bar{Y}_i(x)$ the class associated with $\bar{X}_i(x)$, which is known – we are solving a supervised learning problem.

Definition (k -NN estimator). The estimator associated to the k -NN algorithm is the average of the classes among the k -nearest neighbors:

$$\hat{\eta}_n^k(x) := \frac{1}{k} \sum_{i=1}^k \bar{Y}_i(x) \quad (7.3.1)$$

Definition (k -NN classifier). The k -NN classifier is the plug-in estimator associated to the k -NN estimator $\hat{\eta}_n^k(x)$:

$$\hat{g}_n^k(x) := \begin{cases} 1 & \text{if } \frac{1}{k} \sum_{i=1}^k \bar{Y}_i(x) \geq 1/2 \\ 0 & \text{otherwise} \end{cases} \quad (7.3.2)$$

Definition (Asymptotic risk of the k -NN classifier). The *asymptotic risk* of the k -NN classifier is the limit risk of $\hat{g}_n^k(x)$:

$$\mathcal{R}_{k\text{-NN}} := \lim_{n \rightarrow +\infty} \mathbb{E}_{(X_i, Y_i) \sim \nu} [\mathcal{R}(\hat{g}_n^k)]$$

7.3.1 The nearest neighbor classifier (1-NN)

7.3.2 Inconsistency of the k -NN classifier for fixed k

7.3.3 Consistency of nearest neighbors making $k \rightarrow +\infty$

8 Unsupervised Learning

As described in the introduction of this course, Machine Learning is divided into three main categories: Supervised Learning, Unsupervised Learning, and Reinforcement Learning. In this chapter, we will analyze *Unsupervised Learning*: while Supervised Learning provided a dataset containing both features (x_i) and labels (y_i); in Unsupervised Learning, the dataset contains only features x_i .

8.1 K -means

K -means clustering is a method of vector quantization. K -means clustering is an algorithm of alternate minimization that aims at partitioning n observations into K clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype to the cluster.

8.1.1 Distortion

Definition (Observations). In the clustering problem, we are given $x_i \in \mathbb{R}^p$ for $i \in \llbracket 1, n \rrbracket$, the *observations* we want to partition.

Definition (Means). For some given K , we denote $\mu_k \in \mathbb{R}^p$ for $k \in \llbracket 1, K \rrbracket$ the means. μ_k is the center of the cluster k . We will denote μ the associated matrix.

Definition (Hard membership variables). $z_i^k \in \{0, 1\}$ are indicator variables associated to x_i , such that:

$$z_i^k := \begin{cases} 1 & \text{if } x_i \text{ belongs to the cluster } k \\ 0 & \text{otherwise} \end{cases}$$

We denote by z the matrix which components are equal to z_i^k .

Definition (Distortion). We define the distortion $J(\mu, z)$ by:

$$J(\mu, z) := \sum_{i=1}^n \sum_{k=1}^K z_i^k \|x_i - \mu_k\|^2 \quad (8.1.1)$$

Distortion measures the overall distance of the centers μ_k to the points of the cluster.

Definition (K -means optimization problem). The K -means optimization problem consists in finding matrices μ and z which minimize the distortion:

$$\min_{\substack{\mu \in \mathcal{M}_{n,k}(\mathbb{R}) \\ z \in \mathcal{M}_{n,k}(\mathbb{R})}} J(\mu, z) = \min_{\substack{\mu \in \mathcal{M}_{n,k}(\mathbb{R}) \\ z \in \mathcal{M}_{n,k}(\mathbb{R})}} \sum_{i=1}^n \sum_{k=1}^K z_i^k \|x_i - \mu_k\|^2 \quad (8.1.2)$$

8.1.2 Algorithm

To minimize $J(\mu, z)$, we proceed using an alternating minimization: we will minimize J alternatively with respect to z and μ .

1. Randomly choose a vector μ .
2. Minimize J with respect to z :

$$z_i^k = \begin{cases} 1 & \text{if } \|x_i - \mu_k\|^2 = \min_s \|x_i - \mu_s\|^2 \\ 0 & \text{otherwise} \end{cases} \quad (8.1.3)$$

This is equivalent to associating to x_i the nearest center μ_k .

3. Minimize J with respect to μ :

$$\mu_k = \frac{\sum_{i=1}^n z_i^k x_i}{\sum_{i=1}^n z_i^k} \quad (8.1.4)$$

4. Come back to step 2 until convergence.

Proof. The formula (8.1.4) can be obtained by minimizing the gradient of J with respect to μ_k :

$$\nabla_{\mu_k} J = -2 \sum_{i=1}^n z_i^k (x_i - \mu_k)$$

Therefore:

$$\nabla_{\mu_k} J = 0 \iff \mu_k = \frac{\sum_{i=1}^n z_i^k x_i}{\sum_{i=1}^n z_i^k}$$

□

8.1.3 Convergence and initialization: K -means++

We can show that this algorithm converges in a finite number of iterations. Nevertheless, the convergence could be towards to local minimum, introducing the problem of initialization.

A classic method is to use random restarts. It consists in choosing several random vectors μ , computing the algorithm for each case and finally keeping the partition which minimizes the

distortion. Thus, we hope that at least one of the local minima is close enough to a global minimum.

Another well known method is the K -means++ algorithm, which aims at correcting a major theoretical shortcoming of the K -means algorithm: the approximation found can be arbitrarily bad with respect to the objective function compared to the optimal clustering.

The K -means++ algorithm addresses this obstacles by specifying a procedure to initialize the cluster centers before proceeding with the standard K -means optimization iterations. With the K -means++ initialization, the algorithm is guaranteed to find a solution that is $O(\log K)$ competitive to the optimal K -means solution.

The intuition behind this approach is that spreading the K initial cluster centers will improve the quality of the clusters found by the algorithm. At each iteration of the algorithm we will build a new center, and repeat this step until we have K centers. Here are the steps of the algorithm:

1. Choose the first center uniformly among the data points.
2. For each data point x_i of the data set, compute the distance between x_i and the nearest center that has already been chosen. We denote this distance $D_{\mu_t}(x_i)$ where μ_t is specified to recall that we are minimizing over the current chosen centers.
3. Choose a new data point at random as a new center, but now using a weighted probability distribution where a point x_i is chosen with probability proportional to $D_{\mu_t}(x_i)^2$.
4. Come back to step 2 until K centers have been chosen.

We see that we have now built K vectors with respect to our first intuition which was to well spread the centers – because we used a well-chosen weighted probability. We can now use those vectors as the initialization of our standard K -means algorithm.

8.1.4 Choice of K

It is important to point out that the choice of K is not universal. Indeed, we see that if we increase K , the distortion J decreases, until reaches 0 when $K = n$ – when each data point is the center of its own cluster. To address this issue, one solution could be to add to J a penalty over K . Usually, it takes the following form:

$$J(\mu, z, K) := \sum_{i=1}^n \sum_{k=1}^K z_i^k \|x_i - \mu_k\|^2 + \lambda K$$

Once again, the choice of the penalty coefficient λ is arbitrary. There is no general way to choose K .

8.1.5 Other problems

We can also point out that K -means will work properly when the width of the different clusters are similar – for example when dealing with spheres. But clustering by K -means could also be disappointing in some cases such as the following example:

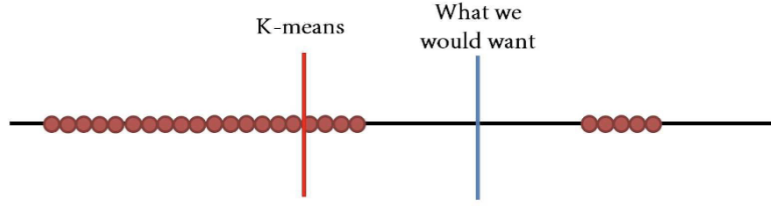


Figure 8.1: Example where K -means does not provide a satisfactory clustering result

For instance, the use of Gaussian mixtures provides a way to avoid this problem.

8.2 Dimensionality reduction: principal component analysis

Imagine that we want to process some video data $(x_i)_{i \in \llbracket 1, n \rrbracket}$ where $x_i \in \mathbb{R}^p$. In the case of video data, p can be prohibitively huge to manipulate, e.g. $p \simeq 10^{11}$. We would like to reduce the dimensionality of the origin space, i.e. finding a new representation for each of the x_i :

$$x_i \in \mathbb{R}^p \mapsto \hat{x}_i \in \mathbb{R}^{p'} \quad \text{where} \quad p' \ll p$$

without losing too much information.

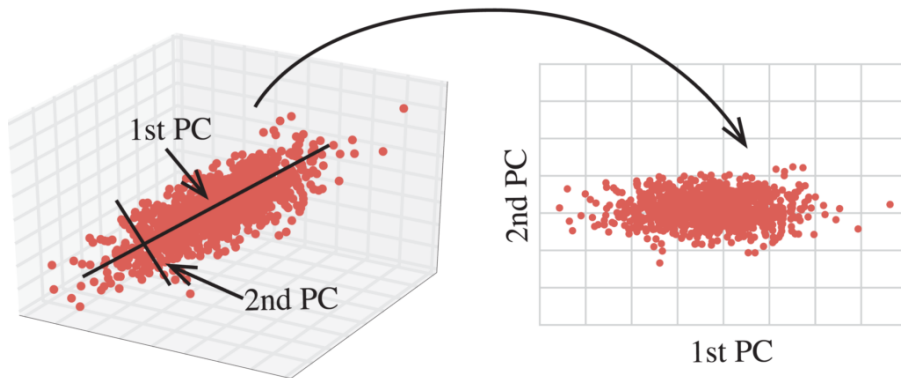


Figure 8.2: Dimensionality reduction from \mathbb{R}^3 to \mathbb{R}^2 , i.e. using two principal components

8.2.1 Analysis view

We will start by introducing formally the problem. Assume that there is some data distribution ν over \mathbb{R}^p . A random variable $X \in \mathbb{R}^p \sim \nu$ samples this data distribution over this high-dimensional space. Assume that $\mathbb{E}[X] = 0$, and let

$$\Sigma := \text{Cov}(X) = \mathbb{E}[(X - \mathbb{E}[X])(X - \mathbb{E}[X])^\top] = \mathbb{E}[XX^\top]$$

Σ is the covariance matrix of X ; note that it is definite non-negative.

The intuition is that we want to preserve as much “diversity” in the output after the dimension reduction. This leads to the following definition of the problem in the case where $p' = 1$.

Definition (PCA problem). Find $w \in \mathbb{R}^p$ such that:

$$w \in \arg \max_{\|w\|=1} \mathbb{V}(w^\top X) \tag{8.2.1}$$

Theorem. w verifies (8.2.1) exactly when w is the eigenvector of Σ corresponding to the largest eigenvalue.

Proof. Note that:

$$\begin{aligned}\mathbb{V}(\omega^\top X) &= \mathbb{E}[\omega^\top X]^2 - (\mathbb{E}[\omega^\top X])^2 \\ &= \mathbb{E}[(\omega^\top X)(\omega^\top X)] \\ &= \mathbb{E}[\omega^\top X X^\top \omega] \\ &= \omega^\top \mathbb{E}[X X^\top] \omega \\ &= \omega^\top \Sigma \omega\end{aligned}$$

Therefore,

$$\sup_{\|\omega\|=1} \mathbb{V}(\omega^\top X) = \sup_{\|\omega\|=1} \omega^\top \Sigma \omega$$

Thus, by property, ω is exactly the eigenvector of Σ corresponding to the largest eigenvalue. We can write $\Sigma = U D U^\top$ with $U \in \mathbb{R}^{p \times p}$, and $U U^\top = I_p$, with $D = \text{diag}(\lambda_1, \dots, \lambda_p)$. If we assume $\lambda_1 \geq \dots \geq \lambda_p$, we have that ω is the first column of U . \square

Remark. To generalize this procedure to p' dimensions, we take the p' eigenvectors associated to the p' largest eigenvalues. Another approach would be to use deflation, defined by the following algorithm:

1. Find w using the largest eigenvalue.
2. Project the data onto $\text{Span}(w)$.
3. Start at Step 1.

8.2.2 In practice

In practice, we do not know Σ – since it depends on the unknown distribution ν – but we are given a dataset to work with. Given $(x_i)_{i \in \llbracket 1, n \rrbracket}$, we want to find a vector $w \in \mathbb{R}^p$ – or a matrix $W \in \mathcal{M}_{p, p'}(\mathbb{R})$ – such that the associated projection maximizes the variance of the projected dataset. In the theoretical analysis, we assumed that $\mathbb{E}[X] = 0$; to verify the assumptions, we need the dataset to be centered, or *center* it if necessary. Thus, we introduce the mean of the dataset (a vector) and subtract it to the initial dataset:

$$x_i \leftarrow x_i - \mu \quad \text{where} \quad \mu = \frac{1}{n} \sum_{i=1}^n x_i$$

This being done, we can define the *empirical covariance matrix*, which is the approximation of the covariance matrix X with respect to the dataset available:

$$\hat{\Sigma} := \frac{1}{n} \sum_{i=1}^n x_i x_i^\top$$

Similarly to the theoretical setting, we can use the normlized eigenvectors associated to the largest eigenvalues to construct a matrix maximizing the variance.

9 Probabilistic models and Maximum Likelihood

9.1 Introduction

In probabilistic modeling, we are given a set of observations $D_n = (y_1, \dots, y_n)$ in \mathcal{Y} that we assume to be generated from some unknown i.i.d. distribution. As always, the objective is to find

a probabilistic model that explains well the data, by estimating the density of the underlying distribution. If possible, we would like the model to predict well new data and to be able to incorporate prior knowledge and assumptions.

Let μ denote some reference measure on the output set \mathcal{Y} . Typically, μ is the counting measure if $\mathcal{Y} \subseteq \mathbb{N}$ or the Lebesgue measure if $\mathcal{Y} \subseteq \mathbb{R}^d$.

Definition (Parametric model). Let $d \geq 1$ and $\Theta \subseteq \mathbb{R}^d$ by a set of parameters. A parametric model \mathcal{P} is a set of probability distributions taking value in \mathcal{Y} with a density with respect to μ and indexed by Θ :

$$\mathcal{P} = \{p_\theta d\mu \mid \theta \in \Theta\} \quad (9.1.1)$$

Here are a few examples of statistical parametric models based on well known family distributions.

Example (Binomial model). For $\mathcal{Y} = \mathbb{N}$ and $\Theta = [0, 1]$:

$$p_\theta(k) = \binom{n}{k} \theta^k (1 - \theta)^{n-k}$$

Example (Gaussian model). For $\mathcal{Y} = \mathbb{R}$ and $\Theta = \mathbb{R} \times \mathbb{R}_+$, where we traditionnaly denote elements of Θ as (μ, σ) , the parameters of the normal distribution:

$$p_{(\mu, \sigma)}(x) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right)$$

Example (Multidimensional⁹ Gaussian model). For $\mathcal{Y} = \mathbb{R}$ and $\Theta = \mathbb{R}^d \times S_d^+(\mathbb{R})^{10}$, where we traditionnaly denote elements of Θ as (μ, Σ) , the parameters of the normal distribution:

$$p_{(\mu, \Sigma)}(x) = \frac{1}{(2\pi)^{d/2} |\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu)^\top \Sigma^{-1}(x - \mu)\right)$$

Other parametric models include the exponential model on $\mathcal{Y} = \mathbb{R}_+$, the Bernoulli model on $\mathcal{Y} = \{0, 1\}$, ...

In the following, we assume that we are given some model \mathcal{P} indexed by $\theta \in \Theta$, and we assume that the data D_n is independently generated from $p_{\theta_*} \in \mathcal{P}$ for some unknown parameter θ_* . We would like to recover the best parameter θ_* from the data.

Remark. *In practice, the data might come from a distribution which is not in \mathcal{P} : this is called misspecification, but we will not enter into further details in this chapter.*

9.2 Maximum likelihood estimation

9.2.1 Definition and example

The idea behind maximum likelihood estimation is to choose the most probable parameter $\theta \in \Theta$ for the observed data. Assume that \mathcal{Y} is discrete and that $Y \sim p_{\theta_*} d\mu$ for some $\theta_* \in \Theta$. Then, for any observation y_i , the probability that $Y = y_i$ equals $p_{\theta_*}(y_i)$:

$$\mathbb{P}(Y = y_i) = p_{\theta_*}(y_i)$$

⁹Also known as *multivariate*.

¹⁰ $S_d^+(\mathbb{R})$ denotes the set of semi-definite positive matrices. In particular, such matrices M are invertible, and the associated bilinear form is positive: $\forall x \in \mathbb{R}^d, x^\top M x$.

Similarly, the probability of observing $(y_1, \dots, y_n) \in \mathcal{Y}^n$ if all the samples were independently sampled from p_θ is $\prod_{i=1}^n p_\theta(y_i)$. Hence, the high level idea of maximum likelihood estimation will be to maximize this probability over $\theta \in \Theta$. This is formalized by the definition of the likelihood, which also holds for non-discret set \mathcal{Y} .

Definition (Likelihood). Let $\mathcal{P} = \{p_\theta d\mu \mid \theta \in \Theta\}$ be a parametric model and $y \in \mathcal{Y}$. Given the outcome $y \in \mathcal{Y}$, the *likelihood* is the function:

$$\theta \longmapsto p_\theta(y) \quad (9.2.1)$$

The likelihood $L(\cdot|D_n)$ of a dataset $D_n = (y_1, \dots, y_n)$ is the function:

$$L(\cdot|D_n) : \theta \longmapsto \prod_{i=1}^n p_\theta(y_i) \quad (9.2.2)$$

Definition (Maximum likelihood estimator). The *maximum likelihood estimator (MLE)* is the parameter which maximizes the likelihood, i.e.:

$$\hat{\theta}_n \in \arg \max_{\theta \in \Theta} \prod_{i=1}^n p_\theta(y_i) \quad (9.2.3)$$

In practice, maximizing a sum is easier than to maximize a product. Since log is an increasing function, the maximum likelihood estimator also maximizes the *log-likelihood*:

$$\hat{\theta}_n \in \arg \max_{\theta \in \Theta} \sum_{i=1}^n \log p_\theta(y_i) \quad (9.2.4)$$

Example (Bernoulli model). For $\mathcal{Y} = \{0, 1\}$ and $\Theta = [0, 1]$, we define the Bernoulli model as:

$$p_\theta(y) = \theta^y (1 - \theta)^{(1-y)}$$

Assuming that D_n was generated from a Bernoulli distribution of parameter θ_* ; then, the maximum likelihood estimator is:

$$\hat{\theta}_n \in \arg \min_{0 \leq \theta \leq 1} \frac{1}{n} \sum_{i=1}^n [y_i \log \theta + (1 - y_i) \log(1 - \theta)]$$

Denoting $\bar{y}_n := \sum_{i=1}^n y_i$ the empirical average, we solve:

$$\left. \frac{d \log L(\theta|D_n)}{d\theta} \right|_{\theta=\hat{\theta}_n} = 0 \implies \frac{\bar{y}_n}{\hat{\theta}_n} - \frac{1 - \bar{y}_n}{1 - \hat{\theta}_n} \implies \hat{\theta}_n = \bar{y}_n$$

Therefore, the maximum likelihood estimator is in this case the empirical mean.

9.2.2 Link with empirical risk minimization

In this chapter, we studied density estimation, in which the goal is to find the density of the distribution which generated the data. This can be connected to empirical risk minimization: assuming that the density belongs to the model \mathcal{P} , the possible densities are p_θ , for some $\theta \in \Theta$. We can define a loss function for this setting to be the negative log-likelihood:

$$\begin{aligned} \ell : \Theta \times \mathcal{Y} &\longrightarrow \mathbb{R}^+ \\ (\theta, y) &\longmapsto -\log p_\theta(y) \end{aligned} \quad (9.2.5)$$

The risk – or generalization error – is then:

$$\mathcal{R}(\theta) = -\mathbb{E}_Y [\log p_\theta(Y)] \quad (9.2.6)$$

In particular, if $Y \sim p_{\theta_*} d\mu$, for some $\theta_* \in \Theta$, θ_* minimizes the risk and the objective is to recover θ_* . The empirical risk is then, by definition:

$$\hat{\mathcal{R}}_n(\theta) = -\frac{1}{n} \sum_{i=1}^n \log p_\theta(y_i) \quad (9.2.7)$$

Therefore, the empirical risk minimizer matches the maximum likelihood estimator.

9.2.3 Link with Kullback-Leibler divergence

The Kullback-Leibler divergence is a measure of dissimilarity between two probability distributions.

Definition. Let $p d\mu$ and $q d\mu$ be two probability distributions with $p \ll q$ ¹¹. The *Kullback-Leibler divergence from q to p* is defined as:

$$\text{KL}(p||q) := \mathbb{E}_{Y \sim p d\mu} \left[\log \frac{p(Y)}{q(Y)} \right] = \int_{\mathcal{Y}} p(y) \log \frac{p(y)}{q(y)} d\mu(y) \quad (9.2.8)$$

The Kullback-Leibler divergence has two interpretations. In information theory, it can be seen as the difference of symbols needed to encode D_n under a code optimized for $p d\mu$ compared to a code optimized for $q d\mu$.

$\text{KL}(p_\theta||p_{\theta_*})$ can also be interpreted as the excess risk of the measure $p_\theta d\mu$ when the data follows distribution $p_{\theta_*} d\mu$ for the negative log-likelihood loss function. Assume that the data D_n was generated from p_θ . Then, the excess risk can be written:

$$\begin{aligned} \mathcal{R}(\theta) - \mathcal{R}(\theta_*) &= -\mathbb{E}_{Y \sim p_{\theta_*} d\mu} [\log p_\theta(Y)] + \mathbb{E}_{Y \sim p_{\theta_*} d\mu} [\log p_{\theta_*}(Y)] \\ &= \mathbb{E}_{Y \sim p_{\theta_*} d\mu} \left[\log \frac{p_{\theta_*}(Y)}{p_\theta(Y)} \right] \\ &=: \text{KL}(p_{\theta_*}||p_\theta) \end{aligned}$$

Property 9.1 (Positivity). By Jensen's inequality:

$$\text{KL}(p||q) \geq 0 \quad (9.2.9)$$

Property 9.2 (Separation).

$$\text{KL}(p||p) = 0 \quad (9.2.10)$$

Therefore, p_{θ_*} minimizes the risk and maximizes the likelihood.

Property 9.3 (Discrete case). If \mathcal{Y} is discrete and μ is the counting measure, we have:

$$\text{KL}(p||q) = \sum_{y \in \mathcal{Y}} p(y) \log \frac{p(y)}{q(y)} \quad (9.2.11)$$

Note that the KL divergence is often seen as a distance, but it does not fill the requirements, since it is not symmetric and does not satisfy the triangular inequality.

¹¹We say that q is *absolutely continuous with respect to p* , meaning that $q(A) = 0 \implies p(A) = 0$

9.2.4 Conditional modeling

Until now, we considered the problem of density estimation when the data set has only outputs $y_i \in \mathcal{Y}$. However, the principle of maximum likelihood can be extended to couples of inputs $D_n = \{(x_1, y_1), \dots, (x_n, y_n)\} \in \mathcal{X} \times \mathcal{Y}$. Then, we can distinguish two different modeling: generative and conditional modeling.

Generative modeling We aim at estimating the density of couples of input and outputs (X, Y) among a family of densities on $\mathcal{X} \times \mathcal{Y}$:

$$(x, y) \longmapsto p_\theta(x, y)$$

In this case, the risk and the empirical risk are:

$$\begin{cases} \mathcal{R}(\theta) = -\mathbb{E}[\log p_\theta(X, Y)] \\ \hat{\mathcal{R}}_n(\theta) = -\frac{1}{n} \sum_{i=1}^n \log p_\theta(x_i, y_i) \end{cases}$$

Estimating the density can be useful to generate some new samples.

Conditional modeling We aim at estimating the density of an output Y *given* X . The densities now are on \mathcal{Y} only:

$$y \longmapsto p_\theta(\cdot | x)$$

but do depend on the inputs. The risk and the empirical risk with negative log-likelihood are:

$$\begin{cases} \mathcal{R}(\theta) = -\mathbb{E}[\log p_\theta(Y|X)] \\ \hat{\mathcal{R}}_n(\theta) = -\frac{1}{n} \sum_{i=1}^n \log p_\theta(y_i | x_i) \end{cases}$$

This is useful if one wants to predict the distribution or the value of a new output Y given an input X .

9.2.5 Linear regression as conditional modeling

Unlike Hinge loss, logistic loss has a probabilist interpretation. Consider $(X, Y) \sim \mathcal{P}$ a pair of input-output random variables in $\mathbb{R}^d \times \{0, 1\}$. In binary classification, the objective is to predict the probability that $Y = 1$ given the input X . In other words, we want to estimate $\mathbb{P}(Y = 1|X)$, that is:

$$x \longmapsto \mathbb{P}(Y = 1|X = x)$$

from observations $D_n = \{(x_i, y_i) \mid i \in \llbracket 1, n \rrbracket\} \subseteq \mathbb{R}^d \times \{0, 1\}$, that were independently generated from \mathcal{P} . The issue is that linear predictions of the form $\theta^\top x_i$ belong to \mathbb{R} , while our model needs to output probabilities with values in the range $[0, 1]$. Therefore, the *sigmoid* function is often used, which maps \mathbb{R} to $[0, 1]$.

Definition (Sigmoid function). The sigmoid function σ is defined by:

$$\begin{aligned} \sigma : \mathbb{R} &\longrightarrow [0, 1] \\ z &\longmapsto \frac{1}{1 + e^{-z}} \end{aligned} \tag{9.2.12}$$

Property 9.4. The following identity holds: $\sigma(-z) = 1 - \sigma(z)$

Property 9.5 (Derivative of sigmoid). The derivative of sigmoid is: $\sigma'(z) = \sigma(z)\sigma(-z)$

Logistic regression assumes the underlying probabilistic model:

$$\mathbb{P}(Y = 1|X) = \sigma(\theta^\top X)$$

It is worth pointing out that this probabilistic model is satisfied for many natural models: for instance, if $X|Y = 1$ and $X|Y = 0$ follow independent Gaussian distributions, this model is satisfied.

We can define the family of possible densities (with respect to the countable measure) on $\{0, 1\}$ considered here:

$$p_\theta(y|x) = \sigma(\theta^\top x)^y (1 - \sigma(\theta^\top x))^{1-y} = \begin{cases} 1 - \sigma(\theta^\top x) & \text{if } y = 0 \\ \sigma(\theta^\top x) & \text{otherwise} \end{cases} \quad (9.2.13)$$

indexed by $\theta \in \mathbb{R}^d$. We recall that similarly to linear regression, to ease the notation, the intercept (to make affine predictions) may be included into the input X by adding the constant 1 as the first coefficient.

Property 9.6. Assume that the (x_i, y_i) are realizations of i.i.d. random variables. Then, the logistic regression estimator

$$\hat{\theta} \in \arg \min_{\theta \in \mathbb{R}^d} \sum_{i=1}^n \ell(\theta^\top x_i, y_i) \quad \text{where} \quad \ell(x, y) := y \log \sigma(x) + (1 - y) \log \sigma(-x)$$

matches the definition of the maximum likelihood estimator of $\theta \in \mathbb{R}^d$ with parametric model (9.2.13).

Proof. The conditional log-likelihood can be written:

$$\begin{aligned} \log L(\cdot|D_n) &= \sum_{i=1}^n \log p_\theta(y_i|x_i) \\ &= \sum_{i=1}^n \log \left(\sigma(\theta^\top x_i)^{y_i} (1 - \sigma(\theta^\top x_i))^{1-y_i} \right) \\ &= \sum_{i=1}^n y_i \log \left(\sigma(\theta^\top x_i) \right) + (1 - y_i) \log \left(1 - \sigma(\theta^\top x_i) \right) \\ &= \sum_{i=1}^n \ell(\theta^\top x_i, y_i) \end{aligned}$$

where ℓ is the logistic loss defined above. □

9.3 Maximum a posteriori

If the dimension d of the parameter space Θ is too large compared to the number of samples n , the maximum likelihood estimator (MLE) may lead to poor performance. Similarly to least square linear regression without regularization, MLE overfits when $d > n$. A second limitation is that no prior knowledge on the parameters θ is included. Let us detail this using an example.

Example. We consider the multinomial model where each observation is a discrete observation in k classes $\{1, \dots, k\}$. Each class $j \in \{1, \dots, k\}$ is sampled with a probability θ_j^* , and we aim at retrieving these probabilities. We define for convenience the output set:

$$\mathcal{Y} = \left\{ y \in \{0, 1\}^k \mid \sum_{i=1}^k y_i = 1 \right\}$$

An observation y is such that $y(j) = 1$ if it is in class $j \in \{1, \dots, k\}$, and 0 otherwise. The multinomial model consists of densities of the form:

$$p_\theta : y \in \mathcal{Y} \mapsto \prod_{j=1}^k \theta_j^{y(j)} \quad \text{where} \quad \theta \in [0, 1]^k, \quad \sum_{j=1}^k \theta_j = 1$$

In other words, the probability of an observation to be in class j equals θ_j , since only one term of the product is different from 1. The dimension of the parameter space is $d = k - 1$ – we choose k coefficients for θ , with 1 constraint. The MLE is:

$$\hat{\theta}_j = \arg \max_{\theta_j} \frac{1}{n} \sum_{i=1}^n y_i(j) \log \theta_j = \frac{n_j}{n} \quad \text{where} \quad n_j = \sum_{i=1}^n y_i(j)$$

n_j is the number of occurrence of class j in the data set. If $k > n$, many classes j are never observed and estimated with 0. This can happen, for example, when we are trying to associate probabilities to words, each word being its own class, but the number of possible words k is much larger than the number of words in the dataset text. The log-loss of these options is infinite, and so is the risk $\mathcal{R}(\theta)$. We say that the model is overestimating.

This problem can be solved by adding a regularization, which can also be seen from a Bayesian point of view as a prior distribution over the possible distributions θ . This is what Maximum a Posteriori (MAP) does. The idea behind MAP is to see the parameter θ as a random variable taking values in Θ , and to choose the most probable value $\hat{\theta}^{\text{MAP}}$ for the observed data. Given the data set D_n , the MAP can be formalized as the solution of:

$$\hat{\theta}_n^{\text{MAP}} \in \arg \max_{\theta \in \Theta} p(\theta | D_n)$$

where $p(\theta | D_n)$ is the density of the posterior distribution of the model given the data. In discrete model space Θ , the MAP is exactly the most probable model. To calculate the posterior distribution we use the Bayes rule:

$$p(\theta | D_n) = \frac{p(D_n | \theta) \cdot p(\theta)}{p(D_n)}$$

where:

- D_n is a random data set
- $p(D_n | \theta)$ is the probability density of observing D_n if the distribution follows $p_\theta d\mu$. This is exactly the likelihood $L(\theta | D_n)$
- $p(\theta)$ is the prior distribution of the model, that is how likely we think it is before seeing the data
- $p(D_n)$ is the marginal distribution of the data

Hence, the MAP is the solution of:

$$\hat{\theta}_n^{\text{MAP}} \in \arg \max_{\theta \in \Theta} L(\theta | D_n) \cdot p(\theta) = \arg \min_{\theta \in \Theta} -\frac{1}{n} \sum_{i=1}^n \log p_\theta(y_i) + \log \frac{1}{p(\theta)} \quad (9.3.1)$$

In some situation, we may not have to prefer one model over another, and one can think of $p(\theta)$ as a constant of the parameter space Θ . Then, the MAP with uniform prior reduces to the MLE. However, this assumption that $p(\theta)$ is constant is problematic, because uniform distribution cannot always be defined if Θ is not compact. Therefore, it may be better to see MAP as a regularized version of MLE, with a regularization of the form $-\log p(\theta)$, rather than MLE as a particular case of MAP with uniform prior.

10 Posterior sampling using MCMC and SGLD

So far, we have discussed several tasks that arise in statistical learning, and we expressed most of these tasks by certain optimization problems, such as the empirical risk minimization problem. The Maximum A Posteriori (MAP) estimation problem, introduced in the previous chapter, being one of the examples of such optimization problems, arose naturally when we limited ourselves to the framework of probabilistic modeling. In this chapter, we will again consider the probabilistic modeling framework, but this time with a different question than the MAP estimation problem.

10.1 Introduction

10.1.1 Bayesian probabilistic model

Let's start by remembering what a Bayesian probabilistic model is. The main idea in Bayesian modeling is to treat the parameter of interest $\theta \in \mathbb{R}^d$ as a random vector that follows a *prior distribution* $p(\theta)$.

A single observation y is assumed to be a random variable drawn from the *conditional distribution* $p(y|\theta)$. We typically consider n observations $D_n = \{y_1, \dots, y_n\}$, which are i.i.d. distributed when conditioned on θ .

We often denote a Bayesian probabilistic model as follows:

$$\begin{aligned} \theta &\sim p(\theta) \\ \forall i \in \llbracket 1, n \rrbracket, \quad y_i | \theta &\sim p(y_i | \theta) \end{aligned} \tag{10.1.1}$$

Under this model, the goal in MAP estimation is to find the value $\hat{\theta}_n$ that maximizes the *posterior distribution*:

$$p(\theta | D_n) = \frac{p(D_n | \theta) p(\theta)}{p(D_n)} \tag{10.1.2}$$

Since the logarithm function is monotonic and $p(D_n)$ does not depend on θ , the MAP estimation problem can be written in the following form:

$$\hat{\theta}_n \in \arg \max_{\theta} \sum_{i=1}^n \log p(y_i | \theta) + \log p(\theta) \tag{10.1.3}$$

10.1.2 Level of uncertainty

The MAP estimator can be very useful in practical applications and there is a broad literature for solving this optimization problem for a variety of choices of prior distributions, with various theoretical guarantees. However, in a broad range of applications, a single point estimate can be insufficient, especially for risk-intolerant applications (such as autonomous driving). Hence, it would be of interest to extract more information about our estimate to have an understanding of *the level of uncertainty* in our estimation.

While there are many ways of doing this, here, we will consider the Bayesian approach and try to infer the uncertainty information by making a better use of the posterior distribution. For instance, as a measure of uncertainty, we can consider the posterior variance.

Definition (Posterior variance). The *posterior variance*, denoted σ_{post}^2 , is defined by:

$$\sigma_{\text{post}}^2 := \int \|\theta - \mu_{\text{post}}\|^2 \cdot p(\theta | D_n) \, d\theta \quad \text{where} \quad \mu_{\text{post}} := \int \theta \cdot p(\theta | D_n) \, d\theta \tag{10.1.4}$$

Note that μ_{post} is the posterior expectation. Intuitively, if the value of σ_{post}^2 is high, we might conclude that there is a high uncertainty in our estimation: the posterior distribution is too widespread. If σ_{post}^2 is low, we might be confident in our estimation.

In general, the goal in Bayesian statistics is to compute the expectation of a *test function* $f : \mathbb{R}^d \rightarrow \mathbb{R}$ under the posterior distribution:

$$\bar{f} := \mathbb{E}_{\theta \sim p(\theta|D_n)}[f(\theta)] = \int f(\theta) \cdot p(\theta|D_n) d\theta \quad (10.1.5)$$

As opposed to MAP estimation, where the marginal likelihood $p(D_n)$ is irrelevant and can be safely discarded, this task unfortunately needs all the information about the posterior distribution, including $p(D_n) = \int p(D_n|\theta) \cdot p(\theta) d\theta$. Therefore, computing these integrals can become very difficult very quickly, especially when d is large.

10.2 Markov Chain Monte-Carlo (MCMC)

There are many ways to approximately compute integrals. In this chapter, we will consider an instance of a powerful family of tools, that is called Markov Chain Monte-Carlo (MCMC) methods. In this section, we will cover the main idea in MCMC and then in the next section, we will introduce the algorithm that is the main topic of this chapter (SGLD).

10.2.1 Monte-Carlo estimator

Let us first consider the following “ideal” scenario and assume that we have access to an algorithm that can give us independent random samples drawn from the posterior distribution $p(\theta|D_n)$:

$$\theta_k \sim p(\theta|D_n), \quad \text{for } k \in \llbracket 1, K \rrbracket$$

If this was the case, we could introduce an estimator to (10.1.5) easily:

$$\hat{f}_K := \frac{1}{K} \sum_{k=1}^K f(\theta_k) \quad (10.2.1)$$

This estimator is called the Monte-Carlo estimator, and under mild integrability assumptions, we would know that:

$$\hat{f}_K \xrightarrow{K \rightarrow +\infty} \bar{f}$$

However, this approach introduces a new question: “*How can we draw samples from $p(\theta|D_n)$?*”

10.2.2 Sampling with Markov processes

The goal in MCMC methods is to address this question through the theory of Markov processes. Essentially, we will try to “design” a Markov process $\theta_t \in \mathbb{R}^d$ – for either continuous-time $t \in \mathbb{R}_+$ or discrete-time $t \in \mathbb{N}$ – in such a way that the probability distribution of θ_t converges to the posterior distribution. Denoting by $q(\theta, t)$ the probability distribution of θ_t , we want:

$$q(\theta, t) \xrightarrow{t \rightarrow +\infty} p(\theta|D_n)$$

If we can manage to design such a Markov process, the above convergence property will imply that, as we keep simulating the Markov process, the probability distribution of the latest sample will get closer and closer to the posterior distribution. Hence, after simulating the Markov process for a sufficiently long period of time, we can treat the simulated trajectory of the process as “approximate samples” drawn from the posterior distribution. Finally, we can use them for approximating \bar{f} by computing a sample average as in (10.2.1).

10.3 Stochastic Gradient Langevin Dynamics

The Stochastic Gradient Langevin Dynamics (SGLD) algorithm was relatively recently proposed as an alternative to existing MCMC approaches, where its main goal was to be able to scale up to large-scale, modern machine learning problems. The algorithm is based on the famous Langevin equation, which arose from statistical physics.

10.3.1 Notation

Before defining the equation, let us introduce some terms, which will ease the overall notation clutter. We will follow the classical Bayesian statistics notation and from now on, we will denote the posterior distribution by using π , such that $\pi(\theta) = p(\theta|D_n)$.

Definition (Potential energy function). By analogy to physics, we define the *potential energy function* as:

$$U(\theta) := -\sum_{i=1}^n \log p(y_i|\theta) - \log p(\theta) \quad (10.3.1)$$

Note that this is the opposite of the quantity that we are trying to maximize to find the MAP estimator. We have access to U , since we assume that we know both the likelihood function and the prior distribution.

Property 10.1 (Posterior distribution as a function of the potential energy). With the previous notations, we have:

$$\pi(\theta) = \frac{\exp(-U(\theta))}{\int \exp(-U(\theta')) d\theta'} \quad (10.3.2)$$

Even though the marginal likelihood term $p(D_n)$ appears in the denominator, notice that if we want to compute the gradient of $\theta \mapsto \log \pi(\theta)$ with respect to θ , this term vanishes:

$$\nabla \log \pi(\theta) = -\nabla U(\theta)$$

This shows that, even though we do not know what the marginal likelihood is, we can still compute the gradient of $\log \pi$ since we have access to U . This is an important property and will form one of the bases of SGLD.

10.3.2 The Langevin equation

Definition (Brownian motion). The *standard Brownian motion* in \mathbb{R}^d is defined as:

$$\left\{ \begin{array}{ll} B_0 = 0 & \text{almost surely} \\ \forall t_0 < t_1 < \dots < t_N, \quad B_{t_n} - B_{t_{n-1}} & \text{are independent} \\ \forall s < t, \quad B_t - B_s \text{ and } B_{t-s} & \text{follow the same distribution } \mathcal{N}(0, (t-s)I) \\ B_t & \text{is continuous almost surely} \end{array} \right. \quad (10.3.3)$$

Definition (Langevin equation). The *Langevin equation* is a Stochastic Differential Equation (SDE):

$$d\theta_t = -\nabla U(\theta_t) + \sqrt{2} dB_t \quad (10.3.4)$$

It is a shorthand notation for the *Markov process* that is defined as follows, which is much more intuitive:

$$\theta_t = \theta_0 - \int_0^t \nabla U(\theta_s) ds + \sqrt{2} B_t \quad (10.3.5)$$

for some given initialization θ_0 .

Let us illustrate why the Langevin equation is interesting for our purposes. Let us denote the trajectory $(\theta_t)_{t \geq 0}$ as defined in (10.3.5) (this is called a “solution” of the SDE). We will try to understand how the probability distribution of θ_t evolves over time t . To do so, let us denote the distribution of θ_t by the density function $q(\theta, t)$. We will now illustrate that $q(\cdot, t) \rightarrow \pi$ as $t \rightarrow +\infty$, which is going to be crucial for the reasons that we discussed in the previous section – this would allow us to sample from distribution π .

For simplicity, assume that $d = 1$. The definition becomes:

$$d\theta_t = -\partial_\theta U(\theta_t) + \sqrt{2} dB_t$$

To understand how $q(\theta, t)$ evolves, we will use another famous equation, the Fokker-Planck equation, which governs the evolution of $q(\theta, t)$ through the following partial differential equation:

$$\partial_t q(\theta, t) = \partial_\theta [\partial_\theta U(\theta) q(\theta, t)] + \partial_\theta^2 q(\theta, t) \quad (10.3.6)$$

This equation characterizes how the “change” in $q(\cdot, t)$ behaves, i.e. $\partial_t q(\theta, t)$. Now, the idea is that if $q(\cdot, t)$ converges to a distribution as $t \rightarrow +\infty$, then whenever this limit is reached, there should not be any more changes in q . In other words, whenever $q(\cdot, t)$ hits its limit, $\partial_t q(\theta, t)$ has to be equal to 0.

Therefore, we can simply “check” if π is a limit of $q(\cdot, t)$ by replacing $q(\theta, t)$ with $\pi(\theta)$ in (10.3.6), and observing whether the right-hand side of (10.3.6) is equal to 0 or not. Let us apply this procedure:

$$\begin{aligned} \partial_\theta [\partial_\theta U(\theta) \pi(\theta, t)] + \partial_\theta^2 \pi(\theta, t) &= \partial_\theta [\partial_\theta U(\theta) \pi(\theta) + \partial_\theta \pi(\theta)] \\ &= \partial_\theta [\partial_\theta U(\theta) \pi(\theta) - \partial_\theta U(\theta) \pi(\theta)] = 0 \end{aligned}$$

since

$$\begin{aligned} \partial_\theta U(\theta) &= -\partial_\theta \log \pi(\theta) \\ &= -\frac{1}{\pi(\theta)} \partial_\theta \pi(\theta) \end{aligned}$$

hence:

$$\partial_\theta \pi(\theta) = -\partial_\theta U(\theta) \pi(\theta)$$

This shows that if we can simulate the process in (10.3.5), and obtain $(\theta_t)_{t \geq 0}$, the distribution of θ_t will get closer and closer to π , which is our posterior distribution. Hence, if we could simulate the process, we could use θ_t as an “approximate sample” for π , for t large enough. Given our ultimate goal of approximating expectations by finite averages, such θ_t can be used conveniently in the samples averages.

10.3.3 Numerical simulation

Unfortunately, an exact simulation of $(\theta_t)_{t \geq 0}$ is not possible in general since t is continuous and (10.3.5) cannot be analytically computed in general. However, we can develop approximate simulation algorithms with nice computational properties. In the stochastic analysis literature, such approximations are called “discretizations”, since we would like to “discretize” the continuous-time process.

One of the most common and also simplest discretization schemes is called the Euler-Maruyama scheme, which is based on the following idea. Given θ_0 , let us try to compute θ_η , for a very small η . Then, following (10.3.5), we have:

$$\theta_\eta = \theta_0 - \int_0^\eta \nabla U(\theta_t) dt + \sqrt{2} B_\eta \quad (10.3.7)$$

Using the definition of the Brownian motion, we know that $\sqrt{2}B_\eta$ is equal in distribution to $\sqrt{2\eta}Z$, where $Z \sim \mathcal{N}(0, I)$, since $B_\eta \sim \mathcal{N}(0, \eta I)$. On the other hand, since we chose η very small, we can assume that $\nabla U(\theta_t)$ “does not change much” for $t \in [0, \eta]$. Hence, we can replace $\nabla U(\theta_t)$ with $\nabla U(\theta_0)$ in the above equation, giving us:

$$\begin{aligned}\theta_\eta &\simeq \theta_0 - \int_0^\eta \nabla U(\theta_0) dt + \sqrt{2\eta}Z \\ &= \theta_0 - \left(\int_0^\eta dt\right) \nabla U(\theta_0) + \sqrt{2\eta}Z \\ &= \theta_0 - \eta \nabla U(\theta_0) + \sqrt{2\eta}Z\end{aligned}$$

We can now iterate this approach k times, giving us a recursion easily implementable on a computer:

$$\theta_{k\eta} \simeq \theta_{(k-1)\eta} - \eta \nabla U(\theta_{(k-1)\eta}) + \sqrt{2\eta}Z_k \quad (10.3.8)$$

where $\forall k, Z_k \sim \mathcal{N}(0, I)$. To simplify the notation, we can drop the dependence on η , which finally yields the following recursion:

$$\theta_k = \theta_{k-1} - \eta \nabla U(\theta_{k-1}) + \sqrt{2\eta}Z_k \quad (10.3.9)$$

This recursion is called *Langevin Monte Carlo*, or sometimes the *Unadjusted Langevin Algorithm*.

Notice that without the terms Z_k in the above equation, the recursion just reduces to the gradient descent algorithm! Hence, we can view this algorithm as a “noisy” version of gradient descent, such that when the “correct amount of noise” is added to the iterates, the algorithm starts providing approximate samples from the posterior distribution.

Having the recursion (10.3.9), we can now try to approximate the expectations in (10.1.5). Assume that we have run the recursion in (10.3.9) for numerous iterations, say until $k = K_0$, such that we can assume that the distribution of θ_k is close to the posterior π . Then, we can generate K more samples to approximate the expectations:

$$\bar{f} = \int f(\theta) \cdot p(\theta|D_n) d\theta \simeq \frac{1}{K} \sum_{k=K_0+1}^{K_0+K} f(\theta_k) =: \hat{f}_K \quad (10.3.10)$$

The initial number of iterations K_0 is called the *burn-in period*, where the Markov process is “getting ready to sample” for π .

10.3.4 Stochastic Gradient Langevin Dynamics

Even though the recursion in (10.3.9) provides a practical algorithm, its computational complexity can be prohibitively high when the number of data points n is very large, which is the typical case in modern machine learning applications. To see this, we need to remember the expression of the function ∇U :

$$\nabla U(\theta) = - \sum_{i=1}^n \nabla \log p(y_i|\theta) - \nabla \log p(\theta)$$

Since we need to compute this gradient at *every iteration*, this means that we need to compute the gradient of the log-likelihood for n different data points at every iteration. Clearly, this can be both problematic and unnecessary when n is very large.

The idea in *Stochastic Gradient Langevin Dynamics* (SGLD) is to address this issue by replacing ∇U with an unbiased estimator, which is much simpler to compute. More precisely, it replaces ∇U with the “stochastic gradient”, that is the gradient of the following function:

$$\hat{U}_k(\theta) := -\frac{n}{n} \sum_{i \in \Omega_k} \log p(y_i|\theta) - \log p(\theta) \quad (10.3.11)$$

where $\Omega_k \subseteq \llbracket 1, n \rrbracket$ is a random data subsample that is drawn either with or without replacement, and $b = |\Omega_k|$ denotes the number of elements in Ω_k , and is typically much smaller than n . Notice that when conditioned on θ , $\nabla \hat{U}_k(\theta)$ is an unbiased estimator of $\nabla U(\theta)$, i.e. :

$$\forall k, \quad \mathbb{E}[\nabla \hat{U}_k(\theta) | \theta] = \nabla U(\theta)$$

By replacing ∇U with $\nabla \hat{U}_k$ in (10.3.9), we only need to compute the gradients of b terms instead of n terms at each iteration. This finally gives us the recursion of the SGLD algorithm:

$$\theta_k = \theta_{k-1} - \eta \nabla \hat{U}_k(\theta_{k-1}) + \sqrt{2\eta} Z_k \quad (10.3.12)$$

We can define the SGLD estimator \hat{f} as before, using sampled θ_k from (10.3.12) instead of (10.3.9).

A final remark on this topic is that, since we are making a series of approximations, it is no longer guaranteed that the distribution of θ_k converges to π . Namely, we discretized the process, and replaced the true gradient with the stochastic gradient. Clearly, at each step we introduce a certain amount of error. However, fortunately, it has been well-studied that this error can be controlled when η is small enough and b is large enough.

11 Neural Networks

11.1 Introduction

This chapter will introduce some basic elements about neural networks. More information can be found in the class about Deep Learning, given during first semester of second year.

In the past classes, the main focus has been on methods to learn from n observations $(x_i, y_i)_{i \in \llbracket 1, n \rrbracket}$, where $x_i \in \mathcal{X}$ – the input space – and $y_i \in \mathcal{Y}$ – the output or label space.

A large class of methods relies on minimizing a regularized empirical risk with respect to a function $f : \mathcal{X} \rightarrow \mathbb{R}$, where the following cost function is minimized:

$$\frac{1}{n} \sum_{i=1}^n \ell(y_i, f(x_i)) + \Omega(f)$$

where $\ell : \mathcal{Y} \times \mathbb{R} \rightarrow \mathbb{R}$ is a loss function, and $\Omega(f)$ is a regularization term. Typical examples are regression or classification, studied in the previous chapters. The class of functions that we have considered so far were affine functions – which are simple to implement and fast to train, but which can solve only linear problems – and non-linear functions through kernel methods – which allow non-linear predictions but have a large complexity.

All the classes can be described by the *prototypical model*.

Definition (Prototypical model). A class of functions follows a prototypical model if all its elements can be described as a linear combination of non-linear parametric functions σ , i.e. of the form:

$$f(x) = \sum_{i=1}^M \alpha_i \sigma(x, w_i)$$

where the w_i are the parameters.

In the case of kernel methods, $M = n$, σ is the kernel, and $w_i = x_i$. The goal of this final chapter is to explore another class of functions for non-linear predictions, namely neural networks. It sets the cornerstone of Deep Learning, a vast field with numerous applications.

11.2 A single neuron

11.2.1 Background

At the beginnings of computer science, scientists tried to replicate actual biological neurons using computers. The first artificial neuron model was presented by McCulloch and Pitts in 1942.

Definition (Artificial neuron model). An artificial neuron is a function of the form:

$$f(x) = \sigma(w^\top x + b)$$

with σ a non-linear and typically non-decreasing function, called the *activation function*. w is often called the *weight* and b the *bias*.

Typical activation functions are:

- Sigmoid: $\sigma(u) = 1/(1 + e^{-u})$
- Step: $\sigma(u) = \mathbb{1}_{u>0}$
- Rectified Linear Unit (ReLU): $\sigma(u) = (u)_+ = \max(u, 0)$

Definition (Perceptron). The *perceptron* is an algorithm used to train a single neuron using a stochastic gradient descent update rule.

11.2.2 Logistic regression using sigmoid activation and cross-entropy loss

If σ is the sigmoid function,

$$f(x) := \sigma(w^\top x + b) \in]0, 1[$$

can be seen as a probability. It is therefore natural to consider the model on $Y \in -1, 1$ where:

$$p(Y = 1|x) = \sigma(w^\top x + b)$$

thus:

$$p(y|x) = \sigma(y \cdot (w^\top x + b))$$

Solving for the maximum likelihood – or equivalently, using cross-entropy:

$$\ell(x, y) := -\mathbb{1}_{y=1} \log p(Y = 1|x) - \mathbb{1}_{y=-1} \log p(Y = -1|x)$$

then the loss to minimize is exactly:

$$-\log \sigma(y \cdot (w^\top x + b)) = \log(1 + \exp(-y \cdot (w^\top x + b)))$$

which is exactly logistic regression.

As a consequence, neural networks as we just saw – that is, without hidden layers – are reduced to linear predictors. The last layers of deeper networks will be treated in practice in the same way: for classification, cross-entropy loss, and for regression, no activation function.

11.2.3 Gradient and SGD

Similarly to methods we previously encountered, parameters w and b are learned using (stochastic) gradient descent. The gradient of:

$$\ell(y, f(x)) = L_y(f(x, w, b)) = L_y(\sigma(w^\top x + b))$$

can be computed using the chain rule¹². Denoting $f = x \mapsto \sigma(w^\top x + b)$, we have:

$$\begin{aligned} \frac{\partial [L_y \circ f]}{\partial w} &= L'_y(f(x, w, b)) \frac{\partial f}{\partial w} = L'(f(x, w, b)) \sigma'(w^\top x + b) x \\ \frac{\partial [L_y \circ f]}{\partial b} &= L'_y(f(x, w, b)) \frac{\partial f}{\partial b} = L'(f(x, w, b)) \sigma'(w^\top x + b) \end{aligned}$$

¹²Recall that the chain rule states that $(f \circ g)' = (f' \circ g) \cdot g'$

Batch gradient To compute the gradient using the dataset, a first approach is to use Batch gradient, in which the gradient is computed using the entire dataset. The full gradient is the average of the gradients of all the data points.

If we define the regularized empirical loss $J(w, b)$ to be:

$$J(w, b) := \frac{1}{n} \sum_{i=1}^n L_{y_i}(\sigma(w^\top x_i + b)) + \Omega(w, b) \quad (11.2.1)$$

where $\Omega(w, b)$ is some regularization term, we have the following derivatives for J :

$$\begin{aligned} \frac{\partial J}{\partial w} &= \frac{1}{n} \sum_{i=1}^n L'_{y_i}(f(x, w, b)) \cdot \sigma'(w^\top x_i + b) \cdot x_i + \frac{\partial \Omega}{\partial w} \\ \frac{\partial J}{\partial b} &= \frac{1}{n} \sum_{i=1}^n L'_{y_i}(f(x, w, b)) \cdot \sigma'(w^\top x_i + b) + \frac{\partial \Omega}{\partial b} \end{aligned}$$

This naive algorithm requires access to the entire dataset at each iteration:

$$\begin{aligned} w &\longleftarrow w - \gamma \frac{\partial J}{\partial w} \\ b &\longleftarrow b - \gamma \frac{\partial J}{\partial b} \end{aligned}$$

where, as seen in the expression of the derivatives of J , all the dataset points are used. This is obviously quite costly and most of the time unnecessary.

Mini-batches stochastic gradient descent The idea of stochastic gradient descent using mini-batches is to consider at each iteration a set $I \subseteq \llbracket 1, n \rrbracket$ which will represent the indices of the dataset points which will actually be used in the computation of the gradient. This allows to reduce to computational cost of a single iteration.

If we define $J_I(w, b)$, the regularized empirical loss over the mini-batch I , to be:

$$J_I(w, b) := \frac{1}{|I|} \sum_{i \in I} L_{y_i}(\sigma(w^\top x_i + b)) + \Omega(w, b) \quad (11.2.2)$$

the derivatives of J_I are as simple to express:

$$\begin{aligned} \frac{\partial J_I}{\partial w} &= \frac{1}{|I|} \sum_{i \in I} L'_{y_i}(f(x, w, b)) \cdot \sigma'(w^\top x_i + b) \cdot x_i + \frac{\partial \Omega}{\partial w} \\ \frac{\partial J_I}{\partial b} &= \frac{1}{|I|} \sum_{i \in I} L'_{y_i}(f(x, w, b)) \cdot \sigma'(w^\top x_i + b) + \frac{\partial \Omega}{\partial b} \end{aligned}$$

A step of the stochastic gradient descent then takes a similar form:

$$\begin{aligned} w &\longleftarrow w - \gamma \frac{\partial J_I}{\partial w} \\ b &\longleftarrow b - \gamma \frac{\partial J_I}{\partial b} \end{aligned}$$

for some I , randomly-chosen at each iteration.

Note that in general, SGD does not converge with constant step size. It needs a decreasing step size to be able to converge. Furthermore, due to the lack of convexity of the loss function, we cannot guarantee that there is convergence to a global optimum.

11.3 One hidden layer

11.3.1 Motivation

Definition (XOR problem). A class \mathcal{C} of functions of the form:

$$f : \mathbb{R}^2 \longrightarrow \mathbb{R}$$

solves the XOR *problem* when a function in this class is equal to the XOR function, that is:

$$\exists f \in \mathcal{C}, \forall x, y \in \{0, 1\}, \quad f(x, y) = x \oplus y$$

Property 11.1. A single neuron – and, more generally, linear regressions – cannot solve the XOR problem.

This shows that many “interesting” functions cannot be solved using single neurons, hence the need to build more complicated architectures of neural *networks*.

11.3.2 Definition of hidden layers

To be able to solve problems such as the XOR problem, we introduce a *hidden layer*: our network is now composed of two layers, the hidden layer and the output layer. The idea will be to feed the result of the hidden neuron to the output layer – which we already had in the simple neuron model.

Formally, we introduce a new variable h such that, for $x \in \mathbb{R}^d$, $h \in \mathbb{R}^m$, $y \in \mathbb{R}$, where m is the size of the hidden layer, which have the following relationship:

$$\begin{aligned} h &= \sigma[(W_h)^\top x + B_h] \\ y &= \sigma[(w_o)^\top h + b_o] \end{aligned}$$

where $W_h \in \mathcal{M}_{d,m}$, $B_h \in \mathbb{R}^m$, $w_o \in \mathbb{R}^m$ and $b_o \in \mathbb{R}$.

Computing the output y requires a *forward pass*: we start by computing h , and use the result of h to compute y . Schematically, this is done from left (input) to right (output), so the computation is done *forward*, as opposed to *backward* computations, which are also used, and start by the output and moves back to the input.

11.3.3 Gradient through backpropagation

Once again, the derivatives of the output with respect to the weights and biases of both layers can be computed using the chain rule.

Respect to the output layer The derivatives of y with respect to parameters of the output layers are quite similar to the single neuron model, since the output layer is simply a single neuron using the hidden layer as input. It can be computed in one step:

$$\begin{aligned} \frac{\partial y}{\partial w_o} &= \sigma'[(w_o)^\top h + b_o] \cdot h \\ \frac{\partial y}{\partial b_o} &= \sigma'[(w_o)^\top h + b_o] \end{aligned}$$

Respect to the hidden layer The derivatives with respect to parameters of the hidden layer can be computed in two steps using chain rule: we will start by computing the derivatives of the hidden layer h , and apply chain rule to connect it to the derivative of the output layer. Denoting by $W_{h,i} \in \mathbb{R}^d$ the i -the input (or hidden) weight:

$$\begin{aligned}\frac{\partial h_i}{\partial W_{h,i}} &= \sigma'[(W_{h,i})^\top x + B_{h,i}] \cdot x \\ \frac{\partial y}{\partial B_{h,i}} &= \sigma'[(W_{h,i})^\top x + B_{h,i}]\end{aligned}$$

Then, applying chain rule to the output layer:

$$\begin{aligned}\frac{\partial y}{\partial W_{h,i}} &= \sum_{j=1}^m \frac{\partial y}{\partial h_j} \cdot \frac{\partial h_j}{\partial W_{h,i}} = \sigma'[(w_o)^\top h + b_o](w_{o,i}) \cdot \frac{\partial h_i}{\partial W_{h,i}} \\ \frac{\partial y}{\partial B_{h,i}} &= \sum_{j=1}^m \frac{\partial y}{\partial h_j} \cdot \frac{\partial h_j}{\partial B_{h,i}} = \sigma'[(w_o)^\top h + b_o](w_{o,i}) \cdot \frac{\partial h_i}{\partial B_{h,i}}\end{aligned}$$

The exact expression of the derivative can therefore be found using the intermediate computation of the derivatives of the hidden layer.

Note that all these operations are vectorized, and are therefore adapted to GPUs. This is one of the reason of the rapid development of neural networks at the beginning of the 2000s: the arrival of GPUs – which are well-fitted to train neural networks – enabled fast and powerful computations for deep learning.

11.3.4 Approximation properties

It can be shown that neural networks with one hidden layer solve the XOR problem. In fact, this result can be generalized to the *Universal approximation theorem*, stating that any continuous function can be approximated up to arbitrary precision using a neural network with sufficiently many hidden neurons.

11.3.5 Link with kernel methods

When no activation is used as the output layer, we have:

$$\begin{aligned}h &= \sigma[(W_h)^\top x + B_h] \\ y &= (w_o)^\top h + b_o\end{aligned}$$

This corresponds to a linear classifier with feature vector:

$$\Phi(x) = \frac{1}{\sqrt{m}} \sigma[(W_h)^\top x + B_h]$$

parametrized by W_h and B_h , with kernel:

$$k(x, x') = \frac{1}{m} \sum_{i=1}^n \sigma[(W_{h,i})^\top x + B_{h,i}] \cdot \sigma[(W_{h,i})^\top x' + B_{h,i}]$$

Note that the feature vector is of finite dimension and *learned from data*.

With random i.i.d. weights $W_{h,i} \in \mathbb{R}^m$ and $B_{h,i} \in \mathbb{R}$:

$$k(x, x') = \mathbb{E} \left[\sigma[(W_h)^\top x + B_h] \cdot \sigma[(W_h)^\top x' + B_h] \right]$$

This can be computed in closed for some simple distributions of weights. Therefore, an infinite number of random input weights lead to a kernel method.

11.4 Multiple hidden layers

Similarly to the appending of a hidden layer, we can add as many hidden layers as we wish. Both the evaluation process and the differentiation using backpropagation are quite similar. For instance, two hidden layers can be expressed as:

$$\begin{aligned}y &= \sigma(W_1^\top \sigma(W_2^\top \sigma(W_3^\top x))) = f_1 \circ f_2 \circ f_3(x) \\y &= f_1(\theta_1, y_2) \\y_2 &= f_2(\theta_2, y_3) \\y_4 &= f_3(\theta_3, x)\end{aligned}$$

Likewise, gradient of y with respect to the weights of the hidden layers can be expressed using chain rule.

Different theorems show the approximation properties of multi-layers neural networks. These are out of the scope of this class, and are mostly technicals: the ground idea is that any function can be approximated using enough hidden layers of large enough sizes. This is the reason of the versatility of neural networks in machine learning: they are quite efficient to train (using SGD and GPUs), and can in theory solve any problem, that is, approximate any function.

12 Appendix

12.1 Complexity of linear algebra computations

In the following, we consider $K, L \in \mathcal{M}_n(\mathbb{R})$ to be two square matrices, and $y \in \mathbb{R}^n$ to be a vector. We give asymptotic computation time for different operations of linear algebra for simple algorithms.

| Operation | Notation | Complexity |
|-----------------------------------|----------------|------------|
| Matrix multiplication | KL | $O(n^3)$ |
| Inversion | K^{-1} | $O(n^3)$ |
| Vector multiplication | Ky | $O(n^2)$ |
| Eigenvalues/vectors decomposition | $\text{Sp}(K)$ | $O(n^3)$ |
| Largest eigenvector | | $O(n^2)$ |