Introduction to Statistical Supervised Machine Learning

Master 1 MIASHS (2022-2023)

Guillaume Metzler

Institut de Communication (ICOM)
Université de Lyon, Université Lumière Lyon 2
Laboratoire ERIC UR 3083, Lyon, France

guillaume.metzler@univ-lyon2.fr

Abstract

This course aims to provide some basics in Supervised Machine Learning. ... to complete at the end ...

Talk about videos from Hugo Larochelle.

Contents

Part I

Mathematical Background

The objective of this part is to give some reminders on the notions of convexity and smooth convex optimization. These reminders are essential in order to understand certain calculations, properties of algorithms but also the way in which solvers solve certain tasks. For further details on convexity and convex optimization optimization, do not hesitate to look at [Boyd and Vandenberghe, 2004].

1 Linear Algebra and Convexity

This first part will rapidly introduce the mathematical tools required for the rest of the presentation[1] In this section we suppose that we work in a real vectorial space E of finite dimension d.

1.1 Norms

Given two vectors $\mathbf{x}$ and $\mathbf{y}$ of $\mathbb{R}^d$, we define the inner product as the sum of the product of the term components of the two vectors, *i.e.*

$$\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x}^T \mathbf{y} = \sum_{j=1}^{d} x_j y_j.$$

When we compute the inner product of $\mathbf{x}$ with itself, we compute the square norm of $\mathbf{x}$:

$$\|\mathbf{x}\|^2 = \langle \mathbf{x}, \mathbf{x} \rangle.$$

Définition 1.1: Norm

Let E be a $\mathbb{R}$-vectorial space of dimension d, then the application $\|\cdot\|$ is said to be a norm if for all $\mathbf{yv} \in E$ and for all $\lambda \in \mathbb{R}$, the following points are verified:

- $\|\mathbf{u}\| \geq 0$ (positive)
- $\|\mathbf{u}\| = 0 \iff \mathbf{u} = 0$ (definite)
- $\|\lambda \mathbf{u}\| = \lambda \|\mathbf{u}\|$ (scalability)
- $\|\mathbf{u} + \mathbf{v}\| \leq \|\mathbf{u}\| + \|\mathbf{v}\|$ (triangle inequality)

The norm can be seen as distance between two vectors $\mathbf{x}, \mathbf{y}$ in the same vectorial space:

$$d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|.$$

We usually used the following norms:

[1] You can also read to course of Linear Algebra that I have made for the second year Bachelor Students, which is available on my website.

- The norm Manhattan also known as the L_1-norm:

$$\|\mathbf{x}\|_1 = \sum_{j=1}^{d} |x_j|.$$

- The Euclidean norm also known as the L_2-norm:

$$\|\mathbf{x}\|_2 = \sqrt{\sum_{j=1}^{d} x_j^2}.$$

- The L_∞-norm:

$$\|\mathbf{x}\| = \max_{j \in [\![1,d]\!]} |x_j|.$$

- More generally, for all positive integers p, we can define the L_p-norm:

$$\|\mathbf{x}\|_p = \left(\sum_{j=1}^{d} |x_j|^p \right)^{1/p}.$$

But it is also possible to define more complex norms when we are working we other objects than vectors.

Exemple 1.1. *We will show that the Euclidean norm is effectively a norm. We have to check each point of the definition. Let us consider* $\mathbf{x}, \mathbf{y} \in \mathbb{R}^d$ *and* $\lambda \in \mathbb{R}$

- *It is obvious that* $\|\mathbf{x}\|_2 = \sqrt{\sum_{j=1}^{d} |x_i|^2}$ *is positive.*
- *Each* $|x_j|^2$ *is a positive number and a sum of positive number is equal to zero if and only if all of the numbers are equal to zero* $\Rightarrow \mathbf{x} = \mathbf{0}$
- *We then focus on the scalability property:*

$$\begin{aligned}
\|\lambda \mathbf{x}\|_2 &= \sqrt{\sum_{j=1}^{d} (\lambda x_j)^2} \\
&= \sqrt{\sum_{j=1}^{d} |\lambda|^2 x_i^2} \\
&\quad \downarrow \ \lambda \text{ does not depend on the index } j \\
&= |\lambda| \sqrt{\sum_{j=1}^{d} x_i^2}.
\end{aligned}$$

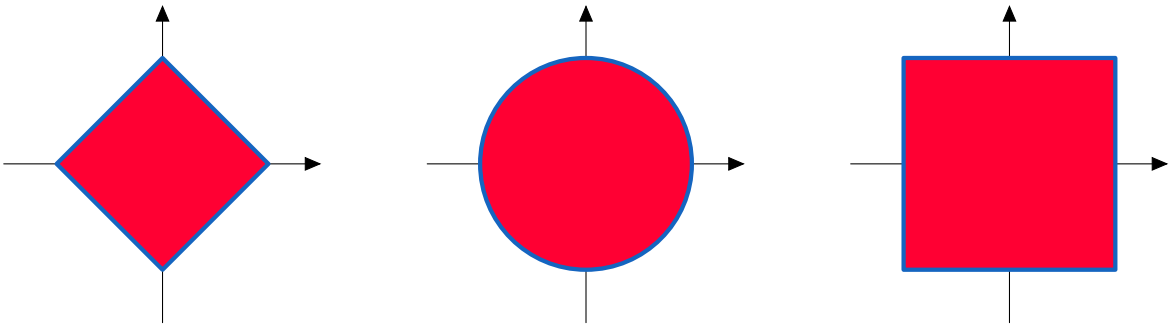


Figure 1: Unit ball for the L_p-norm when $p = 1$, $p = 2$ and $p = \infty$

• *To prove the last point we will use the Cauchy-Schwartz Inequality which is true for all norms:*

$$\langle \mathbf{x}, \mathbf{y} \rangle \leq \|\mathbf{x}\| \, \|\mathbf{y}\| \, .$$

Thus,

$$\begin{aligned} \|\mathbf{x} + \mathbf{y}\|_2^2 = \; & \|\mathbf{x}\|_2^2 + 2\langle \mathbf{x}, \mathbf{y} \rangle + \|\mathbf{y}\|_2^2 \, , \\ & \downarrow \textit{ using Cauchy-Schwartz Inequality} \\ \leq \; & \|\mathbf{x}\|_2^2 + 2 \, \|\mathbf{x}\|_2 \, \|\mathbf{y}\|_2 + \|\mathbf{y}\|_2^2 \\ \leq \; & \left(\|\mathbf{x}\|_2 + \|\mathbf{y}\|_2 \right)^2 . \end{aligned}$$

By taking the square root, which is an increasing function, we get the result.

We can also represent the unit ball associated to the previous defined norms, see Figure 1. It shows that for all $p > 1$, the unit-ball is a convex set, which also means that the underlying application is convex.

Let us now focus on norms on matrices, more specifically, the Frobenius norm.

Définition 1.2: Frobenius norm on Matrices

Given two matrices $\mathbf{X}, \mathbf{Y} \in \mathbb{R}^{m \times d}$, the inner product of these two matrices is defined by :

$$\langle \mathbf{X}, \mathbf{Y} \rangle = Tr\left(\mathbf{X}^T \mathbf{Y}\right) = \sum_{i=1}^{m} \sum_{j=1}^{d} X_{ij} Y_{ij}.$$

The inner product of matrix with itself results in the Frobenius norm of this matrix:

$$\|\mathbf{X}\|_F = \sqrt{Tr\left(\mathbf{X}^T \mathbf{X}\right)} = \left(\sum_{i=1}^{m} \sum_{j=1}^{n} X_{ij}^2 \right)^{1/2}.$$

1.2 Derivatives

The notion of derivative, or more generally of gradient, is fundamental in optimization. It is at the heart of gradient descent algorithms but it will also allow us to characterize convex functions. As we will see, this characterization can be done by studying the first order derivative or, more simply, by studying the second order derivative.

A few reminders Let us start with a reminder on the derivative of a function.

Définition 1.3: Derivative

Let $f : \mathbb{R} \to \mathbb{R}$ be continuous and $x_0 \in \mathbb{R}$. We say that f is differentiable at x_0 if the limit :

$$\lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h},$$

exists and is finite.
Furthermore, if f is continuously differentiable at x_0, so for $h \simeq 0$ we have:

$$f(x_0 + h) = f(x_0) + h f'(x_0) + \varepsilon(h),$$

where $\varepsilon(h)$ is a function that tends to 0 when h tends to 0.

Tee last formula is called a first order approximation of the function f and it can be generalized to any other order. this is what we call the Taylor expansion of a function:

Proposition 1.1: Taylor's Theorem

Let $f : \mathbb{R} \to \mathbb{R}$ be function that is k differentiable, with $k \geq 1$, at a point x_0. Then, there exists a function $\varepsilon_k : \mathbb{R} \to \mathbb{R}$ such that:

$$f(x_0 + h) = f(x_0) + hf'(x_0) + f''(x_0)\frac{h^2}{2} + \ldots + f^{(k)}(x_0)\frac{h^k}{k!} + \varepsilon_k(h),$$

where $\lim_{h \to 0} \varepsilon_k(h) = 0$.

First and second order derivatives of a vectorial function In Machine Learning, as the reader may have seen in linear models, we the number of parameters we have to learn is always greater than 1. We are therefore most often led to study linear forms, i.e. functions which depend on a vector of values and which have real values.

Définition 1.4: First order Derivative

Let $f : \mathbb{R}^d \to \mathbb{R}$ be a C^0 application and $\mathbf{x} \in \mathbb{R}^d$. Then f is differentiable at x_0 if it exists $J \in \mathbb{R}^d$ such that :

$$\lim_{x \to x_0} \frac{\|f(\mathbf{x}) - f(\mathbf{x}_0) - Jf(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0)\|}{\|\mathbf{x} - \mathbf{x}_0\|} = 0.$$

For all i the elements of the matrix J are given by :

$$J_i f(\mathbf{x}_0) = \left.\frac{\partial f(\mathbf{x})}{\partial x_i}\right|_{\mathbf{x}=\mathbf{x}_0}$$

The gradient gives the possibilty to approximate the function near the point its gradient is calculated, *i.e.* in a neighborhood $V(x_0)$ of $\mathbf{x}_0$. For all $\mathbf{x} \in V(\mathbf{x}_0)$ we have:

$$f(\mathbf{x}) \simeq f(\mathbf{x}_0) + \nabla f(\mathbf{x}_0)(\mathbf{x} - \mathbf{x}_0)$$

It is this approximation that will help to give a first characterization of convex functions.

Exemple 1.2. *Let us consider the application* $f : \mathbb{R}^3 \to \mathbb{R}$ *defined by*

$$f(x, y, z) = 3x_1^2 + 2x_1x_2x_3 + 6x_3 + 5x_2x_3 + 9x_1x_3.$$

We want to calculate the Jacobian (or gradient) of this function. To do so, we need to compute:

$$\frac{\partial f}{\partial x_1}, \quad \frac{\partial f}{\partial x_2} \quad and \quad \frac{\partial f}{\partial x_3}$$

.

The Jacobian of f at (x_1, x_2, x_3) is given by :

$$J_{f(x,y,z)} = \begin{pmatrix} 6x_1 + 2x_2x_3 + 9x_3 & 2x_1x_3 + 5x_3 & 2x_1x_2 + 6 + 5x_2 + 9x_1 \end{pmatrix}$$

Exemple 1.3. *We consider the log-sum-exp function defined for all $\mathbf{x}, \mathbf{b} \in \mathbb{R}^d$ by:*

$$f(\mathbf{x}) = \ln \sum_{j=1}^{d} \exp\left(x_i + b_i\right)$$

This function is known to be a good approximation (and smooth!) of the max *function when $\underline{=}\mathbf{0}$.*

The gradient $\nabla_{\mathbf{x}} f(\mathbf{x}, \mathbf{b})$ of this function is given by:

$$\nabla_{\mathbf{x}} f(\mathbf{x}, \mathbf{b}) = \left(\frac{\exp(x_1 + b_1)}{\sum_{j=1}^{d} \exp(x_j + b_j)} \quad \cdots \quad \frac{\exp(x_d + b_d)}{\sum_{i=1}^{n} \exp(x_j + b_j)} \right)$$

Définition 1.5: Second order Derivative

Let $f : \mathbb{R}^d \to \mathbb{R}$ be a real function. Provided that this function is twice diffentiable at $\mathbf{x}_0$, the second order derivative $\mathbf{H}$, (also called the Hessian)of f at $\mathbf{x}_0$ is given by :

$$H_{ij} f(x_0) = \nabla^2_{\mathbf{x}} f(\mathbf{x}_0) = \left. \frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j} \right|_{\mathbf{x}=\mathbf{x}_0},$$

where $\mathbf{H} \in \mathbb{R}^{d \times d}$.

This matrix is usefull to prove that a function f is convex or not and also to build efficient algorithms.

Exemple 1.4. *Let us consider the function $f : \mathbb{R}^2 \to \mathbb{R}$ defined by*

$$f(x, y) = 4x_1^2 + 6x_2^2 + 3x_1x_2 + 2\left(\cos(x_1) + \sin(x_2)\right)$$

for which we aim to compute the Hessian matrix.
We first have to calculate the Jacobian this function

$$J_{f(\mathbf{x})} = \begin{pmatrix} \dfrac{\partial f}{\partial x} & \dfrac{\partial f}{\partial y} \end{pmatrix} = \begin{pmatrix} 8x_1 + 3x_2 - 2\sin(x_1) & 12x_2 + 3x_1 + 2\cos(x_2) \end{pmatrix}.$$

The Hessian matrix is then given by:

$$H_{f(x,y)} == \begin{pmatrix} \dfrac{\partial^2 f}{\partial x_1^2} & \dfrac{\partial^2 f}{\partial x_1 \partial x_2} \\ \dfrac{\partial^2 f}{\partial x_2 \partial x_1} & \dfrac{\partial^2 f}{\partial x_2^2} \end{pmatrix} = \begin{pmatrix} 8 - 2\cos(x_1) & 3 \\ 3 & 12 - 2\sin(x_2) \end{pmatrix}.$$

1.3 Convex sets

The notion of convex set is an important element when we want to build optimization algorithms and guarantee that the successive iterations of this algorithm remain in a precise set. Moreover, it will also allow us to characterize the convex functions.

Let us start with the definition of a convex set.

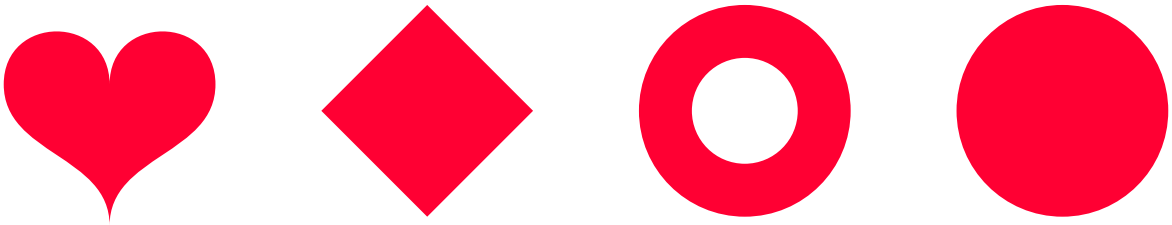
Définition 1.6: Convex Set

A set C is said to be convex if, for every $(\mathbf{u}, \mathbf{v}) \in C$ and for all $t \in [0, 1]$ we have :

$$t\mathbf{u} + (1 - t)\mathbf{v} \in C.$$

In other words, C is said to be convex if every point on the segment connecting $\mathbf{u}$ and $\mathbf{v}$ is in the set.

The figure below provides some examples of convex and non convex sets. The first and third figures represent non convex sets. Indeed, for the third figure, if we take two points that are diametrically opposed, the segment between these two points goes through the center of the ring which does not belong to the set. The second and fourth figure represent convex sets.



Exemple 1.5. *We list some fundamentals sets that are also convex:*

- $\mathcal{B} = \{\mathbf{u} \in \mathbb{R}^d \mid \|\mathbf{u}\| \leq 1\}$ *is convex.*

- *Every segment in* $\mathbb{R}$ *is convex.*

- *Every hyperplane* $\{\mathbf{x} \in \mathbb{R}^d \mid \mathbf{a}^T x = b\}$ *is convex.*

- *If* C_1 *and* C_2 *are two convex sets, then the intersection* $C = C_1 \cap C_2$ *is also convex.*

Let us study the unit ball, we have previously seen that is convex, let us now prove it for the L_2*-norm.*

For the first point, consider $\lambda \in [0, 1]$ *and* $\mathbf{u}, \mathbf{v}$ *two vectors in the unit ball and let us set* $\mathbf{z} = \lambda\mathbf{u} + (1 - \lambda)\mathbf{v}$.

$$
\begin{aligned}
\|\mathbf{z}\|_2 = & \|\lambda\mathbf{u} + (1-\lambda)\mathbf{v}\|_2 , \\
& \downarrow \textit{ apply the triangle inequality} \\
\leq & \|\lambda\mathbf{u}\|_2 + \|(1-\lambda)\mathbf{v}\|_2 , \\
& \downarrow \textit{ the scalability property of norm} \\
\leq & \lambda \|\mathbf{u}\|_2 + (1-\lambda) \|\mathbf{v}\|_2 , \\
& \downarrow \textit{ the scalability property of norm} \\
& \downarrow \textit{ both } \mathbf{u} \textit{ and } \mathbf{v} \textit{ belongs to the unit ball} \\
\leq & \lambda + (1-\lambda), \\
\|\mathbf{z}\|_2 \leq & 1.
\end{aligned}
$$

Proposition 1.2: Convex combination

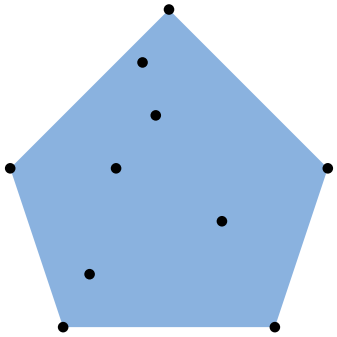
Let $(\mathbf{u}_1, \mathbf{u}_2, \ldots, \mathbf{u}_m)$ be a set of m points belonging to a convex set C. Then for every positive numbers $\lambda_1, \lambda_2, \ldots, \lambda_m$ such that $\sum_{i=1}^n \lambda_i = 1$:

$$\mathbf{v} = \sum_{i=1}^{m} \lambda_i \mathbf{u}_i \in C.$$

It means that every convex combination of points belonging in a convex set, belongs in the convex set.

Given the definition of a convex set and a set of point $\mathbf{x}_1, \dots \mathbf{x}_m$, it is possible to build a convex set. This new set is called the convex hull $\mathcal{H}$ of a set of points

$$\mathcal{H} = \{\mathbf{x} = \sum_{i=1}^{m} \lambda_i \mathbf{x}_i \mid \sum_{i=1}^{m} \lambda_i = 1\}.$$



1.4 Convex functions

The study of convex functions is essential in Machine Learning. Indeed, we will see that most of the problems we will try to solve will involve convex functions. Moreover, it is through the study of this type of problem that we will be able to provide theoretical guarantees on the performance of our algorithms.

Fundamental definitions Let us start by the definition of convex function.

Définition 1.7: Convex functions

et $\mathcal{U}$ be an on empty set of a $\mathbb{R}$-vectorial space of dimension d (*i.e.* $\mathcal{U} \subset \mathbb{R}^d$). A function $f : \mathcal{U} \to \mathbb{R}$ is said to be convex if, for every $(\mathbf{u}, \mathbf{v}) \in \mathcal{U}$ and for all $t \in [0, 1]$, we have :

$$f(t\mathbf{u} + (1-t)\mathbf{v}) \leq tf(\mathbf{u}) + (1-t)f(\mathbf{v}).$$

Exemple 1.6. *The following functions are convex:*

- *A linear function,*
- $f : \mathbb{R} \to \mathbb{R}, \quad f(x) = x^2,$
- $f : \mathbb{R} \to \mathbb{R}, \quad f(x) = 0.4 \times \exp(x).$

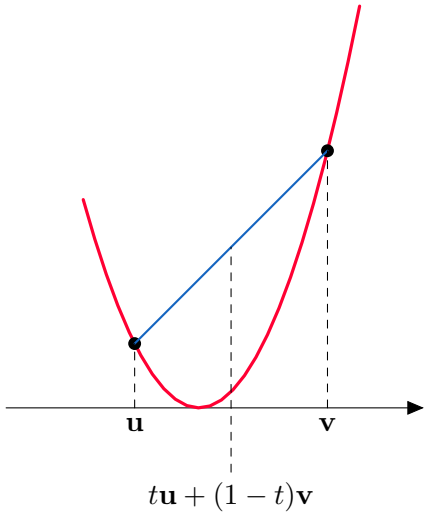
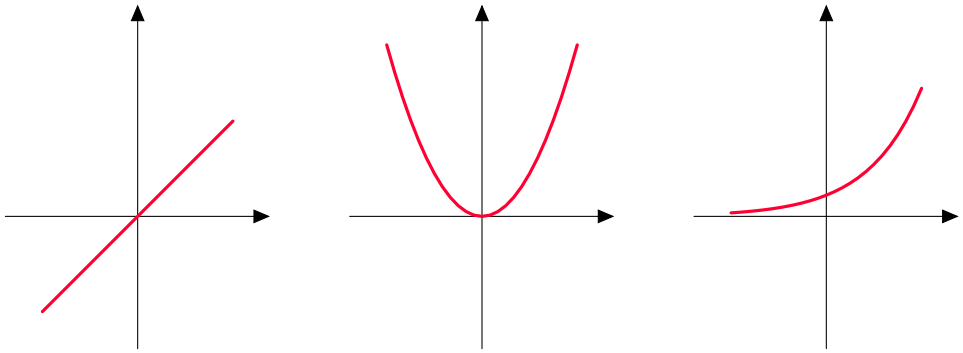


Figure 2: Illustration of a convex function with one of its chord. As you may see, the chords of a convex function are always above the curve



Proposition 1.3: Convexity and Restriction

A function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if and only its restriction to a line is always convex, *i.e.* if the function $g : \mathbb{R} \to \mathbb{R}$ defined by $g(t) = f(\mathbf{x} + t\mathbf{v})$ is convex, for all $\mathbf{x}$ and $\mathbf{v}$ such that $\mathbf{x} + t\mathbf{v}$ belongs to the domain of definition of f (f is concave if and only if g is concave).

Proof. It only requires to write the definition of convex function (à écrire) □

Figure 2 provide an illustration of the definition of convex functions.

The next definition gives the possibility to link the definition of convex function with convex sets.

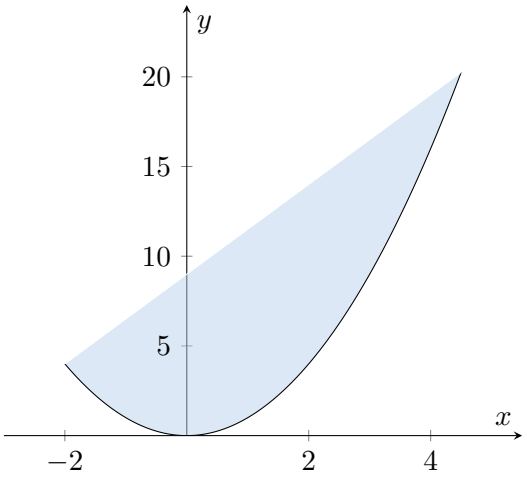


Figure 3: Epigraph of the function $x \mapsto x^2$ represented in shaded blue

Définition 1.8: Epigraph

Let $\mathcal{U}$ be an on empty set of a $\mathbb{R}$-vectorial space of dimension d (*i.e.* $\mathcal{U} \subset \mathbb{R}^d$). Let function $f : \mathcal{U} \to \mathbb{R}$, the epigraph of the function f, noted E_f is defined as the set of points that are above the curve, *i.e.*

$$E_f = \{(\mathbf{x}, y) \mid f(\mathbf{x})y\}\,.$$

Proposition 1.4: Convexity via Epigraph

Let $\mathcal{U}$ be an on empty set of a $\mathbb{R}$-vectorial space of dimension d (*i.e.* $\mathcal{U} \subset \mathbb{R}^d$). A function f is convex on $\mathcal{U}$ if and only if its E is a convex set.

Proof. A ecrire, mais il suffit d'appliquer les définitions et l'équivalence est immédiate. □

The result of this proposition is illustrated on Figure 3 where the studied function is a convex one.

Définition 1.9: Concave functions

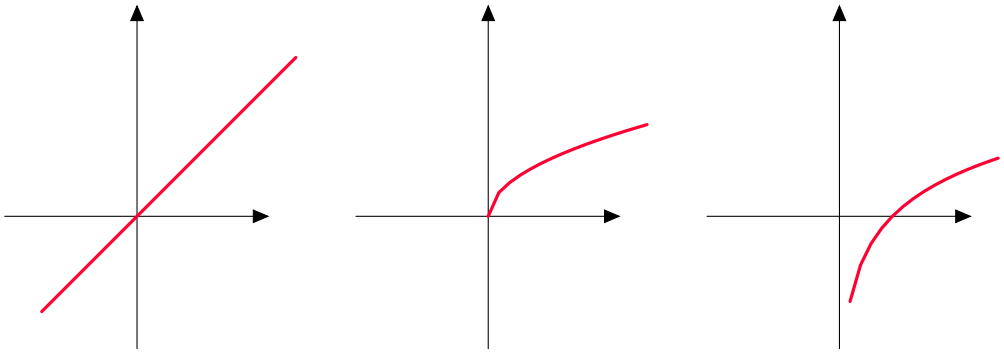
Let $\mathcal{U}$ be an on empty set of a $\mathbb{R}$-vectorial space of dimension d (*i.e.* $\mathcal{U} \subset \mathbb{R}^d$). A function $f : \mathcal{U} \to \mathbb{R}$ is said to be concave if, for every $(\mathbf{u}, \mathbf{v}) \in \mathcal{U}$ and for all $t \in [0, 1]$, we have:

$$f(t\mathbf{u} + (1-t)\mathbf{v}) \geq tf(\mathbf{u}) + (1-t)f(\mathbf{v}).$$

With this definition, you directly see that, if f is concave, then $-f$ is convex. Furthermore, it is also possible to characterize concave function using the the convexity of the hypograph, *i.e.* the area under the function curve.

Exemple 1.7. *The following functions are concave:*

- *A linear function ,*
- $f : \mathbb{R} \to \mathbb{R}, \quad f(x) = \sqrt{x},$
- $f : \mathbb{R} \to \mathbb{R}, \quad f(x) = \ln(x).$



The following proposition provides some properties of convex functions.

Proposition 1.5: Properties Convexity

Let us consider tow convex functions f and g defined on an open subset *spaceU* of $\mathbb{R}^d$:

- any convex combination of f and g remains convex
- if f is furthermore an **increasing** function, then $(f \circ g)(x)$ is convex
- the function h defined by $h(\mathbf{x}) = \max(f(\mathbf{x}), g(\mathbf{x}))$ is convex

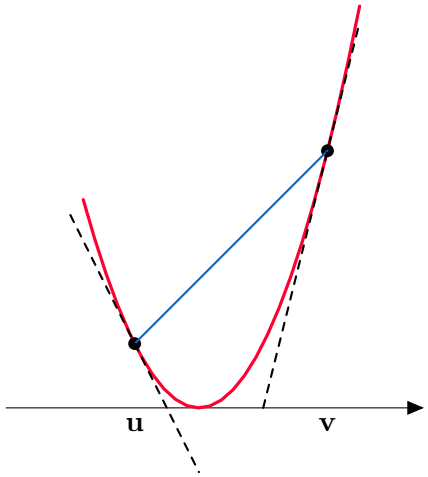


Figure 4: Illustration of a convex function with one of its chord. As you may see, the chords of a convex function are always above the curve

Proof. The proof of this proposition calls for the definition of convexity, it is left to the reader as an exercise □

Characterization of convexity using derivatives In the following, we will study how can characterize the convexity of a function f using the first and second order derivative of the function.

Proposition 1.6: First order derivative and convexity

Let f be a continuously differentiable function on $\mathcal{U}\mathbb{R}^d$. Then f is convex if and only if, for all $(\mathbf{u}, \mathbf{v}) \in \mathcal{U}$, we have :

$$f(\mathbf{v}) \geq f(\mathbf{u}) + \nabla f(\mathbf{u})(\mathbf{v} - \mathbf{u}).$$

Equivalently if and only if, for all $(u, v) \in \mathcal{U}$, we have :

$$(\nabla f(\mathbf{v}) - \nabla f(\mathbf{u}))(\mathbf{v} - \mathbf{u}) \geq 0$$

Proof. Write the proof □

An illustration of Proposition 1.6 is given in Figure 4 where the tangents of a convex function are always below the curve. Note that for concave function, the tangents will be over the curve.

Proposition 1.7: Second order derivative and convexity

Let f be an application twice continuously differentiable on a open subset $\mathcal{U}$ of $\mathbb{R}^d$. Let $\mathbf{H}$ be the matrix of the application $\nabla^2 f$ (*i.e.* the Hessian of f). Then f is said to convex if one of the equivalent propositions holds:

- $\nabla^2 f(\mathbf{u}) \geq 0$ for all $\mathbf{u} \in \mathcal{U}$.
- H is a positive semi definite, *i.e*, $\forall \mathbf{u} \in \mathcal{U}$

$$\mathbf{u}^T \mathbf{H} \mathbf{u} \geq 0.$$

Remember that a matrix $\mathbf{H}$ is said to be PSD if all of it's eigenvalues are positive. It sometime easier to check this point rather than applying the definition of convexity to a complex function.

Having positive eigenvalues means that the gradient is an increasing function along each directions of the space. Furthermore, a function is convex if and only if its gradient is an increasing function along each direction.

Exemple 1.8. *The following matrix* $\mathbf{H}$ *defined by:*

$$\mathbf{M} = \begin{pmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_d \end{pmatrix},$$

where $\lambda_1, \dots, \lambda_d$ *are positive. This kind of matrices can be met when the variables are "independant", for instance for a function* f *of the form:*

$$f(\mathbf{x}) = \sum_{j=1}^{d} \frac{\lambda_j}{2} x_j^2.$$

2 Unconstrained and Smooth Convex Optimization

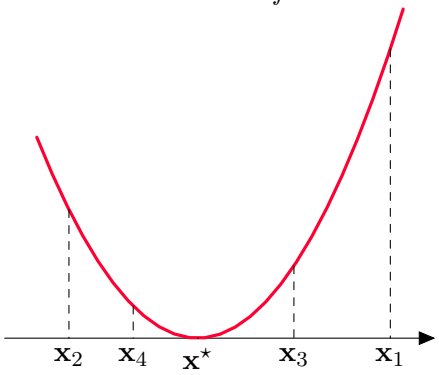
This section is dedicated to the global presentation of convex optimization, it will then focuses on unconstrained convex optimization and presents some fundamental algorithms.

2.1 Motivation

Given a convex function $f : \mathbb{R}^d \to \mathbb{R}$ we would solve the problem :

$$\mathbf{x}^\star = \arg\min_{\mathbf{x}\in\mathbb{R}^d} \quad f(\mathbf{x}).$$

The aim of the course is to introduce some algorithms to build a sequence $(\mathbf{x}_n)_{n\in\mathbb{N}}$ which converges to $\mathbf{x}^\star$, the value of $\mathbf{x}$ for which f reaches its minimum.



We will see how we can build such a sequence, but before doing this, we will see how we can characterize the optimum of a function f, in our case the minimum, *i.e.* what are the conditions of optimality.

2.2 Optimality Condition

Let us first define the minimum of a function f.

Définition 2.1: Minimum

Let $f : \mathbb{R}^d \to \mathbb{R}$ be a continuous function. We say that $\mathbf{u} \in \mathbb{R}^d$ is a local minimum of f if there exists a neighborhood $V \subset \mathbb{R}^d$ of $\mathbf{u}$ such that :

$$f(\mathbf{u}) \leq f(\mathbf{v}), \quad \forall \mathbf{v} \in V.$$

$\mathbf{u}$ is a global minimum of the function f if and only if :

$$f(\mathbf{u}) \leq f(\mathbf{v}), \quad \forall \mathbf{v} \in \mathbb{R}^d.$$

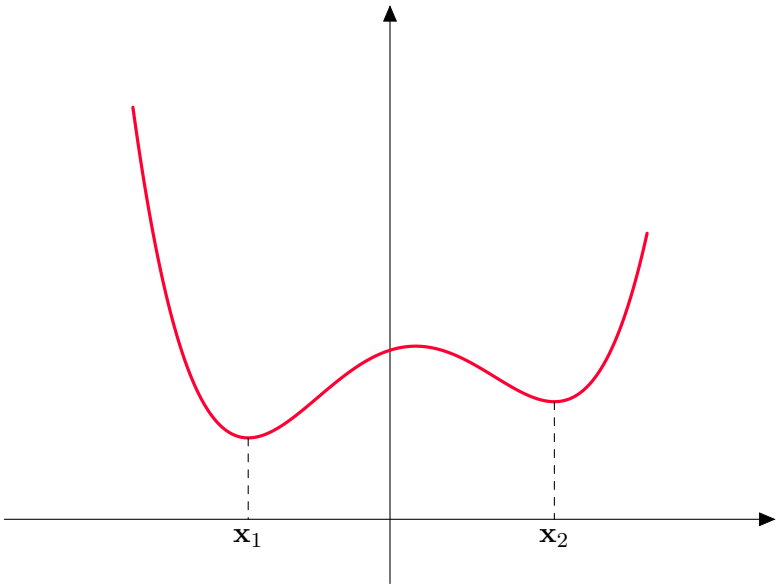


Figure 5: Illustration of local and global minimum of a function. The point x_1 is the global minimum of the function while x_2 is just a local minimum.

In other words, a minimum of the function f is just the value of $\mathbf{x}$ where the function f reaches its lowest value (locally or globally), as illustrated in Figure 5.

If it as graphically, on a one dimensional graph easy to find such minima, we need to find other criteria to detect them.

The following proposition gives a first characterization of relative (or local) minimum:

Proposition 2.1: Euler's Inequation

Let $f : \mathbb{R}^d \to \mathbb{R}$ be a continuous function and $\mathcal{U}$ a non empty and convex set. Furthermore, let $u \in \mathcal{U}$ be a relative minimum of f. If f is differentiable at u we have:

$$\nabla f(u)(v - u) \geq 0, \ \forall \mathbf{v} \in \mathcal{U}$$

However, this first result is rarely used in practice. If we go back to Figure 5, we see that the minima (or maxima) are located on the space where the derivative of the function is equal to 0. This condition of optimality is know under Euler's equation.

Proposition 2.2: Euler's Equation

Let $f : \mathbb{R}^d \to \mathbb{R}$ be a continuous function and differentiable at $\mathbf{u} \in \mathbb{R}^d$. If $\mathbf{u}$ is a local minimum then we have:

$$\nabla f(\mathbf{u}) = 0.$$

Proof. In fact, using the definition of a minimum in a neighborhood: $\forall \mathbf{v} \in \mathbb{R}^d, \exists t > 0$ such that $\mathbf{u} + t\mathbf{v} \in V$ a neighborhood of $\mathbf{u}$, we have:

$$\begin{aligned} & f(\mathbf{u}) && \leq f(\mathbf{u} + t\mathbf{v}) = f(\mathbf{u}) + \nabla f(\mathbf{u})(t\mathbf{v}) + t\mathbf{v}\, \varepsilon(t\mathbf{v}), \quad t \ll 1 \\ \iff & 0 && \leq \nabla f(\mathbf{u})(t\mathbf{v}) + t\mathbf{v}\, \varepsilon(t\mathbf{v}) \end{aligned}$$

Dividing by $t > 0$ and taking the limit $t \to 0$ we have:

$$0 \leq \nabla f(\mathbf{u})\mathbf{v}$$

Same thing by replacing $\mathbf{v} \to -\mathbf{v}$ we have:

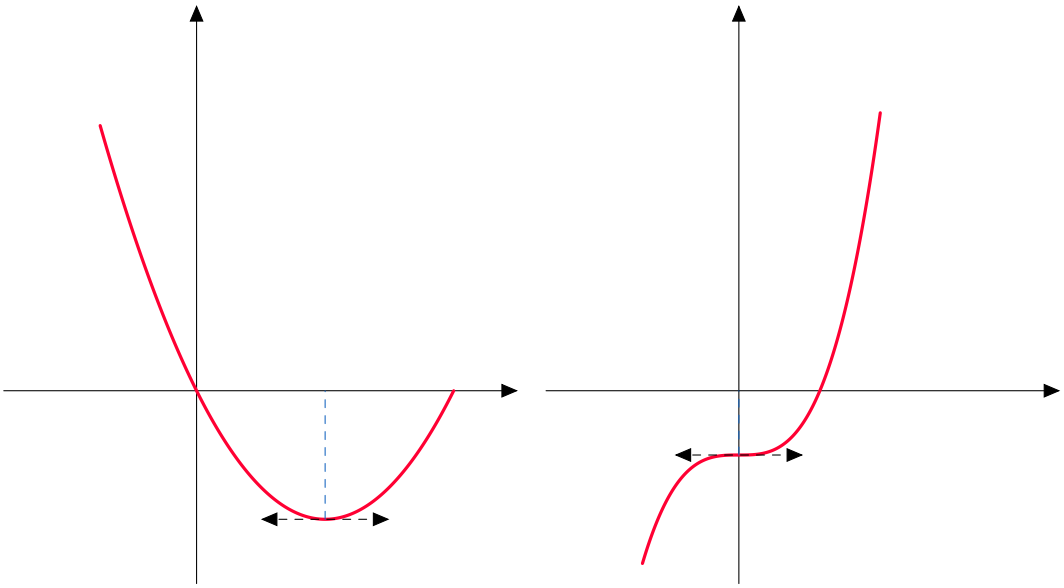
$$0 \leq -\nabla f(\mathbf{u})\mathbf{v}.$$

So for all $\mathbf{v} \in \mathbb{R}^d$ we have $\nabla f(\mathbf{u})\mathbf{v} = 0$ thus $\nabla f(\mathbf{u}) = 0$. □

If this proof has been made using the definition of a (local or global) minimum, we can derive the same result for the maximum. The solutions of Euler's Equation are also called *critical values* or *critical points*.

Note that this proposition only provides a necessary condition for a point to be a local optimal, but that this condition is by no means sufficient as the example below shows.

Exemple 2.1. *Let us consider the functions f and g respectively defined by $\frac{1}{2}(x-2)^2 - 2$ and $\frac{1}{2}x^3 - 1$ and represented below.*



The represented points $\mathbf{x}$ *on both graphs are solutions of Euler's Equation* $\nabla f(\mathbf{x}) = 0$ *and* $\nabla g(\mathbf{x}) = 0$ *respectively. However, if it represents a minimum for the function* f*, this is no more the case for the function* g *where the solution* $\mathbf{x}$ *is neither a minimum nor a maximum.*

This example shows that is important to characterize the different solutions of Euler's Equation, *i.e.* when do they correspond to optima and what kind of optimum is it?

The solution of *Euler's Equation* gives us the points where the function f reaches a local extremum (a minimum or maximum (local or global)).

Given a solution $\mathbf{u}$ of $\nabla f(\mathbf{u}) = 0$, we can say that :

- $\mathbf{u}$ is a local minimum if $\nabla^2 f(\mathbf{u}) = H_f(\mathbf{u}) \geq 0$, i.e. the Hessian matrix evaluated at the point $\mathbf{u}$ is PSD. This point is a global minimum if the function is convex everywhere or if for all $\mathbf{v} \neq \mathbf{u}$ we have $f(\mathbf{u}) \leq f(\mathbf{v})$.
- $\mathbf{u}$ is a local maximum if $\nabla^2 f(\mathbf{u}) = H_f(\mathbf{u}) \leq 0$, i.e. the Hessian matrix evaluated at the point $\mathbf{u}$ is NSD (Negative Semi Definite). This point is a global maximum if the function is concave everywhere or if for all $\mathbf{v} \neq \mathbf{u}$ we have $f(\mathbf{u}) \geq f(\mathbf{v})$.
- in other cases, it is neither a maximum nor a minimum.

Exemple 2.2. *Let us consider the function* $f : \mathbb{R}^2 \to \mathbb{R}$ *defined by:*

$$f(\mathbf{x}) = x_1^2 + 4x^2 - 2x_1x_2.$$

We aim to find the extrema of this function.

We will first compute the gradient ∇f of this function and find for which value of $\mathbf{x}$ *this gradient is equal to* $\mathbf{0}$*. The gradient is given by:*

$$\nabla f(\mathbf{x}) = \begin{pmatrix} 2x_1 - 2x_2 & 8x_2 - 2x_1 \end{pmatrix}$$

Setting this gradient equal to $\mathbf{0}$ *leads to $x_1 = x_2$ using the first component and $x_1 = x_2 = 0$ using the second one. So the point $\mathbf{x}^\star = (0, 0)$ is the only solution of Euler's Equation.*

We can now compute the Hessian matrix to study what kind of extremum it is. The hessian is given by:

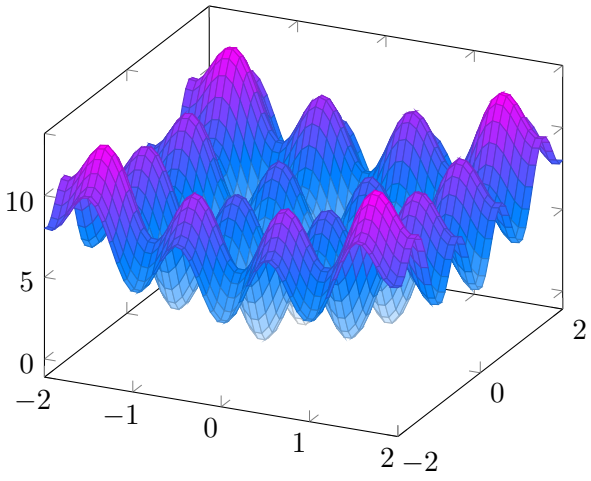
$$\nabla^2 f(\mathbf{x}) = \begin{pmatrix} 2 & -2 \\ -2 & 8 \end{pmatrix}.$$

Looking at this matrices, we can see the two eigenvalues are strictly positives, it means the function f is convex so the point $\mathbf{x}^\star$ is the global minimum.

Exemple 2.3. *Let us study the function $f : [-2, 2]^2 \to \mathbb{R}$ defined by*

$$f(\mathbf{x}) = 4 + \left(x_1^2 - 2\cos(2\pi x_1)\right) + \left(x_2^2 - 2\cos(2\pi x_2)\right).$$

This function is particularly interesting to study because it admits several local extrema but also a global minimum as it be seen below It is therefore very interesting for most optimization algorithms, especially to test their efficiency.



We can see that this function is neither convex nor concave, which could be verified by studying the hessian. We can easily compute the gradient of this function:

$$\nabla f(\mathbf{x}) = \begin{pmatrix} 2x_1 + 4\pi \sin(2\pi x_1) & 2x_2 + 4\pi \sin(2\pi x_2) \end{pmatrix}.$$

The solutions of Euler's Equation are given solving the following system:

$$\begin{aligned} x_1 &= -2\pi \sin(2\pi x_1), \\ x_2 &= -2\pi \sin(2\pi x_2). \end{aligned}$$

The two equations are independant but remain hard to solve, but an obvious solution is given by the point $\mathbf{x}^\star = (0, 0)$. *We can also verify that this point is the global minimum of our function.*

2.3 Minimization Problems

Given a vectorial space E of dimension d and a function $f : E \to \mathbb{R}$, an optimization problem consists of solving the following problem :

$$\min_{\mathbf{x} \in E} f(/vx).$$

The function f is sometimes called the cost function. it can represent the cost for a company to store a series of products (represented by the parameter $\mathbf{x}$) or the risk that is taken by making decisions.

Most of times, we want to minimize this function f under some constraints, but we will study this in Section 3.

Let us first have a look at some basics optimization problems.

Linear Regression Given a vector response $\mathbf{y} \in \mathbb{R}^m$ and feature vectors $\mathbf{X} = (\mathbf{x}_1, \ldots, \mathbf{x}_m), x_i \in \mathcal{R}^d$ where $d < m$. We would like to find a vector $\boldsymbol{\theta} \in \mathbb{R}^d$ that explain the value of $\mathbf{y}$ using $\mathbf{X}$ with the following model:

$$\mathbf{y} = \mathbf{X}\beta + \boldsymbol{\varepsilon}, \quad \textit{where} \quad \boldsymbol{\varepsilon} \sim \mathcal{N}(0, \sigma^2 \mathbf{I}),$$

where $\boldsymbol{\varepsilon}$ represents the errors due to the model.

To find the best vector $\boldsymbol{\theta}$ we have to minimize this error, *i.e.* to solve:

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^d} \sum_{i=1}^{m} \varepsilon_i = \min_{\boldsymbol{\theta} \in \mathbb{R}^d} \|\mathbf{y} - \mathbf{X}\boldsymbol{\theta}\|_2^2.$$

We easily check that is problem is convex. Indeed, the gradient of the function to optimize is:

$$\nabla_{\boldsymbol{\theta}}\, \boldsymbol{\varepsilon} = -2X^T(Y - X\boldsymbol{\theta}),$$

and the solution of Euler's Equation is given by:

and the second order derivative is given by $\boldsymbol{\theta} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$.

$$\nabla^2_\beta = 2\mathbf{X}^T\mathbf{X},$$

Which is a symmetric positive semi definite matrix of size d. Let us show it rapidly and consider that λ is an eigenvalue of X^TX and $\mathbf{u}_\lambda$ the associated eigenvector. Using the definition of eigenvalue and eigenvector, we have:

$$\begin{aligned}
\mathbf{X}^T\mathbf{X}\mathbf{u}_\lambda &= \lambda\mathbf{u}_\lambda, \\
&\downarrow \text{ pre-multipying by } \mathbf{u}_\lambda^T \\
\mathbf{u}_\lambda^T\mathbf{X}^T\mathbf{X}\mathbf{u}_\lambda &= \lambda\mathbf{u}_\lambda^T\mathbf{u}_\lambda, \\
&\downarrow \text{ right: definition of norm} \\
&\downarrow \text{ left: property transposition} \\
(\mathbf{X}\mathbf{u}_\lambda)^T\mathbf{X}\mathbf{u}_\lambda &= \lambda\left\|\mathbf{u}_\lambda\right\|_2^2, \\
&\downarrow \text{ left: definition of norm} \\
\left\|\mathbf{X}\mathbf{u}_\lambda\right\|_2^2 &= \lambda\left\|\mathbf{u}_\lambda\right\|_2^2.
\end{aligned}$$

As an eigenvector is necessarily non-zero, we have:

$$\lambda = \frac{\left\|\mathbf{X}\mathbf{u}_\lambda\right\|_2^2}{\left\|\mathbf{u}_\lambda\right\|_2^2} \geq 0.$$

An analytical solution exists to solve this minimization problem. Unfortunately this is not always the case.

Logistic Regression Let us consider now the logistic regression problem which is quite similar as the previous one.
We want to find a model that predict the class of our data (see Section 7.4 for further details).

To predict the class of the individual we use a model of the form:

$$g(\mathbf{x}, \boldsymbol{\theta}) = \log\left(\frac{\mathbb{P}(\ vx \mid y = 1)}{1 - \mathbb{P}(\mathbf{x} \mid y = 1)}\right) = \theta_0 + \theta_1 x_1 + \ldots + \theta_d x_d.$$

The parameters of the model are estimated by minimizing the negative log-likelihood of our data.

$$\ell(\mathbf{X}, \boldsymbol{\theta}) = -\sum_{i=1}^{m} y_i \log(p_i) + (1 - y_i) \log(1 - p_i), \quad \text{where} \quad p_i = \frac{1}{1 + \exp(-\sum_{j=1}^{d} \theta_d X_{ij})}.$$

Unfortunately, there is no analytical solution to this problem. We need a way to approximate it step by step. This is the purpose of gradient descent algorithm.

2.4 Gradient Descent Algorithms

Given a function f and a non empty set *spaceU* and knowing there is a solution to the problem:

$$\mathbf{u}^\star = \arg\min_{\mathbf{v} \in \mathcal{U}} f(\mathbf{v}).$$

The idea is to build a sequence $(\mathbf{u}_k)_{k \in \mathbb{N}}$ which converges towards $\mathbf{u}^\star$ using the following scheme:

- Take an initial value $\mathbf{u}_0$: where we start our optimization procedure
- $\mathbf{u}_k \rightarrow \mathbf{u}_{k+1}$: choose a direction d_k and minimize the function f along this direction
- Solve $\arg\min_{\rho > 0} f(\mathbf{u}_k - \rho \mathbf{d}_k) = \rho_k$: how far shall we go in the selected direction?
- $\mathbf{u}_{k+1} = \mathbf{u}_k - \rho_k \mathbf{d}_k$: update the value of the current position

This process is repeated until we reach the optimal solution $\mathbf{u}^\star$.

Several questions then arise (i) how to choose the direction of descent and (ii) at what point should we advance in the chosen direction?

Indeed, as shown in Figure 6 several paths seem to lead to the desired solution, but is there one that will be preferable to the other? How can we use our knowledge of the function to be optimized to build such a path rather than being guided by chance?

To answer the first question, we will focus on the following quantity at a given iteration k.

$$f(\mathbf{u}_k - \rho \mathbf{d}_k).$$

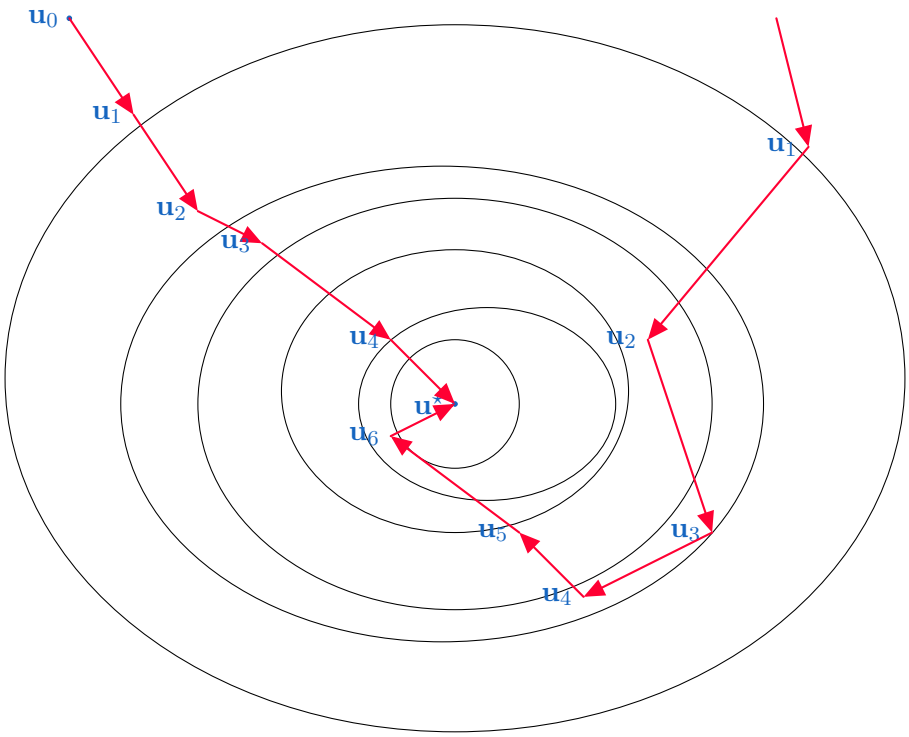


Figure 6: Illustration of different ways to reach the optimal solution $\mathbf{u}^\star$ of the function f we aim to minimize. The different ellipses represent the level set of the function f.

More precisely, we will focus on the first order approximation of this quantity. Remember that we have the following approximation when ρ is closed to 0:

$$f(\mathbf{u}_k - \rho \mathbf{d}_k) = f(\mathbf{u}_k) - \rho \langle \nabla f(\mathbf{u}_k), \mathbf{d}_k \rangle + \rho \varepsilon(\rho),$$

where ε is a function that tends to 0 when ρ tends to 0. To minimize $f(\mathbf{u}_k - \rho \mathbf{d}_k)$ we have to choose the direction $\mathbf{d}_k$ that maximize the inner product $\langle \nabla f(\mathbf{u}_k), \mathbf{d}_k \rangle$. Due to Cauchy-Scwhartz Inequality we have to choose $d_k = \nabla f(u_k)$.
So the previous algorithm become :

- Choose $\mathbf{u}_0$ to initialize the algorithm,
- set $\mathbf{u}_{k+1} = \mathbf{u}_k - \rho_k \nabla f(\mathbf{u}_k)$ for $\rho_k > 0$

However, we also need to find a stopping criterion. To find it, keep in mind that the critical values $\mathbf{u}^\star$ verify $\nabla f(\mathbf{u}^\star) = \mathbf{0}$. Thus, when the norm of a gradient of f is close to 0 we can suppose to we have rich an (or the) optimal solution. In practice, we choose a enough constant η and apply the previous algorithm until

$$\|\nabla f(\mathbf{u}_k)\| \leq \eta.$$

The remaining is how to choose the value ρ_k at each step of the gradient descent algorithm. The easiest choice is to choose a constant one, *i.e.* for all k, set $\rho_k = \rho > 0$.

It gives our first descent algorithm.

Définition 2.2: Gradient Descent with Constant Step

Let f be a function from a vectorial space E of dimension d and ρ, η strictly positives values. Then the Gradient Descent with Constant Step is defined by

- choose $\mathbf{u}_0$ to initialize the algorithm, $k = 0$
- while $\|\nabla f(\mathbf{u}_k)\| \geq \eta$
  1. compute $\nabla f(\mathbf{u}_k)$
  2. set $\mathbf{u}_{k+1} = \mathbf{u}_k - \rho \nabla f(\mathbf{u}_k)$
  3. k = k+1
- return $\mathbf{u}_k$

However, the a constant step is not the best choice to make in practice. Indeed, a bad choice of this constant can lead to a very slow convergence of our algorithm or even

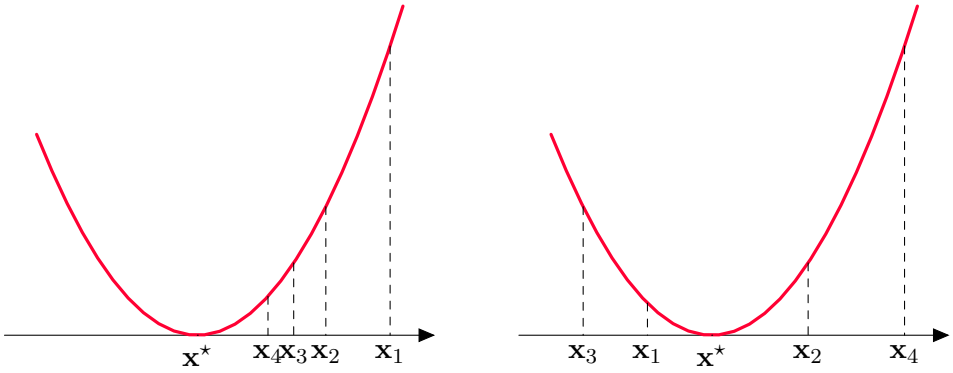


Figure 7: Importance of the choice of the learning rate ρ. A small value implies a lower speed of convergence of the algorithm (left). A too high value implies a divergence of the algorithm (right).

to its divergence as illustrated on Figure 7.

We rather choose ρ_k non constant but decreasing with the value of k, *e.g.* $\rho_k = \frac{1}{k}$. The most interesting one is to choose the value ρ_k which minimizes the function along the given direction.

It leads to another gradient descent algorithm with a so called *optimal step*.

Définition 2.3: Gradient Descent with Optimal Step

Let f be a function from a vectorial space E of dimension d and η strictly positive value. Then the Gradient Descent with Optimal Step is defined by

- choose $\mathbf{u}_0$ to initialize the algorithm, $k = 0$
- while $\|\nabla f(\mathbf{u}_k)\| \geq \eta$
  1. compute $\nabla f(\mathbf{u}_k)$
  2. solve $\arg\min_{\rho>0} f(\mathbf{u}_k - \rho \nabla f(\mathbf{u}_k))$
  3. set $\mathbf{u}_{k+1} = \mathbf{u}_k - \rho \nabla f(\mathbf{u}_k)$
  4. k = k+1
- return $\mathbf{u}_k$

We can show that, for *strongly convex functions*, this algorithm converges. We have already talk about convexity but not strong convexity, let us define it.

Définition 2.4: Strong Convexity

Let f be a convex and continuously differentiable function on $\mathbb{R}^n d$. We say that f is strongly convex or α-elliptical if it exists $\alpha > 0$ such that

$$\langle \nabla f(\mathbf{v}) - \nabla f(\mathbf{u}), \mathbf{v} - \mathbf{u} \rangle \geq \alpha \|\mathbf{v} - \mathbf{u}\|^2, \ \forall \mathbf{u}, \mathbf{v} \in \mathbb{R}^d$$

Saying differently, and using the second order derivative, a function h is said to be *α-strongly convex* if we have:

$$\nabla^2 f \succeq \alpha \mathbf{I},$$

i.e. for all $\mathbf{u} \in \mathbb{R}^d$, $\mathbf{u}^T \nabla^2 f \mathbf{u} \geq \alpha \|\mathbf{u}\|^2$.

Proposition 2.3: Optimal Step: Convergence

f f is an α-strongly convex function with respect, the Gradient Descent with optimal step converges.

Let us now have a look at the gradient at each step. What can we say about $\langle \nabla f(\mathbf{u}_{k+1}), \nabla f(\mathbf{u}_k) \rangle$ using $\rho_k = \arg\min_{\rho > 0} f(\mathbf{u}_k - \rho \mathbf{d}_k)$?

If ρ_k minimize $f(\mathbf{u}_k - \rho_k \mathbf{d}_k)$ we have :

$$\begin{aligned} & \frac{\partial}{\partial \rho} f(\mathbf{u}_k - \rho \nabla f(\mathbf{u}_k))|_{\rho = \rho_k} = 0, \\ \iff & \langle \nabla f(\mathbf{u}_k - \rho_k \nabla f(\mathbf{u}_k), \nabla f(\mathbf{u}_k) \rangle = 0, \\ \iff & \langle \nabla f(\mathbf{u}_{k+1}), \nabla f(\mathbf{u}_k) \rangle = 0. \end{aligned}$$

We can see that the two successive directions of descent are orthogonal. This interesting observation is of great interest to determine the optimal step for some problems in Machine Learning, the quadratic problems as illustrated in the following example.

Exemple 2.4. *Let $\mathbf{A}$ be a symmetric, positive and semi definite matrix and $\mathbf{b} \in \mathbb{R}^d$. Let us consider the quadratic form f for all $\mathbf{u} \in \mathbb{R}^d$ by:*

$$f(\mathbf{u}) = \frac{1}{2} \langle \mathbf{A}\mathbf{u}, \mathbf{u} \rangle - \langle \mathbf{b}, \mathbf{u} \rangle.$$

We aim to determine the optimal step, at each iteration, of gradient descent procedure applied to this function. We first compute the gradient. For all $k \in \mathbb{N}$ we have $\nabla f(\mathbf{u}_k) = \mathbf{A}\mathbf{u}_k - \mathbf{b}$. Furthermore,

$$\begin{aligned}\nabla f(\mathbf{u}_{k+1}) &= \mathbf{A}\mathbf{u}_{k+1} - \mathbf{b}, \\ &= \mathbf{A}(\mathbf{u}_k - \rho_k(\mathbf{A}\mathbf{u}_k - \mathbf{b}) - \mathbf{b}, \\ &= \mathbf{A}\mathbf{u}_k - \mathbf{b} - \rho_k\mathbf{A}(\mathbf{A}\mathbf{u}_k - \mathbf{b}).\end{aligned}$$

Now, we use the fact that for such an algorithm we have $\langle \nabla f(\mathbf{u}_{k+1}), \nabla f(\mathbf{u}_k)\rangle = 0$

$$\begin{aligned}\implies \langle \mathbf{A}\mathbf{u}_k - \mathbf{b}, \mathbf{A}\mathbf{u}_k - \mathbf{b} - \rho_k\mathbf{A}(\mathbf{A}\mathbf{u}_k - \mathbf{b})\rangle &= 0, \\ \implies \langle \mathbf{A}\mathbf{u}_k - \mathbf{b}, \mathbf{A}\mathbf{u}_k - \mathbf{b}\rangle &= \langle \mathbf{A}\mathbf{u}_k - \mathbf{b}, \rho_k\mathbf{A}(\mathbf{A}\mathbf{u}_k - \mathbf{b})\rangle, \\ \implies \rho_k &= \frac{\langle \mathbf{A}\mathbf{u}_k - \mathbf{b}, \mathbf{A}\mathbf{u}_k - \mathbf{b}\rangle}{\langle \mathbf{A}\mathbf{u}_k - \mathbf{b}, \mathbf{A}(\mathbf{A}\mathbf{u}_k - \mathbf{b})\rangle}\end{aligned}$$

Finally, for quadratic problems of the previous form, the optimal learning rate at each step k is equal to

$$\rho_k = \frac{\|\mathbf{A}\mathbf{u}_k - \mathbf{b}\|^2}{\|\mathbf{A}\mathbf{u}_k - \mathbf{b}\|^2_{\mathbf{A}}},$$

where $\|\mathbf{u}\|^2_{\mathbf{A}} = \mathbf{u}^T\mathbf{A}\mathbf{u}$.

Other algorithms or way to choose the learning rate at each step can be studied as:

- **Line search:** Armijo's criterion, Wolfe criterion
- **Algorithms:** Conjugate Gradient Descent

We will finish this part with a last algorithm called the *Newton's Method*.

The Newton's Method is also an algorithm of gradient descent. It uses the second derivative to refine the direction of the descent as follow:

$$\mathbf{u}_{k+1} \leftarrow \mathbf{u}_k - \left(\nabla^2 f(\mathbf{u}_k)\right)^{-1} \cdot \nabla f(\mathbf{u}_k).$$

- It requires less iteration to converge to the solution compared to the other gradient descent methods.
- Harder to implement, requires the inverse of the Hessian of the function we want to optimize ($\mathcal{O}(n^3)$).
- The Hessian is not always invertible at a given point.

The main drawback of the Newton's Method is to compute the inverse of the Hessian matrix $\mathbf{H}_k^{-1}$. To avoid it, an other process is proposed as follow

$$\begin{aligned} \mathbf{u}_{k+1} &= \mathbf{u}_k - \mathbf{M}_k \nabla f(\mathbf{u}_k), \\ \mathbf{M}_{k+1} &= \mathbf{M}_k + \mathbf{C}_k. \end{aligned}$$

The idea is to approximate the $\mathbf{H}_k^{-1}$ by matrix $\mathbf{M}_k$ at which, we add a matrix of correction $\mathbf{C}_k$ at each step.

Remember that :

$$\nabla f(\mathbf{u}_k) = \nabla f(\mathbf{u}_{k+1} + (\mathbf{u}_k - \mathbf{u}_{k+1})) \sim \nabla f(\mathbf{u}_{k+1}) + \nabla^2 f(\mathbf{u}_{k+1})(\mathbf{u}_k - \mathbf{u}_{k+1}),$$

we then have the following approximation:

$$\left(\nabla^2 f(\mathbf{u}_{k+1})\right)^{-1} \left(\nabla f(\mathbf{u}_{k+1}) - \nabla f(\mathbf{u}_k)\right) \sim \mathbf{u}_{k+1} - \mathbf{u}_k.$$

If we set : $\mathbf{M}_{k+1} = \left(\nabla^2 f(\mathbf{u}_{k+1})\right)^{-1}$, $\boldsymbol{\gamma}_k = \nabla f(\mathbf{u}_{k+1}) - \nabla f(\mathbf{u}_k)$ and $\boldsymbol{\delta}_k = \mathbf{u}_{k+1} - \mathbf{u}_k$, we get the Quasi Newton's Condition :

$$\boxed{\mathbf{M}_{k+1}\boldsymbol{\gamma}_k = \boldsymbol{\delta}_k}.$$

It remains to see how we can define such a correction matrix C_k. We present two solutions that are based on the use of rank 1 and rank 2 matrices and their respective algorithm.
Note that, in practice, we use the rank 2 corrections matrix in the optimization algorithm, known as BFGS.

Rank 1 Correction .

The matrix of correction $\mathbf{C}_k$ is supposed to be of rank 1. So we can rewrite $\mathbf{C}_k$ as $\mathbf{v}_k\mathbf{v}_k^T$ where $\mathbf{v}_k \in \mathbb{R}^d$.

The update rule of the matrix $\mathbf{M}_k$ can be written as

$$\mathbf{M}_{k+1} = \mathbf{M}_k + \mathbf{v}_k\mathbf{v}_k^T,$$

and the Quasi Newton's Condition gives us:

$$(\mathbf{M}_k + \mathbf{v}_k\mathbf{v}_k^T)\boldsymbol{\gamma}_k = \boldsymbol{\delta}_k,$$

$$\begin{aligned}\mathbf{M}_k\gamma_k + \mathbf{v}_k\mathbf{v}_k^T\boldsymbol{\gamma}_k &= \boldsymbol{\delta}_k,\\ \mathbf{v}_k\mathbf{v}_k^T\boldsymbol{\gamma}_k &= \boldsymbol{\delta}_k - M_k\boldsymbol{\gamma}_k,\\ \mathbf{v}_k &= \frac{\boldsymbol{\delta}_k - \mathbf{M}_k\boldsymbol{\gamma}_k}{\mathbf{v}_k^T\boldsymbol{\gamma}_k}.\end{aligned}$$

And the second line gives us : $\mathbf{v}_k^T\boldsymbol{\gamma}_k = (\boldsymbol{\gamma}_k\boldsymbol{\delta}_k - \boldsymbol{\gamma}_k\mathbf{M}_k\boldsymbol{\gamma}_k)^{1/2}$.

This first rank correction leads us to the *Broyden Algorithm.*

Définition 2.5: Broyden Algorithm

Let f be a function from a vectorial space E of dimension d and η strictly positive value. Then the Broyden Algorithm is defined by:

- choose $\mathbf{u}_0$ to initialize the algorithm, $\mathbf{M}_0 = \mathbf{I}$ and $k = 0$
- while $\|\nabla f(\mathbf{u}_k)\| \geq \eta$
  1. set $\rho_k = \arg\min_{\rho\in\mathbb{R}} f(\mathbf{u}_k - \rho\mathbf{M}_k\nabla f(\mathbf{u}_k))$,
  2. set $\mathbf{u}_{k+1} = \mathbf{u}_k - \rho_k\mathbf{M}_k\nabla f(\mathbf{u}_k)$,
  3. set $\mathbf{M}_{k+1} = \mathbf{M}_k + \frac{(\boldsymbol{\delta}_k - \mathbf{M}_k\boldsymbol{\gamma}_k)(\boldsymbol{\delta}_k - \mathbf{M}_k\boldsymbol{\gamma}_k)^T}{(\boldsymbol{\delta}_k - \mathbf{M}_k\boldsymbol{\gamma}_k)^T\boldsymbol{\gamma}_k}$,
  4. k = k+1
- return $\mathbf{u}_k$

Rank 2 Correction The idea is quite similar, but instead writing the correction matrix $\mathbf{C}_k$ as a rank one matrix, we write as the sum of two rank one matrices, which leads to rank 2 correction, *i.e.*

$$\mathbf{C}_k = \alpha\mathbf{v}_k\mathbf{v}_k^T + \beta\mathbf{w}_k\mathbf{w}_k^T.$$

Thus,

$$\mathbf{M}_{k+1} = \mathbf{M}_k + \mathbf{C}_k = \mathbf{M}_k + \alpha\mathbf{v}_k\mathbf{v}_k^T + \beta\mathbf{w}_k\mathbf{w}_k^T.$$

In order to determine good values of α and β we use the Quasi Newton's Condition and set $\mathbf{v}_k = \boldsymbol{\gamma}_k$ and $\mathbf{w}_k = \mathbf{M}_k\boldsymbol{\delta}$.

It leads to:

$$\alpha = \frac{1}{\boldsymbol{\gamma}_k^T \boldsymbol{\delta}_k} \quad \text{and} \quad \beta = -\frac{1}{\boldsymbol{\delta}_k^T \mathbf{M}_k \boldsymbol{\delta}_k}$$

and the update rule becomes:

$$\mathbf{M}_{k+1} = \mathbf{M}_k + \frac{\boldsymbol{\gamma}_k \boldsymbol{\gamma}_k^T}{\boldsymbol{\gamma}_k^T \boldsymbol{\delta}_k} - \frac{\mathbf{M}_k \boldsymbol{\delta}_k \boldsymbol{\delta}_k^T \mathbf{M}_k^T}{\boldsymbol{\delta}_k^T \mathbf{M}_k \boldsymbol{\delta}_k}.$$

The use of this update rule is known as the *Broyden–Fletcher–Goldfarb–Shanno* (BFGS) Algorithm.

Définition 2.6: BFGS Algorithm

Let f be a function from a vectorial space E of dimension d and η strictly positive value. Then the BFGS Algorithm is defined by:

- choose $\mathbf{u}_0$ to initialize the algorithm, $\mathbf{M}_0 = \mathbf{I}$ and $k = 0$
- while $\|\nabla f(\mathbf{u}_k)\| \geq \eta$
  1. set $\rho_k = \underset{\rho \in \mathbb{R}}{\arg\min}\, f(\mathbf{u}_k - \rho \mathbf{M}_k \nabla f(\mathbf{u}_k))$,
  2. set $\mathbf{u}_{k+1} = \mathbf{u}_k - \rho_k \mathbf{M}_k \nabla f(\mathbf{u}_k)$,
  3. set $\mathbf{M}_{k+1} = \mathbf{M}_k + \frac{\boldsymbol{\gamma}_k \boldsymbol{\gamma}_k^T}{\boldsymbol{\gamma}_k^T \boldsymbol{\delta}_k} - \frac{\mathbf{M}_k \boldsymbol{\delta}_k \boldsymbol{\delta}_k^T \mathbf{M}_k^T}{\boldsymbol{\delta}_k^T \mathbf{M}_k \boldsymbol{\delta}_k}$,
  4. k = k+1
- return $\mathbf{u}_k$

3 Constrained and Smooth Convex Optimization

Write this part later, maybe next year.

Part II
Supervised Machine Learning

blabla

4 Introduction

In this first part, we present some generalities on what machine learning is as well as the importance of data and the various problems related to the good representation of the latter.

4.1 Generalities

Machine Learning is a subfield of *Artificial Intelligence* at the frontier of computer science and *Applied Mathematics* (statistics and as optimization). This discipline also partially overlaps Data Science as it is based on collecting data which are analyzed and studied in order to extract the substantial information that can be used for application at hand.

The main question we want to provide an answer is:

Can the machines think?

In other words, are the machines able to do some tasks that are usually done by humans (A. Turing), such as:

- driving a care,
- create a masterpiece,
- recognize people,
- detect anomalies in medical images,
- etc.

Tom Mitchell [Mitchell, 1997] has provided a more formal definition of *Machine Learning*[2]

> A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E.

The experience E can be thought as a collection of data but also the training process that is used to train your *Algorithm*. The more data and the more you train your model, the better the results will be. However, everything depends on the used performance measure P, but we will talk about it later. The task T can refer to a

[2]The book is available here: https://www.cs.cmu.edu/ tom/mlbook.html. The first chapters might be interesting for the readers.


Figure 8: An example of data that can be used for classification tasks. These two images are extracted from the *Dogs vs Cats* dataset, available on Kaggle

classification or a regression task as it has been studied before in the linear regression course.

To sum up, to solve a given task and *to help our machine to think or to solve a problem* we need a collection of observations, *i.e.* a collection of data.

4.2 Data and Machine Learning problems

An other definition of Machine Learning can be stated as follows

> Machine Learning is a field of Computer Science, subfield of Artificial Intelligence, which aims to explore the way to elaborate and study able to learn and to make predictions using data.

Depending on the application, the nature of the data can be multiple: it can consists of images, videos, raw data, categorical data, trees, graphs, times series, etc.

A concrete example is illustrated on Figure 9 with these two images. The aim of the algorithm is to be able to distinguish the dog from the cat in a collection of images. In this case, the algorithm is trained to solve a classification problem where the algorithm output can take two values: *dog* or *cat*.

We also often deals with data that can be represented in a table. It is the case of a regression task where we aim to predict the score obtained at a language test (y) according to the scores obtained to four other tests x_1, x_2, x_3 and x_4 (see Table 1).

y	x_1	x_2	x_3	x_4
125	13	18	25	11
158	39	18	59	30
207	52	50	62	53
182	29	43	50	29
196	50	37	65	56

Table 1: Data describing the score obtained at a language test according to the score obtained to four other tests.

Although the data, by their standard representation may look very different, they are necessarily seen in the same form for a machine. This form is most often the one represented in the second example, i.e. in the form of an array of values, *i.e.* in the form of a matrix.

This data representation is one of the important pillars for machine learning. Although it may seem simple at first, having a good representation of the data is far from being trivial. This last point is even an important axis in research which is very often guided by the problem we are trying to solve. The representation learning aspect is particularly important in *Natural Language Processing* (NLP) where it is necessary to learn a good representation of words or sentences in the form of *vectors*.

Lets us go back with our images to see how we can see more precisely. Remember that images can be seen as a collection of *pixels* (or *voxels* if you working in a three dimensional space) to which a figure is associated. Generally on a black and white image, the pixel value is between 0 and 255 (or 0 and 1 if the data are *scaled*). On color images, as the one presented in Figure 9, an image is represented by three different matrices[3], where each entry represent the pixel intensity associated to a color channel (R: red, G: green and B: blue)

$$R = \begin{pmatrix} 0.1 & 1 & 0.23 \\ 0.12 & 0.23 & 0.95 \\ 0 & 0.45 & 0.22 \end{pmatrix}, \quad G = \begin{pmatrix} 0.34 & 1 & 0.19 \\ 0.67 & 0.53 & 0.95 \\ 1 & 0.95 & 0.23 \end{pmatrix}, \quad B = \begin{pmatrix} 0.56 & 83 & 0.23 \\ 0.74 & 0.81 & 0.34 \\ 0.45 & 0.45 & 0.29 \end{pmatrix}$$

However, all these features have some flaws:

- the intensity of the pixels will depend on the luminosity but also on the conditions of capture of the image

[3]In reality, the images are not left in matrix form, we use models, such as convolutional neural networks to learn a vector representation of these images, these vectors will then be easier to manipulate for future algorithms.

- the shape of the object on the image will depend on the angle of view and the distinction of the object can also be more complex
- finally, the size of the object will also take into account the distance at which it is located

For all these reasons, it is important to define/find descriptors that are independent of environmental factors, this will be the objective of a *Representation Learning* course. These methods are generally based on neural networks algorithm. We will present them later in this document.

As we can see, data is the core of Machine Learning algorithms (which could still be compared to the E experiment mentioned by Tom Mitchell). We have also seen that it can take different forms, it can be raw or transformed to make it usable by an algorithm (translation of a text or word in the form of a vector of numbers). Finally, the descriptors of these data can also be learned in order to obtain a representation of the data adequate to the studied task.

A question one might then ask when observing a series of numbers is:

Do the observed values of the different descriptors follow a **well known distribution**?

Obviously, it is true the different values follows a given distribution. However, and it is the main problem we have to deal with in Machine Learning,this distribution **is always unknown**. We will see that this fact has consequences on the learning process of an algorithm and on its performance.

Let us first focus on the use of the data and let us go back on the image classification task with *Cats* and *Dogs*. Suppose that we have a collection of such images as illustrated on Figure 9 (keep in mind that each image is actually a point in d-dimensional space, where d is the length of the vector, *i.e.* the number of descriptors of the image.) and you want to classify this image, *i.e.* learn an algorithm, so a model, which is able to predict the class of the image, *i.e.* to separate the two types of images.

To build such an algorithm, you thus need to define three things:

- **Input:** the data that will be used for the task, *e.g.* our images of cats and dogs (in a vectorial form)
- **Output:** the answer of the studied task, *e.g.* the label of the image.
- **Model:** a collection of several object that are used to solve the problem (objective function, type of hypothesis and optimization process).

In this case we could learn several types of separators or classifiers for the images, such as straight lines or more complex curves in the studied space.

Figure 9: An example of data that can be used for classification tasks. These images are extracted from the *Dogs vs Cats* dataset, available on Kaggle.

The classifier that we will learn will try to minimize the error in classification on the observed data, *i.e.* we will try to minimize the number of images of dogs that we will classify as cats and vice versa. This will be done based on the descriptors of the images as well as on the label of these images. Via a learning process that we will detail later, an algorithm will then learn iteratively from these errors in order to improve and reach an optimal solution.
Once this model is learned, it will then be submitted to new data (*e.g.* new images) that it will have to classify without knowing the labels of these data.

As we have seen, once the data are collected, and often completed and cleaned, they can be used for several machine learning tasks such as regression when it comes, e.g. to predict the price of a share or the price of a house according to its characteristics. They can also be exploited for classification tasks when, e.g. one aims to discriminate a spam from a ham when receiving an email, identify if a transaction is fraudulent or genuine, detect anomalies in a medical examination such as a blood test. In both cases, the data are labeled using a variable y. When we have a genuine transaction or a ham email, the example is usually labeled -1 (also called negative example) while it is labeled 1 (positive example) when it is an object of interest like a spam or a fraudulent transaction. When such a labeled information is used in a Machine Learning algorithm, we talk about *Supervised Learning*, and Unsupervised Learning[4] otherwise. It also exists

[4]This branch will be studied in a dedicated course

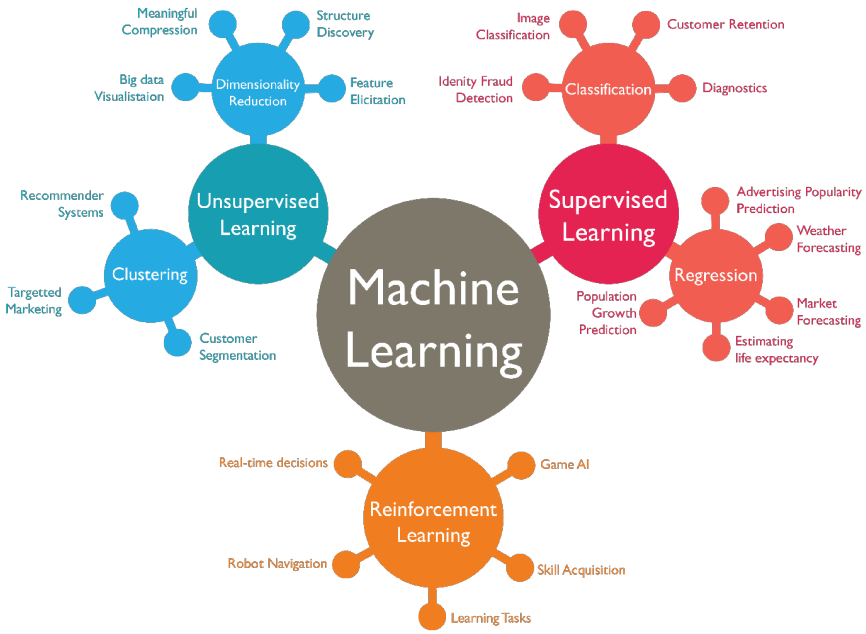


Figure 10: Descriptive diagrams of the different branches in Machine Learning as well as the different research axes in these different components. This image can be found here here.

an other branch in Machine Learning which is called *Reinforcement Learning*[5]. However, for the sake of completeness, we will give a brief description of each branch and more information can be found on Figure 10.

Supervised Learning This includes all classification and regression tasks. It includes applications such as fraud detection, recommendation systems, time series analysis, study of medical images, EEG, ECG, prediction of the number of infected people for a given disease/virus, the stock market value of a given company, etc. The important point here is the access to labeled data.
We can also classify some research fields as transfer learning (how to exploit the knowledge learned in one field in order to apply it to a related field).
There are many algorithms that belong to this field. Readers who have taken a course on linear models will surely remember the Gaussian linear model (regression) or the logistic linear regression (classification). Other models are of course commonly used and we will present them in the next section.

Unsupervised Learning Unlike supervised learning, here the data used do not have a label. The classification is then done according to the distribution of the data. We find

[5]We do not study Renforcement Learning here

applications such as anomaly detection, clustering for population segmentation for marketing strategy, recommendation systems based on customer typology. Another aspect is dimension reduction via statistical methods such as Principal Component Analysis and feature construction.
We can also include approaches such as unsupervised domain adaptation which is a subcategory of transfer learning but in its unsupervised version.

Reinforcement Learning This last category differs from the first two in that there is no real data per se. Here the system evolves in an environment and must be able to undertake the right actions according to this environment and the state in which it is via a reward/punishment system.
The simplest example is the automatic piloting of a drone whose goal is to avoid obstacles in order to reach its destination point. The environment can then be considered as the set of objects that are close to the drone. The state of the drone can be its structural state as well as its position relative to the elements around it and the actions correspond to the movement of the drone (left, right, up, down). In the case where the system hits or approaches an obstacle, we decrease the value of a quantity that reflects the reward obtained by the system.

In this course, we will focus on the former branch, *i.e.* on Supervised Learning. Our objective will be to learn a classifier h, also called an hypothesis, using a collection of labeled data $\{\mathbf{x}_i, y_i\}_{i=1}^m$ to classify new instances, where $\mathbf{x}_i$ is a set of descriptors. The next section will focus on the theoretical aspect of Supervised Learning.

Requirements In order to understand the different notion presented in this document, but also in Machine Learning in general, the reader is invited to review the fundamental concepts in *Probability/Statistics* - *Linear Models* - *Linear Algebra*: operations on matrices (determinant, transpose, multiplications, eigenvalues, eigenvectors) and finally in *Analysis*: derivation, integration, study of vector functions with real or vector values - sequences of functions and real vector sequences.

5 Statistical Learning Theory

First, we introduce the definition of Risk, the quantity we aim to minimize in most of the optimization problems when it comes We will see that it is possible to provide theoretical guarantees on the performance of the learned model using statistical tools such as concentration inequalities.

Let us consider $\mathcal{X}$ the input space of our data, also called the feature space and $\mathcal{Y}$ the output space[6]. The input space, i.e. the features used to describe the data can be of different types: it can consist of continuous or discrete descriptors. In this document, we will focus on continuous features, i.e. $\mathcal{X} \subset \mathbb{R}^d$.

The output space can also vary from a task to another. It can be discrete, real, or even a structured data. The nature of the output gives important information on the nature of the problem and leads the user to choose the appropriate tools.

- When $\mathcal{Y} \subset \mathbb{R}$ or $\mathcal{Y} = [0, 1]$, i.e. when the output is a real value, the aim is to learn either a score, a probability or to estimate a quantity as it is usually done in *regression tasks*.
- When $\mathcal{Y} = \{-1, +1\}$ or $\{0, 1\}$, i.e. the output is binary, we aim to classify the data in two categories, also called *classes*. This type of tasks is called *binary classification*. Note that the output can take more than two values, i.e. $\mathcal{Y} = \{1, \ldots, q\}$ where q is the number of classes. In this case, we talk about *multi-class classification*.

To solve these different tasks, we use an algorithm $\mathcal{A}$ which aims at learning a function $h : \mathcal{X} \to \mathcal{Y}$ called *hypothesis*. From a practical point of view, the joint distribution of the data $\mathcal{D} = \mathcal{X} \times \mathcal{Y}$ is unknown and we only have access to a collection of *sample points*, i.e. a set of observations $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, drawn i.i.d. from $\mathcal{D}$. The goal is then to learn a function h using the sample S which is "good enough" on the given training sample but which is also able to perform well on a new sample $(\mathbf{x}, y)$.

The set of observed data is usually represented in a matrix $\mathbf{X} \in \mathbb{R}^{m \times d}$ as follows:

$$\mathbf{X} = \begin{array}{c} \\ \mathbf{x}_1 \\ \vdots \\ \mathbf{x}_i \\ \vdots \\ \mathbf{x}_n \end{array} \begin{array}{c} \begin{array}{cccccc} \mathbf{v}_1 & \cdots & \mathbf{v}_k & \cdots & \mathbf{v}_p \end{array} \\ \begin{pmatrix} x_{11} & \cdots & x_{1k} & \cdots & x_{1d} \\ \vdots & & \vdots & & \vdots \\ x_{i1} & \cdots & x_{ik} & \cdots & x_{id} \\ \vdots & & \vdots & & \vdots \\ x_{n1} & \cdots & x_{nk} & \cdots & x_{nd} \end{pmatrix} \end{array},$$

[6] Note that there may be no output space, i.e. $\mathcal{Y} = \emptyset$ as in *unsupervised learning*.

where a raw represents an observation (m observations) and a a variable or descriptor is represented in column (d columns).

In the next section, we explain how the function h is learned and the meaning of performance of an algorithm.

5.1 Empirical Risk Minimization

In order to solve a given task, the algorithm $\mathcal{A}$ needs a criterion to optimize, also called a *loss function* usually denoted by $\ell(h(\cdot), \cdot) : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}_+$. In a regression task, it can be based on the difference between the estimated value, $h(\mathbf{x})$, and the true value y, for example $\ell(h(\mathbf{x}), y) = |h(\mathbf{x}) - y|$ or $(h(\mathbf{x}) - y)^2$. In a classification task, the most natural way is to consider the number of errors made by the classifier h. The considered loss function is then defined by $\ell(h(\mathbf{x}), y) = \mathbb{1}_{\{h(\mathbf{x}) \neq y\}}$ and is called *0-1* loss. However, because of its non convexity and non differentiability, minimizing this loss is known to be NP-hard. That is why, we usually change the *0-1* loss by a surrogate function.

Whatever h, we need to minimize its expected value over the distribution $\mathcal{D}$, leading to the *True Risk*.

Définition 5.1: True Risk

Let $\ell : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ a loss function and $\mathcal{D}$ the distribution of the data. The *True Risk* $\mathcal{R}$ of an hypothesis h is defined by:

$$\mathcal{R}^{\ell}(h) = \mathop{\mathbb{E}}_{(\mathbf{x},y)\sim\mathcal{D}} [\ell(h(\mathbf{x}), y)].$$

However, we can not compute this quantity in practice because the joint distribution $\mathcal{D}$ of the data is unknown. We rather minimize the *Empirical Risk*, the average error over the sample S.

Définition 5.2: Empirical Risk

Let $S = \{(\mathbf{x}_i, y)\}_{i=1}^m$ a collection of m examples drawn i.i.d. from $\mathcal{D}$ and $\ell : \mathcal{X} \times \mathcal{Y} :\to \mathbb{R}_+$ a loss function. The *Empirical Risk* risk of an hypothesis h is defined by:

$$\mathcal{R}_S^{\ell}(h) = \frac{1}{m} \sum_{i=1}^{m} \ell(h(\mathbf{x}_i), y_i).$$

In other words, the empirical risk is the mean value of the loss over all the observed data.

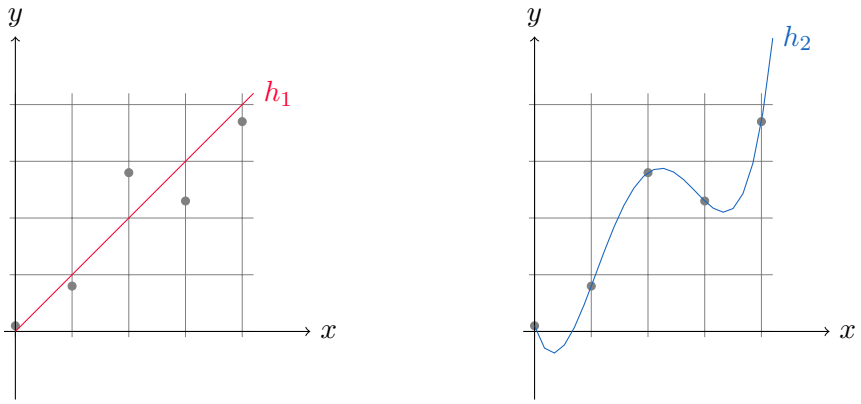


Figure 11: Illustration of the over-fitting phenomenon on a regression task. On the left, the learned hypothesis h_1 do not learn exactly the data and is able to generalize better than the hypothesis h_2 on the right which perfectly fits the data.

We aim to minimize the empirical risk with the hope that the sample S is representative of the unknown distribution. At the first glance, minimizing the empirical risk might be then equivalent to minimizing the true/generalization risk. But this is not sufficient in general, and the risk is to tend to an over-fitting phenomenon, it is illustrated on Figure 11.

In this same figure we can see that the model learned on the right seems more complex (it is curve) than the one on the right which is straight line. This remark shows that it is important to fix some constraints on the hypothesis we can learn to avoid such a situation. We present below a non exhaustive list of ways to constrain the learned hypothesis.

Empirical risk minimization. The most straightforward solution is to fix in advance the hypothesis space $\mathcal{H}$, called a *Set of Hypotheses* or *Class Function*. Then, given this set of hypotheses $\mathcal{H}$, a sample S and a loss function ℓ, we are looking for the hypothesis $h_S^\star$ as the solution of:

$$h_S^\star = \underset{h \in \mathcal{H}}{\operatorname{argmin}} \, \mathcal{R}_S^\ell(h).$$

If the size of $\mathcal{H}$ is small enough, we can avoid the situation previously described. However, without any side information on the data, it is hard to choose the appropriate set of hypotheses for the task at hand.

Structural risk minimization. [*See Section 4.1 of [Vapnik, 1995]*] The idea here is to choose a growing sequence of sets of hypotheses $\{\mathcal{H}_n\}_{n \in \mathbb{N}}$, i.e. the complexity or the

number of hypotheses is growing with n, and to choose the hypothesis $h^\star$ solution of the following minimization problem:

$$h_S^\star = \underset{h \in \mathcal{H}_n,\, n \in \mathbb{N}}{\operatorname{argmin}} \; \mathcal{R}_S^\ell(h) + pen(n, d).$$

Compared to the previous formulation, a *penalty term* is added to the empirical risk. This term measures the complexity/size of the set of hypotheses $\mathcal{H}_n$ and is a growing function of n (note that it also depends on the dimension of the data). The aim is then to learn the best hypothesis in the smallest set $\mathcal{H}_n$. With such a formulation, we are able to find a good trade-off between the empirical risk minimization and the complexity of the learned model, i.e. find a hypothesis that is able to generalize well. But choosing a sequence of sets of hypotheses remains hard in practice.

Regularized risk minimization. The preferred solution in practice, because easier to implement, is to choose a set of hypotheses $\mathcal{H}$ sufficiently large enough and to add a constraint on the parameters θ of the learned model. The added constraint, $\lambda \|\boldsymbol{\theta}\|$, is called a *regularization term* and is associated to a *regularization constant* λ.

$$h_S^\star = \underset{h \in \mathcal{H}}{\operatorname{argmin}} \; \mathcal{R}_S^\ell(h) + \lambda \|\boldsymbol{\theta}\| .$$

With the parameter λ, we are able to indirectly control the size or complexity of the set of hypotheses by constraining the norm of the parameters of the model for instance[7].

Exemple 5.1. *Such a formulation has been met in the Linear Model course when the lasso regression was presented. In this problem, we aim to find the best linear model, described by a vector $\boldsymbol{\theta}$, which fits a set of data $(\mathbf{y}, \mathbf{X})$. We also ask that the chosen model to be as sparse as possible, i.e. based on the minimum number of features. To do so we solve the following minimization problem:*

$$\min_{\boldsymbol{\beta} \in \mathbb{R}^{d+1}} \; \|\mathbf{y} - \mathbf{X}\boldsymbol{\theta}\|_2^2 + \lambda \|\boldsymbol{\theta}\|_1,$$

where the use of $\|\cdot\|_1$ is the lasso regularization term which induces a sparse solution for a sufficiently large λ.

In this last expression (as in the previous example), the smaller the value of λ is, the more importance we give on fitting well the data. In the same way, the greater λ, the more importance we give on the complexity of the learned model. The errors made by the algorithm become insignificant compared to the control of the magnitude of the parameters θ. The risk is that, when $\lambda \to \infty$ we lead to an *under-fitting* phenomenon without a good generalization capacity.

[7] In the rest of the document, such a regularization constant will be called *hyper-parameter* instead of parameter to distinguish them from the parameters of the learned hypothesis h.

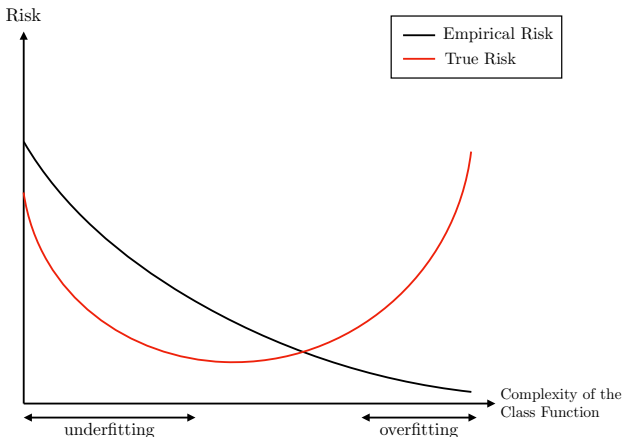


Figure 12: Evolution of the Empirical and True Risks according to the complexity of the set of hypotheses.

The aim is to learn a model with the appropriate value of λ, i.e. to find a good trade-off between the minimization of the empirical risk and the complexity of the hypothesis as depicted in Figure 12. To *tune* the value of λ, we typically use a *k-fold cross-validation* on the training set: we separate the dataset into $k-$folds of regular size. This process will be described later, but to present it rapidly, it consists in learning a hypothesis/model using $k-1$ folds of the training data and test its performance on the remaining one. The process is repeated $k-$times and we keep the hypothesis/model, so the value of λ, with which achieves the lowest empirical risk in average.

Even if we can empirically assess the performance of a model on a test set, it is also important to provide *generalization guarantees* in order to bound the performance of the learned algorithm regarding its behavior at training time.

The next section is dedicated to this point and presents generalization bounds based on the complexity of the set of hypotheses or on some properties on the loss function.

5.2 Generalization Bounds

A generalization bound has the following general form and is often referred to a *Probably Approximately Correct* (PAC) bound [Valiant, 1984]:

$$Pr\left(|\mathcal{R}(\cdot) - \mathcal{R}_S(\cdot)| \geq \varepsilon\right) \leq \delta, \tag{1}$$

where $\varepsilon \geq 0$ and $\delta \in [0, 1]$. δ is an upper bound on the probability that the true risk deviates from at least ε from its empirical value. Looking at this bound, an immediate consequence is that the lower the value of ε, the greater (i.e. the closer to 1) the value

of δ. Furthermore, the closer from 0 both δ and ε are, the more reliable our estimation is. Such a bound is usually derived using concentration inequalities such as Hoeffding [Hoeffding, 1963] or McDiarmid [McDiarmid, 1989] inequalities. The bound (1) can be rewritten as a probabilistic *bound of convergence* from the empirical estimate to its mean [Vapnik and Chervonenkis, 1982, Valiant, 1984]

$$\left|\mathcal{R}^{\ell}(\cdot) - \mathcal{R}_S^{\ell}(\cdot)\right| \leq \varepsilon(\delta, m),$$

where δ is the rate of confidence on the given bound and ε is a function of the rate of confidence and a decreasing function on the number of training examples. Such a bound holds with probability at least $1-\delta$ and the aim is to build a function $\varepsilon(\cdot)$ with a high rate of convergence.

In the following, we will see how we can build such generalization bounds and that their convergence rate is often $\sim \mathcal{O}(\ln(m)/\sqrt{m})$ or even $\mathcal{O}(1/\sqrt{m})$.

For a complete construction of the presented bound, the reader can refer to the references cited in the different part and also to the excellent book *Foundations of Machine Learning* [Mohri et al., 2012].

Uniform Deviation

As stated before, generalization bounds are based on the convergence of empirical quantities to their means [Vapnik and Chervonenkis, 1971] and resort on the law of large numbers (see. Intuitively, the larger our training set is, the closer the empirical risk will be to the expected value, i.e.:

$$\lim_{m\to\infty} \frac{1}{m}\sum_{i=1}^{m} \ell(y_i, h(\mathbf{x}_i)) = \mathop{\mathbb{E}}_{(y,\mathbf{x})\sim\mathcal{D}}[\ell(y, h(\mathbf{x}))].$$

The bound is said to be *uniform* because it holds for any hypothesis $h \in \mathcal{H}$.
To get such a bound when the size of $\mathcal{H}$ is finite, we need to estimate the probability of the following event:

$$\left\{\sup_{h\in\mathcal{H}} \left|\mathcal{R}^{\ell}(h) - \mathcal{R}_S^{\ell}(h)\right| \geq \varepsilon\right\}.$$

The probability of such an event can be estimated using the Hoeffding inequality and the *Union bound*. It leads to the following result [Bousquet et al., 2004].

Théorème 5.1: Uniform Generalization Bound

Let $\mathcal{H}$ be a set of hypotheses of finite size, S a training sample of size m drawn i.i.d. from $\mathcal{D}$, ℓ a loss function which takes its values in $[0, 1]$ and $\delta > 0$. Then, for any $h \in \mathcal{H}$, with probability at least $1 - \delta$, we have:

$$\mathcal{R}^{\ell}(h) \leq \mathcal{R}^{\ell}_{S}(h) + \sqrt{\frac{\ln |\mathcal{H}| + \ln(2/\delta)}{2m}}.$$

When the space $\mathcal{H}$ is not finite, e.g. in the case of a set of linear classifiers in $\mathbb{R}^d$ where the parameters which define a linear separator belong to $\mathbb{R}^{d+1}$, we need another method to measure the size or complexity of the set of hypotheses. A way to measure the complexity is the VC-dimension, noted $VC(\mathcal{H})$ and introduced by Vapnik and Chervonenkis in the 70s [Vapnik and Chervonenkis, 1971].
Another way to measure the complexity of a set oh hypotheses is the Rademacher complexity [Bartlett and Mendelson, 2003, Koltchinskii and Panchenko, 2000]. Informally, it measures how the set of hypotheses is able to fit noise in the dataset.
However, these two measures are usually hard to estimate for most of the problems (except for linear ones).

Fortunately, it exists an other to establish generalization bounds for convex optimization problems using the *Uniform Stability* framework.

Uniform Stability

This framework is more recent [Bousquet and Elisseeff, 2002] and is not directly based on a measure of complexity of the set of hypotheses. It resorts on the concept of *stability*. Roughly speaking, an algorithm is *stable* if its output does not change significantly under a small modification of the training sample. More precisely, we focus on *uniform stability*: we are looking for the greatest modification in the loss function under a small modification of the training sample. A more formal definition is given below.

Définition 5.3: Uniform Stability

A learning algorithm $\mathcal{A}$ has a uniform stability in $\frac{\beta}{m}$ with respect to a loss function ℓ and parameter set θ, with β a positive constant if:

$$\forall S,\ \forall i,\ 1 \leq i \leq m,\ \sup_{\mathbf{x}} |\ell(\theta_S, \mathbf{x}) - \ell(\theta_{S^i}, \mathbf{x})| \leq \frac{\beta}{m},$$

where S is a learning sample of size m, θ_S the model parameters learned from S, θ_{S^i} the model parameters learned from the sample S^i obtained by replacing the i^{th} example $\mathbf{x}_i$ from S by another example $\mathbf{x}'_i$ independent from S and drawn from $\mathcal{D}$. $\ell(\theta_S, \mathbf{x})$ is the loss suffered at $\mathbf{x}$ [a] .

[a] Note that the authors also provide a definition of stability in which an example is removed. For the sake of convenience, we rather use this definition throughout this document. Indeed, we keep the same number of training instances from a set to another.

The constant β depends on the properties of the loss function but also on the regularization term of the minimization problem. The property of uniform stability has been shown to hold for a wide range of minimization problems [Bousquet and Elisseeff, 2002]. Using the *convexity* of the loss function and the Mc Diarmid inequality we get the following generalization bound with a rate of convergence in $\mathcal{O}(1/\sqrt{m})$.

Théorème 5.2: Uniform Stability based Bound

Let $\delta > 0$ and $m > 1$. For any algorithm with uniform stability β/m, using a loss function ℓ bounded by K, with probability at least $1 - \delta$ over the random draw of S we have:

$$\mathcal{R}^{\ell}(\theta_S) \leq \mathcal{R}^{\ell}_S(\theta_S) + \frac{2\beta}{m} + (4\beta + K)\sqrt{\frac{\ln 1/\delta}{2m}},$$

where $\mathcal{R}(\cdot)$ is the true risk and $\mathcal{R}_S(\cdot)$ its empirical estimate over S.

The previous bound has a global convergence rate in $\mathcal{O}(1/\sqrt{m})$ and depends on two parameters: β the constant of uniform stability of the algorithm $\mathcal{A}$ and K an upper bound on the loss function ℓ.

This generalization bound is more informative in practice and both terms are easy to compute. Moreover, compared to the two previous frameworks, it is consistent with the practice in Machine Learning, i.e. it takes into account the regularization term which is often used in the minimization problems. Furthermore, the constant of uniform stability can be show, to be a decreasing function of the hyper-parameter λ (associated to the regularization term). The impact of the hyper-parameter value is depicted on Figure 13 and shows that is important to find the right value in order to find the best solution.

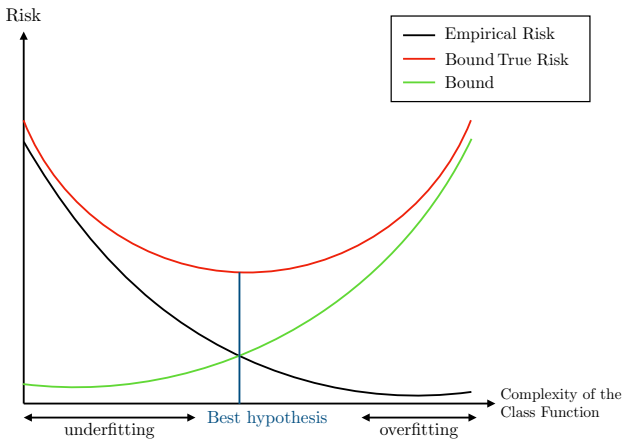


Figure 13: Evolution of the Empirical and True Risks and the the value of the bound according to the complexity of the set of hypotheses, *i.e.* the value of the hyper-parameter of the model.

We will see that is easy to build such bounds on convex regularized optimization problem for both classification and regression tasks. It only requires some tools in linear algebra and the definition of convex function.

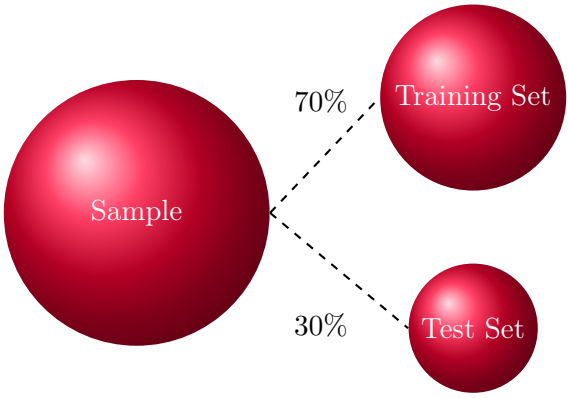


Figure 14: Illustration with some figures of how to split the sample to train-test sets

6 Learning Procedure

Before presenting different algorithms used in Supervised, it is important to describe the way we will learn a model. The aim of this section is to describe the good way to study an algorithm and to tune your hyper-parameters when one or several are present in the optimization problem.

We also use this section to make some reminders about convex optimization and in particular about gradient descent which can be useful in most learning algorithms.

6.1 How to train and tune a Model

Let's imagine that we want to solve a certain task in practice, such as a classification task, and that for this we have a data set labeled $(y_i, x_i)_{i=1}^m$. Our goal is then to learn a certain hypothesis h from our data that is able to solve this task in an optimal way with the data we have. We also want our model to be able to generalize optimally on new data that it has not yet encountered. This last point is important because it allows us to develop algorithms that can be put into production to solve certain tasks automatically.

The first question that arises is:

How to measure the efficiency of our model on data that it has not yet encountered, starting from the data I have?

This is the first step of an experimental protocol in Machine Learning. In order to evaluate the performance of the model learned from our dataset, we will start by separating our dataset into two sets. The first set will be a training set and the second set will be called the test set (see Figure 14).

Training set This set is used to learn the parameters of the model, *e.g.* the parameter of a linear seperator or the parameters of a linear model. The algorithm has access to the features $\mathbf{x}_i$ **and** the label y_i of the data in order to be trained.

Test set This set is used to evaluate the model once it has been learned. The prediction is only based on the descriptors of the data $\mathbf{x}_i$ and the label is used a posteriori only to check if the prediction made by the model is correct or nor, *i.e.* to evaluate the performance of the model on new data, *i.e.* data that has not been encountered before.

Usually, we keep approximately 2/3 of the examples to train the model and the remaining 1/3 to test the model. It is important to keep a large amount of data to train the model, so that we are able to capture all the specificities of the data. Indeed, the more the data you have, the more they will describe the underlying distribution. However, it is also important to keep enough example in the test set so that the observed results are statistically significant.

But what about the hyper-paremeter tuning ?

The previous presented methodology to evaluate the model does not solve the problem of hyper-parameter tuning. The first thing we can do is to mimic the previous procedure but on the *training set* ! So we can have access to a sort of test set, called *validation set*, that can be used to evaluate the performance of the learned model with each values of the hyper-parameter. More concretely, let us imagine that we aim to learn an hypothesis h which depends on a parameter $\boldsymbol{\theta}$ and on a hyper-parameter λ. To tune the value of λ we first select a range of possible values for λ, we note them λ_i and the process works as follows:

1. Split the dataset into training set - test set
2. Split the training set into learning set - validation set
3. For each value of λ_i:
   (a) train your hypothesis (*i.e.* learn the parameter $\boldsymbol{\theta}$) using λ_i using sur learning set
   (b) test the performance of learned model on the validation set. It gives you a performance p_i.
4. Keep the hyper-parameter λ_i associated the highest value p_i, noted $i_{\max}$
5. Learn a model with *all the training set* using the hyper-paramter value $\lambda_{i_{\max}}$
6. Test this model on the test set

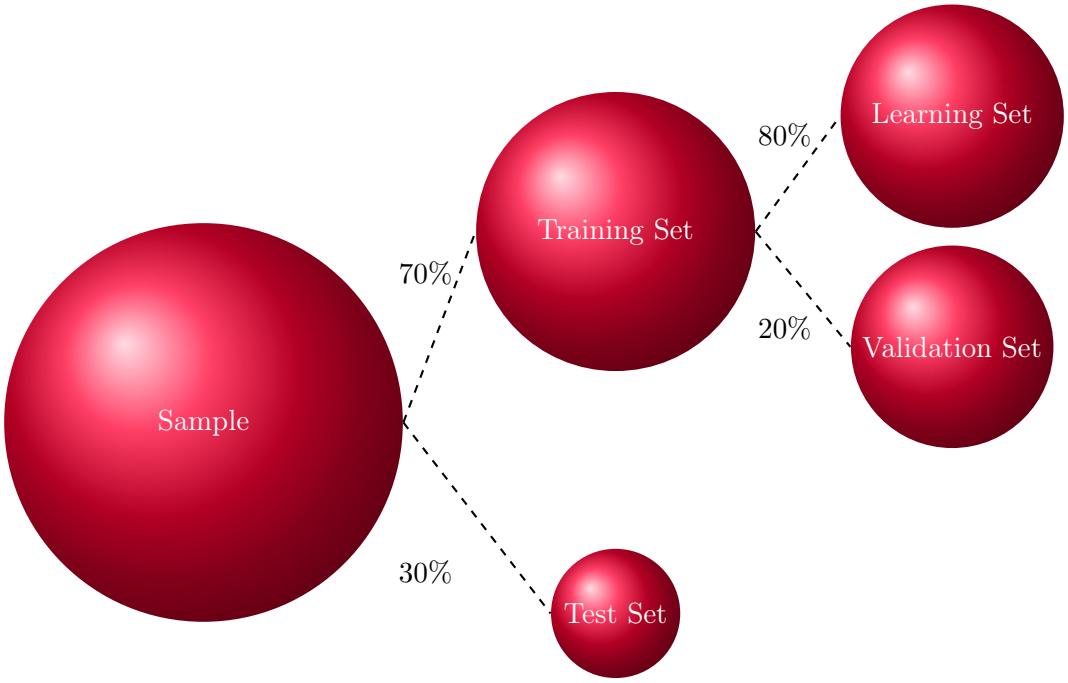


Figure 15: Illustration with some figures of how to split the sample to train-test sets

The splitting process is illustrated on Figure 15. Note that once the best hyper-parameter value is found, the model is train again using a larger amount of data. By doing so, we hope that we will be able to improve the performance. Furthermore, by doing so, we will be able to detect if the model suffer from *over-fitting*, *i.e.* the validation performance will be lower than then one observed on the training set, or *under-fitting*, *i.e.* the validation performance will be greater than then one observed on the training set.

However, it is possible to improve this tuning technique. Indeed, this method has a small flaw, the validation is based only on a very small subset of the data, so potentially on a very small part of their distribution.

In the previous section, we discussed cross-validation as a way to tune hyper-parameters. **What does this consist of?** Let us illustrate it using Figure 16.

In this case, the training is no more seperated into two groups, we will instead separate into k groups or k folds. The tuning process is then the following:

1. For each value λ_i of the hyper-parameter
   (a) at the first round, you select the first fold as the validation set and the $k-1$ others as the learning set. This allows us to obtain a first measure of performance p_1.

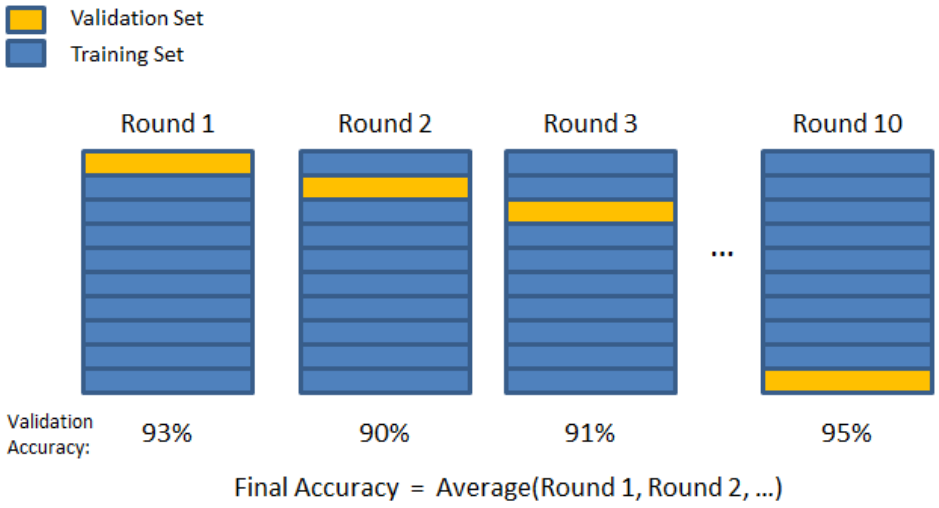


Figure 16: Illustration of the cross-validation process using a 10 fold cross-validation. The training set is divided in 10 folds or groups. The model is learned using 9 folds and is validated on the remaining one. At each round, we change the validation in order to browse the whole dataset. Thus the final performace, here the accuracy, is computed as the average of the 10 values measured on the validation fold at each round.

   (b) at the second round you select the second fold as the validation set and the $k-1$ others as the learning set. This allows us to obtain a second measure of performance p_2.

   (c) ...

   (d) at round k you select the fold k as the validation set and the $k-1$ others as the learning set. This allows us to obtain a last measure of performance p_k

   (e) compute $s_i = \frac{1}{k}\sum_{l=1}^{k} p_l$, *i.e.* the average score obtained at each round.

2. Keep the hyper-parameter λ_i associated the highest value s_i, noted $i_{\max}$
3. Learn a model with *all the training set* using the hyper-paramter value $\lambda_{i_{\max}}$
4. Test this model on the test set

This tuning method allows us to run the whole data set and thus a larger part of the underlying distribution. On a practical level, it will therefore bring better results. However, it requires to learn as many models as we have groups (*i.e.* k) and as we have values of hyper-parameters to test.

In what has just been presented there is mention of performance measures without ever having explained what they are. In Section 5.1, we have already hinted at a potential performance measure, called accuracy, when we talked about the error rate of our algorithms. The next section is devoted to the presentation of classical performance measures in classification and regression.

6.2 Performance Measures

The nature of the used performance measure depends on the nature of the problem, *i.e.* classification or regression tasks.

As in the previous section, we will denote $\mathbf{x}$ as the feature vector and y as the label or the value the real value to predict. We will also denote $\hat{y} = h(\mathbf{x})$ as the output of an hypothesis h.

6.2.1 Regression tasks

In regression tasks, where the output space $\mathcal{Y} \subset \mathbb{R}$, we already have met several performance measure.

The most used is the *Mean Square Error* (MSE) which consists in computing the mean value, over all examples, of the squared difference between the predicted value $\hat{y}$ and the true value y:

$$MSE_h = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2.$$

If we skip the constant $1/m$, the MSE can be seen as the ℓ_2 norm of the vector $\mathbf{y} - \hat{\mathbf{y}}$.
This measure has the main advantage to be convex and differentiable and can thus be directly optimized. It is not uncommon to consider also the *Root Mean Square Error* (RMSE) when the squared deviations become very large so as not to create a numerical problem :

$$RMSE_h = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2}.$$

It is certainly the most used performance measure in regression. However, it has an important flaw, it is very sensitive to outliers, i.e. when a deviated data is far from the regression line, its value becomes very quickly important. This also leads to an important bias in the models that will try to get closer to this outlier (see Figure That is why we often use an other performance measure which is *Mean Absolute Error* (MAE), which is

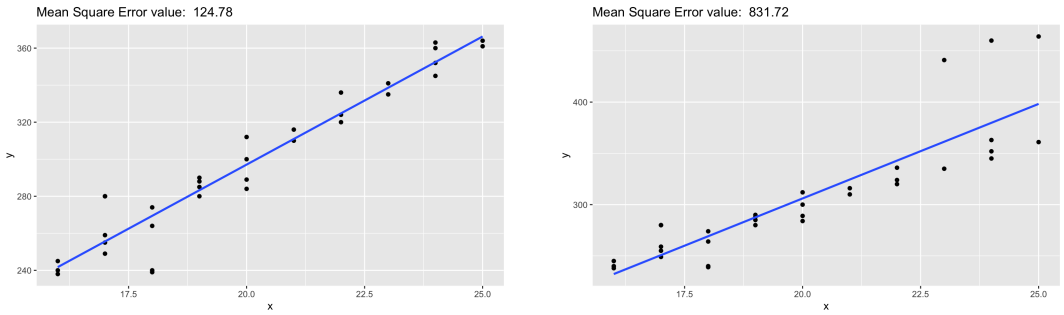


Figure 17: Caption

the mean value, over all examples, of the absolute difference between the predicted value $\hat{y}$ and the true value y:

$$MAE_h = \frac{1}{m}\sum_{i=1}^{m}|y_i - \hat{y}_i|.$$

If we skip the constant $1/m$, the MSE can be seen as the ℓ_1 norm of the vector $\mathbf{y}-\hat{\mathbf{y}}$.

Other performance measures have also been met to measure the *quality* of a regression model : AIC and BIC criterion or R squared value.

6.2.2 Classification tasks

In this section, we present different metrics or performance measures that are suited in a binary context. However, they can easily be extended to the multi-class setting (for some of them) as it will be shown.

We will consider a binary classification problem with two classes: $y = -1$ and $y = 1$. The class $y = 1$ will be called the positive class and the class $y = -1$ is the negative one[8]. With this setting, we can compute several performance measures using a *Confusion Matrix* an example is given in Table 2. Using this table, an example from the positive class predicted positive by the learned model is said to be a *True Positive (TP)*. If it is miss-classified it is called a *False Negative (FN)*. Similarly, for the majority class, a well classified example is called a *True Negative (TN)* and a *False Positive (FP)* otherwise.

[8]We will sometime use the class 0 to denote the negative class for the sake of simplicity to present some of the algorithms.

	Predicted positive ($h(\mathbf{x}) = 1$)	Predicted negative ($h(\mathbf{x}) = 0$)
Actual positive ($y = 1$)	True Positive (TP)	False Negative (FN)
Actual negative ($y = 0$)	False Positive (TP)	True Negative (TN)

Table 2: Confusion matrix for a binary classification task. The label is denoted by y and the prediction by $h(\mathbf{x})$.

Standard Measures The most common performance metric used to evaluate an algorithm is the *Accuracy*, also known as the complementary of the *Error rate*. Using the notations of Table 2, they are defined as:

Définition 6.1: Accuracy - Error rate

Let us consider a set of m examples and an hypothesis h which leads to a given a confusion matrix using the examples. The Accuracy of h is then defined by:

$$\text{Accuracy} = \frac{TP + TN}{m}$$

and the Error rate is defined by :

$$\text{Error rate} = 1 - \text{Accuracy} \frac{FP + FN}{m}$$

While this measure is the most used to evaluate the performance of the algorithms, it is not relevant it all settings, *e.g.* when the dataset is said imbalanced, *i.e.* when a given class is less represented than the other.

Exemple 6.1. *Let us consider a dataset with $m = 100,000$ examples. Suppose that the number of positive examples is equal to $m_+ = 1,000$ and the number of negative examples is equal to $m_- = 99,000$. Then a trivial solution, which consists in predicting all the examples as negative, would lead to an accuracy of 99%. It is good at a first glance, but the solution is not satisfactory because the learned hypothesis was not able to find examples of the positive class.*

To overcome the issue of the above example, we need to use another performance measure which takes into account the capacity of the predicted model h to capture the minority class :

Définition 6.2: Sensivity or Recall

Let us consider a set of m examples and an hypothesis h which leads to a given a confusion matrix using the examples. The *Sensivity*, also known as *Recall*, of a classifier h is defined as:

$$\text{Recall} = \frac{TP}{TP + FN}.$$

By definition, the Recall returns the percentage of examples in the class of interest the model is able to capture. The higher this value (i.e. the closer to 1), the more examples in the minority class are retrieved. By measuring how a model is able to detect the examples that belong to the minority class, the Recall seems to be a good candidate for a performance measure to use in imbalanced scenarios, *e.g.* in the context of medical diagnosis, missing a possible patient with a serious disease can be fatal.

In some banking it is important to be confident on its positive prediction, *e.g.* when an individual is denied a more or less fundamental right, the decision had better be the right one, otherwise there could be serious consequences for the company or organization that made the decision. To this end, we prefer the model to be confident on its positive predictions, *i.e.* we focus on the Precision of the model.

Définition 6.3: Precision

Let us consider a set of m examples and an hypothesis h which leads to a given a confusion matrix using the examples. The *Precision*, also called *Positive Predictive Value*, of a hypothesis h is defined as:

$$\text{Precision} = \frac{TP}{TP + FP}.$$

Other performance measures [Konukoglu and Ganz, 2014, Branco et al., 2016] based on the classification made by h are given below:

$$\text{True Negative Rate} = \frac{TN}{TN + FP}, \quad \text{False Negative Rate} = \frac{FN}{FN + TP},$$
$$\text{False Positive Rate} = \frac{FP}{FP + TN}, \quad \text{Negative Predictive Value} = \frac{TN}{TN + FN}.$$

Such measures can be seen as complementary to the Precision and Recall. Note that the True Negative Rate is also known as *Specificity*. As shown before, the metric used depends on the user preference [Torgo and Ribeiro, 2007, Torgo and Lopes, 2011].

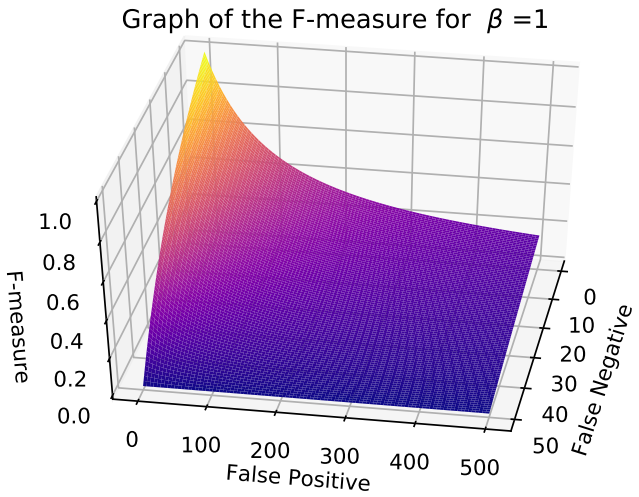


Figure 18: Illustration of the F-measure with 50 positive instances and 500 negative instances. The axes represent respectively: FN, FP and the value of the F-measure.

In imbalanced settings, as in fraud detection, it is important to take into account both the precision and the recall of the model of the model This is the goal of the F-measure introduced in the seventies [Rijsbergen, 1979].

A Versatile Measure : the F-measure The F-measure is a good performance measure when a user wants to focus on the behavior of a model on a minority class. It is highly used in information retrieval [Sanderson, 1994] and obviously in Fraud and Anomaly Detection [Gee, 2014, Bahnsen et al., 2014, Bolton and Hand, 2002]. It is defined as the harmonic mean of the Precision and the Recall and depends on a parameter β:

$$F_\beta = \frac{(1+\beta^2)\text{Precision} \times \text{Recall}}{\beta^2 \times \text{Precision} + \text{Recall}} = \frac{(1+\beta^2)TP}{(1+\beta^2)TP + \beta^2 FN + FP} = \frac{(1+\beta^2)(P-FN)}{(1+\beta^2)P - 2FN + FP}.$$

An illustration of the F-measure is given in Figure 18. The F-measure is a flexible measure. By modifying the β value, the user is able to control the importance of either the Precision or the Recall. If $\beta = 1$ the same weight is given to both Precision and Recall. If the user wants to give more importance to the recall, he can choose a value of β greater than one. By choosing $\beta < 1$, he will give more importance to the precision.

Exemple 6.2. *Let us take an example of a sample of* 1000 *instances in which the minority class represents* 1% *of the data. So we have 10 positive examples for* 990 *negative ones. Let us also consider two hypothesis,* h_1 *and* h_2*, which have the following confusion matrices:*

h_1		*Predicted positive*	*Predicted negative*
	Actual positive	$TP = 3$	$FN = 7$
	Actual negative	$FP = 0$	$TN = 990$

h_2		*Predicted positive*	*Predicted negative*
	Actual positive	$TP = 9$	$FN = 1$
	Actual negative	$FP = 6$	$TN = 984$

Both of these classifiers have an error rate less than 1%. *The first classifier,* h_1 *has a Precision equal to* 1 *and a Recall equal to* 0.3 *while for the second classifier* h_2*, these two values are respectively equal to* 0.6 *and* 0.9. *These two classifiers lead to completely different F-measures (with* $\beta = 1$*):* 0.46 *for the first hypothesis and* 0.72 *for the second one. Despite the high precision reached by the first classifier (equal to* 1*), the F-measure remains lower than the one achieved by the hypothesis* h_2.

Compared to the Accuracy and despite how useful such a measure can be in imbalanced scenarios, it remains hard to optimize. The F-measure presents two main drawbacks: (i) it is non-linear and (ii) non convex, as depicted on Figure 18. Because of (i), it is hard to derive generalization guarantees for such a measure. Furthermore, we can not use standard gradient descent algorithms to optimize it. Because of (ii), an optimization algorithm can fall into local optima that might be far from the optimal solution. However, the literature is rich of studies and algorithms which aim to deal with such a task [Zhao et al., 2013, Busa-Fekete et al., 2015].

6.2.3 Other Performance Measures

The *Class Weighted Accuracy*, noted CWA and proposed by [Cohen et al., 2006] is similar to the F-measure. However, it does not take into account the Precision anymore but the Specificity instead, which is directly linked to the Precision of the model. It is expressed as a convex combination of both Sensitivity (i.e. the Recall) and Specificity , i.e. for any $\alpha \in [0, 1]$ it can be expressed as:

$$CWA = \alpha \times \text{Sensitivity} + (1 - \alpha) \times \text{Specificity}.$$

As for the β parameter of the F-measure, the user can also choose the value of the parameter α.

Another evaluation metric used in imbalanced scenarios is the G-mean measure. As for the Class Weighted Accuracy, its expression depends on both Sensitivity and Specificity. It is the square root of the product of these two quantities:

$$\text{G-mean} = \sqrt{\text{Specificity} \times \text{Sensitivity}}.$$

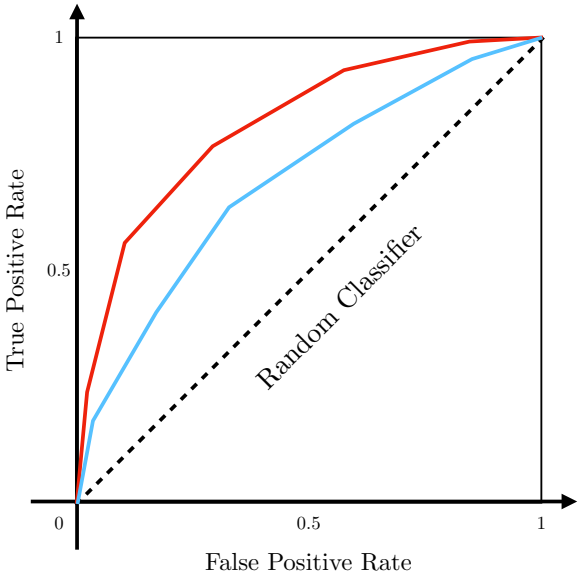


Figure 19: ROC and AU ROC for two different classifiers. The classifier associated to the red curve achieves a larger AUC than the blue one.

This measure is used in imbalanced scenarios because it takes into account the information on both classes but gives the same importance to each class.

A last popular evaluation metric used in imbalanced scenarios is the AUC ROC [Metz, 1978, Cortes and Mohri, 2004], the "Area Under the Receiver Operating Curve". This metric is used when the learned model returns a score, such as a confidence in a given prediction. The training examples can be ranked according to this score and a curve which plots the Recall according to the False Positive Rate is drawn using the different scores obtained by the model (see Figure 19). The closer to 1 the value of the AUC is, the better the model is.

Compared to the previous measures, the AU ROC allows us to can visualize the performance of the model. Furthermore, it is usually used to choose the appropriate threshold for the given task. It also gives the possibility to plot several models on a single graph and choose the one best suited for a given purpose. The ideal model is the one that achieves a True Positive Rate equal to 1 while the False Positive Rate remains close to 0, i.e. when all the examples of the minority class have a greater score than the examples of the majority class.

There exist other measures used in imbalanced scenarios, such as the Average Precision, the H-measure or the Precision-Recall curve to cite a few [Frery et al., 2017,

Ferri et al., 2009, Jeni et al., 2013, Branco et al., 2016, Behl et al., 2014]. Despite most of them were suited for imbalanced tasks, they can also be used to evaluate performances in standards contexts.

Now that we have explained how to learn, tune and evaluate a model in general, it is time for us to present some fundamentals algorithms in supervised Machine Learning.

7 Supervised Algorithmes

7.1 Surrogate losses

In Section 4, the generalization bounds based on uniform deviation has been presented for a loss ℓ function taking its values in $[0,1]$, typically, the 0-1 loss, in the context of classification. However, as said before, such loss is hard to optimize from an algorithmic point of view due to its non-convexity and non-differentiability. So, we rather use some convex (and sometimes differentiable) relaxation, also called *surrogate losses*, to learn our hypothesis $h(\cdot,\cdot)$, where the two variables are respectively the parameters $\mathbf{w}$ of the model and an input $\mathbf{x}$. Both the 0-1 loss and some of its surrogates are presented in Figure 20.

Among these surrogates, the *hinge loss* is used when training a Support Vector Machine [Boser et al., 1992, Vapnik and Cortes, 1995]. The hinge loss is used in the context of classification and is defined by:

$$\begin{aligned}\ell(h(\mathbf{x}),y) &= [1-yh(\mathbf{w},\mathbf{x})]_+ ,\\ &= \max(0,1-yh(\mathbf{w},\mathbf{x})).\end{aligned}$$

Another well known surrogate loss function is the exponential loss which is widely used in the context of boosting [Friedman et al., 2000]:

$$\ell(h(\mathbf{x}),y) = \exp(-yh(\mathbf{x})).$$

Compared to the hinge loss, the exponential loss gives a more important weight on the errors and never equals 0.

Finally the logistic loss, used when training a logistic regression model [Mohri et al., 2012], is defined by:

$$\ell(h(\mathbf{x}),y) = \frac{1}{\ln(2)}\ln\left(1+\exp(-yh(\mathbf{x}))\right).$$

Note that this definition holds when $y\in\{-1,1\}$ and $h(\mathbf{x})$ belongs in $\mathbb{R}$ or $[-1,1]$. There is another definition if we focus on the probability of belonging to a class or an other, i.e. if $y\in\{0,1\}$ and $h(\mathbf{x})\in[0,1]$, such variant [Cox, 1958] is defined by:

$$\ell(h(\mathbf{x}),y) = y\ln\left(1+\exp(-h(\mathbf{x}))\right)^{-1} + (1-y)\ln\left(1-\frac{1}{1+\exp(-h(\mathbf{x}))}\right).$$

Now we present the different algorithms that can be used in supervised learning.

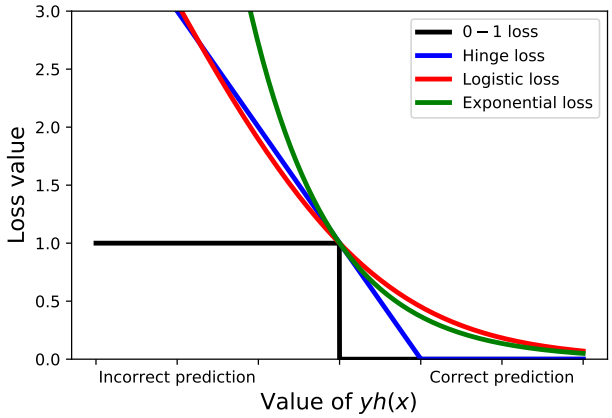


Figure 20: Examples of loss functions used for classification tasks: the hinge loss (linked to the SVM algorithm), the logisitic loss (used in for logistic regression) and the exponential loss (used in the context of boosting).

7.2 k-Nearest Neighbors

A description of the algorithm The k-Nearest Neighbors algorithm [Cover and Hart, 1967] is a non parametric method which does not impose any assumption on the underlying distribution. To predict the label of a new instance $\mathbf{x}'$, it computes the "distances" between the new query $\mathbf{x}'$ and the set of samples $\mathbf{x}_i \in S$. Then, using a selected k value, it looks for the k nearest neighbors of $\mathbf{x}'$ and predicts the label of $\mathbf{x}'$ using a majority vote, *i.e.*

$$\hat{y(\mathbf{x}')} = \arg\max_{y \in \mathcal{Y}} \frac{k_y}{k}$$

In other words, the predicted label $y(\mathbf{x}')$ is equal to the most represented class y in the k-neighborhood of the new query $\mathbf{x}'$.

Exemple 7.1. *An example of the use of the k-NN algorithm is drawn on Figure 21. It shows the importance of the choice of the k value to predict the label of a new instance. We see on this example that the new point is closed to points of both classes, that is why its prediction varies according to k. It is "red" when k is equal to* 3 *or* 5*. However, if we consider* $k = 9$*, it is predicted "blue".*

For small values of k, for instance when $k = 1$, the algorithm assigns to the new example $\mathbf{x}'$ the same label as the one of its closest data in the training set, i.e. similar examples shall share the same label. Such a rule is the simplest one and it has been shown by [Cover and Hart, 1967], when m is large enough, that the error rate is no more than twice the bayesian error, i.e. the smallest error we can achieve given the distribution of the data. Conversely, when we increase the value of k we tend to smooth our decision.

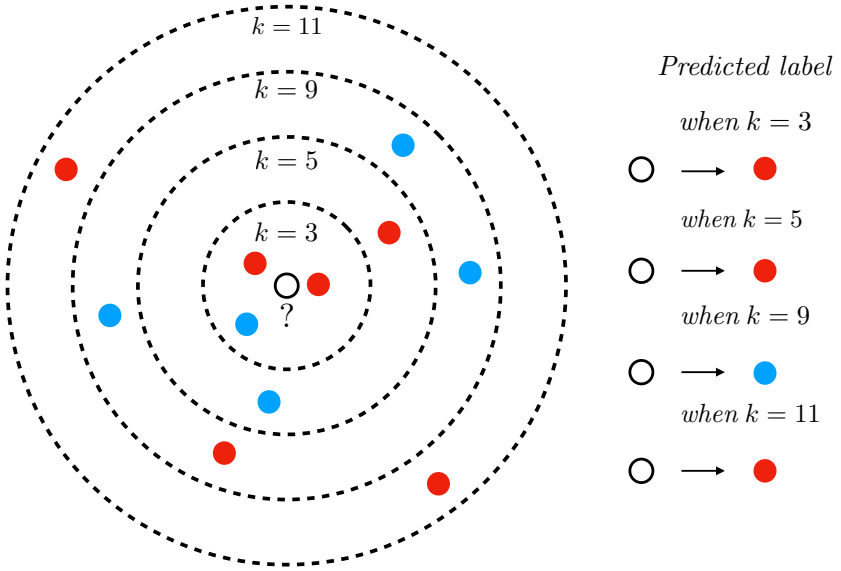


Figure 21: Illustration of the k-NN using the euclidean distance. We show that the predicted label depends on the value of k.

When k is very large (typically when $k = m$), the label given to a new data is no more than the majority class in the dataset. It raises one important question: *which value of k shall we take?* In practice, we usually use a cross validation procedure to choose the best k value.

Some problems The dimensionality d of the data and the choice of the distance are also important for such an algorithm. Indeed, when we are dealing with high-dimensional data, some features (or attributes) sometimes non informative can have a huge impact in the computation of the distance. This leads to a classification based on non relevant features and poor performances. Furthermore, some features can naturally be more important in the distance computation because they take higher values than the other one. Most the of the time, the distance we are dealing with is the Euclidean one, *i.e.* for $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$:

$$d_2^2(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|_2^2 = \sum_{j=1}^{d} (x_j - x'_j)^2.$$

Exemple 7.2. *Let us consider the following dataset with the three different descriptors*

y	x_1	x_2	x_3
1	*0.3*	*0.89*	*232*
-1	*-0.1*	*0.23*	*652*
-1	*0.2*	*0.7*	*122*
1	*0.3*	*-0.8*	*232*

If we compute the Euclidean distance between the first and second example, we have:

$$\begin{aligned} d_2^2(\mathbf{x}_1, \mathbf{x}_2) &= \|\mathbf{x}_1 - \mathbf{x}_1\|_2^2, \\ &= \sum_{j=1}^{d} (x_{1j} - x_{2j})^2, \\ &= (0.3 + 0.1)^2 + (0.89 - 0.23)^2 + (232 - 652)^2, \\ &\simeq (232 - 652)^2. \end{aligned}$$

The first and second features have no impact on the the computed distance, it only depends on the third variable.

To overcome this issue, we can modify the distance we are considering. We can also with the Manhattan distance (or L_1-norm)[9],

i.e. for $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$:

$$d_1(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|_1 = \sum_{j=1}^{d} |x_j - x'_j|.$$

We can also give a weight w_j to each features according to impact we want to give to some of them. For instance, if a feature is known to be a noisy one, we can decide to give a weight close to 0. However, if it is know to be informative, we can give to a weight close to 1. The choice of the weight can be guided by the domain knowledge.

Finally, another method we can use is to scale our data, so that all the features values lie in the same range of values or will describe a normal distribution. The transformation that are usually done are the following :

[9] Note that all of these distances can be summarized into the definition of L_p-norm, for all $p \geq 1$, defined for $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^d$ by

$$d_p(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|_p = \sqrt[p]{\sum_{j=1}^{d} |x_j - x'_j|^p}.$$

- **Normalization:** for each feature vector $\mathbf{v}_j$ we compute

$$\mathbf{v}_j \leftarrow \frac{\mathbf{v}_j - \mu_j}{\sigma_j},$$

where μ_j and σ_j are respectively the mean and the standard deviation values of the feature vector $\mathbf{v}_j$.

- **Norm hypercube:** for each feature vector $\mathbf{v}_j$ we compute

$$\mathbf{v}_j \leftarrow 2 \times \frac{\mathbf{v}_j - \min\ \mathbf{v}_j}{\max \mathbf{v}_j - \min \mathbf{v}_j} - 1.$$

- **Norm hypercube 0-1 (also called min-max normalization)** for each feature vector $\mathbf{v}_j$ we compute

$$\mathbf{v}_j \leftarrow \frac{\mathbf{v}_j - \min\ \mathbf{v}_j}{\max \mathbf{v}_j - \min \mathbf{v}_j}.$$

The objective of the first transformation is to transform the data so that they follow a centered, reduced normal distribution, *i.e.* for all $j \in [\![1, d]\!]$, $\mathbf{v}_j \sim \mathcal{N}(0, 1)$. The second and third transformations are very similar and simply consist in shifting the values of the different descriptors so that they are in the interval $[0, 1]$ or $[-1, 1]$.

With the first transformation, we modify the variance of the data, *i.e.* all features will have the same variance and thus the same weights (because they all have implicitly the same quantity of information). When you "norm" your feature vector, this does not happen.

Exemple 7.3. *Let us consider the previous data from Example 7.2, we will apply the first and second transformation on this data set*

y	x_1	x_2	x_3
1	*0.66*	*0.84*	*-0.33*
-1	*-1.45*	*-0.03*	*1.46*
-1	*0.13*	*0.59*	*-0.80*
1	*0.66*	*-1.40*	*-0.33*

y	x_1	x_2	x_3
1	*1*	*1*	*0.21*
-1	*0*	*0.61*	*1*
-1	*0.75*	*0.89*	*0*
1	*1*	*0*	*0.21*

Remark: this last suggestion is common to all algorithms based on the notion of distance or inner product. So the fact to rescale data using one of the above presented method can be used with any other learning algorithms. Most of the time, data normalization or scaling is done, so keep it mind when you will do some experiments. This generally allows us to improve the performance of the algorithms when we do not have an expert able to inform us about the importance of the features. On a practical level, it also makes it easier to compare the algorithms between them.

Algorithm 1: k-Nearest Neighbor Algorithm

Inputs : Training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$; an example to classify $\mathbf{x}'$, k the number of nearest neighbor

Output: the label $y' = y(\mathbf{x})$ of the example to classify

1: **for** $i = 1, \ldots, m$ **do**
2: compute the distance $d(\mathbf{x}_i, \mathbf{x}')$
3: **end for**
4: sort the distances by increasing order
5: keep the k-nearest neighbor among the sample S
6: count the number of occurences of each class among the nearest neighbors
7: set y' equal to the most frequent class =0

Algorithm and Complexity The algorithmic procesure of the k-NN algorithm is described in Algorith 1. As the reader may see, to predict the class of a new instance $\mathbf{x}'$, we need to keep in memory all the training set and to compute the distance of the new example to all of the training instances.

As a consequence, the complexity of this algorithm is in $\mathcal{O}(nd + nk)$ for distances computation (nd) and neighbors selection (nk). Furthermore the required memory for this algorithm is in $\mathcal{O}(n)$. This little study shows that is algorithm is not well suited for large datasets.

Refinement of the Algorithm As we have seen one drawback of this algorithm is that it needs to store the entire dataset and to compute all the distances to the training samples to predict the label of a new one. However, we can reduce the computation time and size using approaches based on the triangle inequality [Elkan, 2003], or on structured segmentations [Bentley, 1975] or by selecting the most relevant instances of the training set as in *Condensed Nearest Neighbor* [Hart, 1968] presented in Algorithm 2. This algorithm selects a subset S from from the training data S, such that the 1-NN with S' can classify the examples almost as accurately as the 1-NN does with the whole data set S.

Other refinements of the k-NN exist, the most common is a weighted k-Nearest Neighbors [Dudani, 1976]. The idea is simple and consists in assigning weights to all training examples A standard weight is the use of the inverse of the distance [Liu and Chawla, 2011]. By this way, we give less importance to the examples that are far from the tested one.

Exemple 7.4. *Let us consider an example* $\mathbf{x}'$ *we aim to classify using three instances* $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2)$ *and* $(\mathbf{x}_3, y_3)$ *using the* 3*-NN. The distances have been computed and are given below:*

Example	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$
Label	*1*	*1*	*0*
Distance	*3*	*4*	*1*

Algorithm 2: Condensed Nearest Neighbor

Inputs : Training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$
Output: S': a smaller sample

S' $= \emptyset$
Select random on example from S, remove it from S and add it to S'
for all $\mathbf{x}_i \in S$ **do**
  if $\mathbf{x}_i$ is miss-classified using S' and 1-NN **then**
    add $\mathbf{x}_i$ to S' and remove it from S
  else
    keep $\mathbf{x}_i$ in S
  end if
end for=0

If we follow the classical classification rule for this algorithm, we should predict this new data as belonging to class 1 since two of the three nearest neighbors belong to this class. This situation corresponds to the case where each example has the same weight in the decision making

On the other hand, let us now imagine that we assign a point to each example which is equal to the inverse of the distance to the new example $\mathbf{x}'$*. In this case, the sum of the weights associated with the label* 0 *is equal to* 1 *whereas it is only* $\frac{7}{12} < 1$ *for the data having the label* 1*. With this taking into account of the weights of the voters*[10]*, the new data* $\mathbf{x}'$ *would then be given the label* -1*.*

The use of such decision rule can be relevant if some region of the space are sparse and some points are isolated. Another possibility is to learn a new representation of the data by projecting the data in a (better) latent space [He and Wang, 2008, Weinberger and Saul, 2009] as is done using a Metric Learning Algorithm.

An interesting reference [von Luxburg and Bousquet, 2004] draws a link between the nearest neighbor algorithm (the 1-NN) and the next algorithm we will study, the Support Vector Machine (SVM)

7.3 Support Vector Machine

The Support Vector Machine algorithm (SVM) is probably one of the most known and used classification algorithm in Machine Learning [Vapnik and Cortes, 1995] for binary classification.

[10] this notion of majority vote or weight of the voters is very common in Machine Learning and especially in *PAC Bayesian Learning*. Despite we will not talk about it in this course, we will see that majority vote is involved in many algorithms such as boosting as presented in Section 8

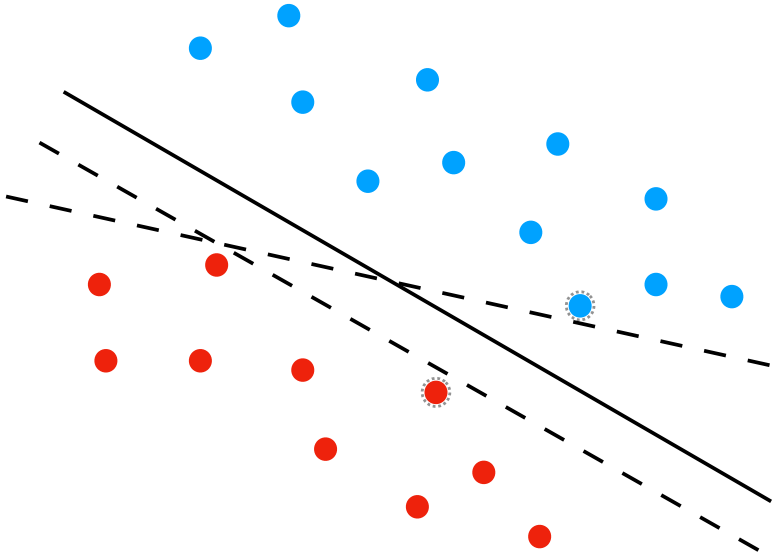


Figure 22: Representation of different linear separators for the classification task. The separator shown in solid line is the solution obtained with a SVM algorithm that maximizes the margin. The two other separators, represented in dotted line, are two other classifiers that lead to the same performances.

Linear SVM The SVM algorithm outputs an hypothesis h which returns the label of an example $\mathbf{x}$. This hypothesis is an (affine) hyperplane which separates the space into two spaces. e.g. $\{-1, +1\}$ as follows:

$$h(\mathbf{x}) = sign[\langle \mathbf{w}, \mathbf{x} \rangle + b] = \begin{cases} -1 & \text{if} \quad \langle \mathbf{w}, \mathbf{x} \rangle + b < 0, \\ +1 & \text{if} \quad \langle \mathbf{w}, \mathbf{x} \rangle + b > 0. \end{cases} \tag{2}$$

When the two classes can be perfectly separated, there are several hyperplanes, i.e. several values of $(\mathbf{w}, b)$, which can well separate the data. The idea developed by [Boser et al., 1992] is to choose the one that has the largest margin, but why? Let us have a look at Figure 22 to understand.

To understand, why, we just need to remember our main goal in Machine Learning, which is to find the best classifier that is able to generalize best on new data. If we focus on the two dotted lines, we see that are really close to the examples from one class, at test time we increase the risk that a new data, coming from the same distribution, is located on the wrong side of our separator. Thus to avoid this problem and then select the one that is able to generalize better on new data, we select the one with the largest margin.

In Figure 23, the margin γ is defined by the distance between the hyperplanes of equation $\langle \mathbf{w}, \mathbf{x} \rangle + b = \pm\rho$. Note that we can rewrite these equations as $\langle \mathbf{w}, \mathbf{x} \rangle + b = \pm 1$ by a normalization of $(\mathbf{w}, b)$. Let us now consider the points $\mathbf{x}_-$ and $\mathbf{x}_+$ which lie on the two hyperplanes as depicted in Figure 22. Each of these vectors can be decomposed as $\mathbf{x} = \mathbf{x}^{\mathbf{w}} + \mathbf{x}^{\mathbf{w}^\perp}$, where $\mathbf{x}^{\mathbf{w}}$ is colinear to $\mathbf{w}$ and $\mathbf{x}^{\mathbf{w}^\perp}$ is orthogonal to $\mathbf{w}$. These remarks lead to the following relations:

$$\begin{aligned} h(\mathbf{x}_+) - h(\mathbf{x}_-) &= 2, \\ \langle \mathbf{w}, \mathbf{x}_+ \rangle + b - (\langle \mathbf{w}, \mathbf{x}_- \rangle + b) &= 2, \\ \langle \mathbf{w}, \mathbf{x}_+^{\mathbf{w}} \rangle + \underbrace{\langle \mathbf{w}, \mathbf{x}_+^{\mathbf{w}^\perp} \rangle}_{=0} + b - (\langle \mathbf{w}, \mathbf{x}_-^{\mathbf{w}} \rangle + \underbrace{\langle \mathbf{w}, \mathbf{x}_-^{\mathbf{w}^\perp} \rangle}_{=0} + b) &= 2, \\ \langle \mathbf{w}, \mathbf{x}_+^{\mathbf{w}} \rangle - \langle \mathbf{w}, \mathbf{x}_-^{\mathbf{w}} \rangle &= 2. \end{aligned} \tag{3}$$

Furthermore, the margin γ, *i.e.* the distance between the two hyperplanes is defined as the projection of the vector $\mathbf{x}_+ - \mathbf{x}_-$ on the unit vector $\frac{\mathbf{w}}{\|\mathbf{w}\|_2}$, *i.e.*

$$\gamma = \frac{\langle \mathbf{x}_+ - \mathbf{x}_-, \mathbf{w} \rangle}{\|\mathbf{w}\|_2}.$$

Thus we have:

$$\begin{aligned} \gamma &= \frac{\langle \mathbf{x}_+ - \mathbf{x}_-, \mathbf{w} \rangle}{\|\mathbf{w}\|_2}, \\ &= \frac{1}{\|\mathbf{w}\|_2} \underbrace{\langle \mathbf{x}_+ - \mathbf{x}_-, \mathbf{w} \rangle}, \\ &\quad \downarrow \text{ developing and keeping only the colinear part} \\ &= \frac{1}{\|\mathbf{w}\|} \underbrace{\langle \mathbf{w}, \mathbf{x}_+^{\mathbf{w}} \rangle - \langle \mathbf{w}, \mathbf{x}_-^{\mathbf{w}} \rangle}, \\ &\quad \downarrow \text{ using Equation (3)} \\ \gamma &= \frac{2}{\|\mathbf{w}\|_2}. \end{aligned}$$

Thus, maximizing the margin γ is equivalent to minimizing $\frac{\|\mathbf{w}\|_2}{2}$ or $\frac{\|\mathbf{w}\|_2^2}{2}$.

The minimization problem associated to *hard margin SVM* is the following:

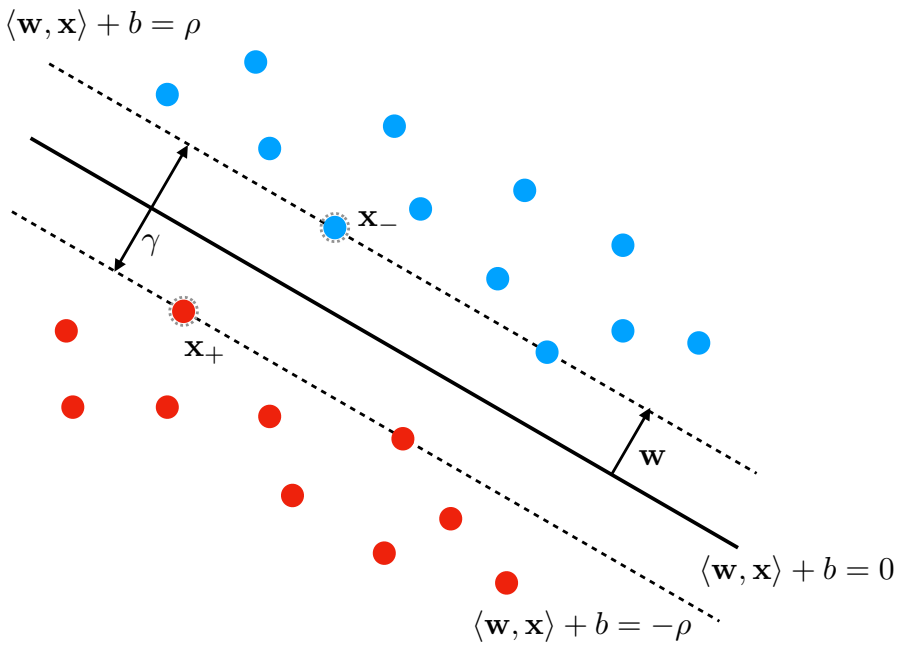


Figure 23: Illustration of the SVM algorithm when the problem is linearly separable. The circled points are the support vectors. The dotted lines represent the hyperplanes of equation $\langle \mathbf{w}, \mathbf{x} \rangle + b = \pm\rho$ and the distance between the two hyperplanes is equal to γ.

Définition 7.1: Hard Margin SVM problem

Let $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ be sample of size m. Then, the best linear separator using the SVM algorithm is the solution of the following optimization problem:

$$\begin{array}{ll} \min\limits_{(\mathbf{w},b)\in\mathbb{R}^{d+1}} & \frac{1}{2}\|\mathbf{w}\|_2^2 \\ s.t. & y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1, \quad \text{for all } i = 1, ..., m. \end{array}$$

,
Where the inequality constraint is in fact a synthetic form of the classification rule (2).

For the moment we do focus on how to solve such an optimization problem (which is here a convex and constrained optimization problem). We will see that, in practice, there are solvers capable of solving these problems without taking care of the mathematical aspects and difficulties.

The problem studied here represents an ideal situation where the data are linearly

separable. However, in most of the real cases, the two classes are not linearly separable (see Figure 24), and some points violate the constraint: $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1$. It means that the point can be on the wrong side of the hyperplane or on the right side of the hyperplane but inside the margin.
So we need to define a relaxation of the optimization problem in which that kind of errors are taken into account. These errors take the form of a vector of *slack variables* $\boldsymbol{\xi} = (\xi_1, ..., \xi_m)$ and are included in the optimization problem. This lead to a new optimization, called *soft margin SVM* where the aim is to find a right balance between margin maximization and model errors:

Définition 7.2: Soft Margin SVM problem

Let $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ be sample of size m and let us denote $\boldsymbol{\xi} = (\xi_1, ..., \xi_m)$ the vector of slack variables. Then, the best linear separator using the SVM algorithm is the solution of the following optimization problem:

$$\begin{array}{ll} \min\limits_{\xi \in \mathbb{R}^m,\ (\mathbf{w},b) \in \mathbb{R}^{d+1}} & \frac{1}{2}\|\mathbf{w}\|_2^2 + \frac{C}{m}\sum_{i=1}^m \xi_i \\ s.t. & y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 - \xi_i, \quad \text{for all } i = 1, ..., m, \\ & \xi_i \geq 0, \quad \text{for all } i = 1, ..., m. \end{array} \tag{4}$$

Compared to Definition 7.1, a term $\frac{C}{m}\sum_{i=1}^m \xi_i$ is added, which represents the the cost of violating the constraints. In the first contraint, we allow some violations by adding a term $-\xi_i$ which is necessary positive! The balance is made using a hyper-parameter C that has to be tuned during the learning process.

Proposition 7.1: Equivalent Formulation Hard Margin SVM

Let $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ be sample of size m and let us denote $\boldsymbol{\xi} = (\xi_1, ..., \xi_m)$ the vector of slack variables. If we take the constraints into account, the optimization problem 4 can be rewritten:

$$\min_{(\mathbf{w},b) \in \mathbb{R}^{d+1}} \frac{1}{2}\|\mathbf{w}\|_2^2 + \frac{C}{m}\sum_{i=1}^m [1 - y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle) + b]_+ ,$$

where $[x]_+ = \max(0, x)$, *i.e.* the hinge loss.

Proof. To show this result, we only need to focus on the slack variables $\boldsymbol{\xi}$ that are positives using the second constraint.
If we look at the first one, the value of the slack variable ξ_i is equal to zero if $y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1$, otherwise it is equal to the difference, *i.e.*

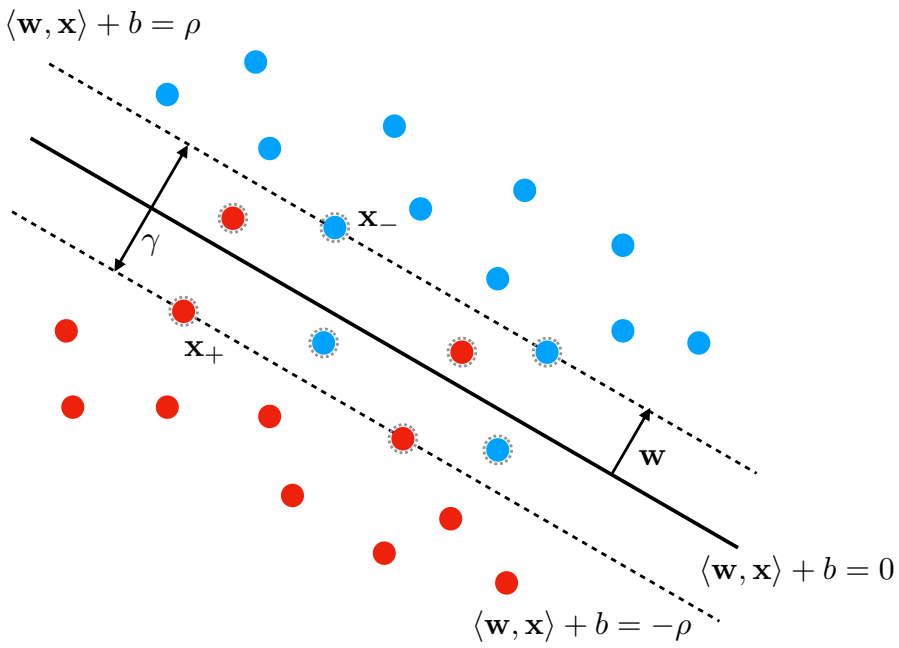


Figure 24: Illustration of the SVM algorithm when the problem is not linearly separable. The circled points are the support vectors. The dotted lines represent the hyperplanes of equation $\langle \mathbf{w}, \mathbf{x} \rangle + b = \pm\rho$ and the distance between the two hyperplanes is equal to γ.

$$\forall\, i = 1, \dots, m,\ \xi_i = \begin{cases} 0 & \text{if } 1 - y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \leq 0, \\ 1 - y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) & \text{otherwise.} \end{cases}$$

This definition can be summarized as :

$$\forall\, i = 1, \dots, m,\ \xi_i = [1 - y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle) + b]_+$$

□

However, such a problem is hard to optimize, even if it convex, it is not smooth. Furthermore, when the dimension of the data is high, the complexity is in general $\mathcal{O}(d^3)$ [Chapelle, 2007]. Thus, we usually solve what we call the *dual formulation* of such a problem to improve the speed of convergence of the optimization algorithm. We thus prefer to optimize the dual formulation of the problem:

Proposition 7.2: Dual Problem SVM

Let $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ be sample of size m. The dual formulation of the soft margin SVM described in Definition 7.2 is:

$$\begin{array}{ll} \max\limits_{\boldsymbol{\alpha}} & -\frac{1}{2}\sum_{i=1}^m \sum_{j=1}^m \alpha_i\alpha_j y_i y_j \mathbf{x}_i\mathbf{x}_j + \sum_{i=1}^m \alpha_i, \\ s.t. & 0 \leq \alpha_i \leq \dfrac{C}{m} \qquad \text{for all } i = 1, ..., m, \\ & \sum_{i=1}^m y_i\alpha_i = 0. \end{array} \tag{5}$$

The vector $\boldsymbol{\alpha}$ is the vector of Lagrangian variables or dual variables.

Proof. We have to deal with a constrained optimization problem. To get its dual formulation we need to use the Lagrangian of this problem

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^m \alpha_i \left(1 - \xi_i - y_i\left(\langle \mathbf{w}, \mathbf{x}_i\rangle + b\right)\right) - \sum_{i=1}^m \beta_i\xi_i.$$

where $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are the vectors of dual variables respectively associated to the first and second constraint.
We recall that the primal problem is a convex problem, so the solutions are in the space of values of $(\mathbf{w}, b, \boldsymbol{\xi})$ verifying:

$$\frac{\partial\mathcal{L}}{\partial\mathbf{w}}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = 0, \quad \frac{\partial\mathcal{L}}{\partial b}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = 0, \quad \text{and} \quad \frac{\partial\mathcal{L}}{\partial\boldsymbol{\xi}}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = 0.$$

These conditions give us the possibility to express the primal variable (*i.e.* $\mathbf{w}$, b and $\boldsymbol{\xi}$ as functions of the dual ones (*i.e.* $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$). Remember, this is what we have called the Karush-Kuhn-Tucker conditions. We respectively have:

$$\frac{\partial\mathcal{L}}{\partial\mathbf{w}}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = 0, \iff \mathbf{w} - \sum_{i=1}^m y_i\alpha_i\mathbf{x}_i = 0 \iff \mathbf{w} = \sum_{i=1}^m y_i\alpha_i\mathbf{x}_i, \tag{6}$$

$$\frac{\partial\mathcal{L}}{\partial b}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = 0, \iff \sum_{i=1}^m \alpha_i y_i = 0 \tag{7}$$

$$\frac{\partial\mathcal{L}}{\partial\boldsymbol{\xi}_i}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) = 0 \iff \frac{C}{m} - \alpha_i - \beta_i = 0, \iff \alpha_i, \beta_i \geq 0. \tag{8}$$

We just have to reuse these results and include them in the expression of our Lagrangian.

$$
\begin{aligned}
\mathcal{L}(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\beta}) &= \frac{1}{2}\|\mathbf{w}\|_2^2 + \sum_{i=1}^{m} \alpha_i \left(1 - \xi_i - y_i\left(\langle \mathbf{w}, \mathbf{x}_i\rangle + b\right)\right) - \sum_{i=1}^{m} \beta_i \xi_i, \\
&\quad \downarrow \text{ using Equation (6)} \\
&= \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j\rangle + \sum_{i=1}^{m} \xi_i \left(\frac{C}{m} - \alpha_i - \beta_i\right) + \sum_{i=1}^{m} \alpha_i \\
&\quad - \sum_{i=1}^{m} \alpha_i y_i \langle \sum_{j=1}^{m} y_j \alpha_j \mathbf{x}_j, \mathbf{x}_i\rangle + b \sum_{i=1}^{n} \alpha_i y_i, \\
&\quad \downarrow \text{ using Equations (7) and (8)} \\
&= \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j\rangle + \sum_{i=1}^{m} \xi_i \times 0 + \sum_{i=1}^{m} \alpha_i \\
&\quad - \sum_{i=1}^{m}\sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_j, \mathbf{x}_i\rangle + b \times 0, \\
&\quad \downarrow \text{ computing the difference} \\
&= -\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} y_i y_j \alpha_i \alpha_j \langle \mathbf{x}_i, \mathbf{x}_j\rangle + \sum_{i=1}^{m} \alpha_i.
\end{aligned}
$$

Keep in mind that we have to maximize the Lagrangian *w.r.t.* the dual variables. Furthermore Equations(7) and (8) lead to constraints of the dual formulation. □

Note that the dual optimization problem is always a strictly concave problem with respect to the dual variables. Thus, there exists one and only one solution.
Indeed, the two constraints are linear, thus convex. When it comes to the objective function, note that it can be rewritten as a quadratic form:

$$
\max_{\boldsymbol{\alpha}} \; -\frac{1}{2}\boldsymbol{\alpha}^T \mathbf{G} \boldsymbol{\alpha} + \sum_{i=1}^{m} \alpha_i,
$$

where $\mathbf{G}$ is the matrix defined by $G_{ij} = y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j\rangle$. This matrix is known as the Gram matrix and this matrix is Positive Semi Definite.

Remarks. The dual problem aims to find a solution in a different space, we rather work in the individual rather than the parameter one. Furthermore, when the solution of the dual problem is found, it is possible to have an expression of the primal one using Equation (6) for the value of $\mathbf{w}$. If we want to find the value of b we have to use the

bi-dual formulation, *i.e.* the dual of the dual problem, but we skip this point here.

If the dual formulation leads to an easier problem to solve when the number of data is small, it does not solve the problem of non linear separability of the data. To tackle this issue, we use what is commonly called the *kernel trick*.

Kernel SVM Instead of using the standard inner product between two examples, we define a function $K(\cdot,\cdot)$ which takes two vectors as input and returns a real number. Such function K is called a kernel. We also denote by $\mathbf{K}$ its associated matrix, that is (i) symmetric and (ii) positive semi-definite, i.e.:

(i) $\forall\ (\mathbf{x},\mathbf{x}') \in \mathbb{R}^d \times \mathbb{R}^d$, we have: $K(\mathbf{x},\mathbf{x}') = K(\mathbf{x}',\mathbf{x})$,

(ii) $\forall\ (\mathbf{x}_i,\mathbf{x}_j) \in \mathbb{R}^d \times \mathbb{R}^d$ and $\forall\ \mathbf{c} \in \mathbb{R}^d$, we have: $c^T\mathbf{K}c = \sum_{i=1}^m \sum_{j=1}^m c_i c_j K(\mathbf{x}_i,\mathbf{x}_j) \geq 0$.

These properties on the function (or matrix) K play a key role and lead to the following result due to Mercer [Mercer, 1909].

Théorème 7.1: Mercer Theorem

Let $\mathcal{X}$ be a compact subset of $\mathbb{R}^d$ and let K be a continuous symmetric positive semi-definite function, i.e. a kernel. Then, there exists an orthonormal basis of functions $(\Phi_j)_{j\in\mathbb{N}}$ and a sequence $(\lambda_j)_{j\in\mathbb{N}}$, where $\lambda_j \geq 0$ for all j, such that:

$$K(\mathbf{x},\mathbf{x}') = \sum_{j=1}^{\infty} \lambda_j \Phi_j(\mathbf{x})\Phi_j(\mathbf{x}') = \langle \Phi(\mathbf{x}), \Phi(\mathbf{x}') \rangle,$$

where $\Phi(\mathbf{x}) = (\sqrt{\lambda_1}\Phi_1(\mathbf{x}), ..., \sqrt{\lambda_j}\Phi_j(\mathbf{x}), ...)$ is the representation of $\mathbf{x}$ in a new space.

Thus, what we call the kernel trick is that there is no need to explicitly know the transformation Φ, the knowledge of K is enough. Furthermore, it gives the possibility to project the data in a higher dimensional space, called the latent space (possibly of infinite dimension) in which the classes are linearly separable.

Introducing the kernel K, in the optimization problem 5 leads to the following dual formulation:

$$\begin{array}{lll} \max\limits_{\boldsymbol{\alpha}} & -\frac{1}{2}\sum_{i=1}^m \sum_{j=1}^m \alpha_i\alpha_j y_i y_j K(\mathbf{x}_i,\mathbf{x}_j) + \sum_{i=1}^m \alpha_i, & \\ s.t. & 0 \leq \alpha_i \leq \dfrac{C}{m}, & \text{for all } i = 1, ..., m, \\ & \sum_{i=1}^m \alpha_i = 0, & \end{array}$$

There exists a large number of kernel functions (see [Genton, 2002] for a more complete list of kernels) among which the most used are:

- **Linear Kernel:** $K(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' \rangle$, which is the standard inner product.
- **Polynomial Kernel:** $K(\mathbf{x}, \mathbf{x}') = (\langle \mathbf{x}, \mathbf{x}' \rangle + c)^p$, where c is a constant
- **Gaussian Kernel:** $K(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{\|\mathbf{x} - \mathbf{x}'\|_2^2}{2\sigma^2}\right)$, where σ is an hyper-parameter that needs to be tuned. It controls the importance given to the similarity between two instances. The bigger this value is the less importance we give to two similar examples and the more uniform is the role of each examples in the dataset.

Exemple 7.5. *Let us consider* $\mathbf{x}$ *and* $\mathbf{z}$ *two* 2*-dimensional vectors and consider the polynomial kernel of degree* 2*.*
We aim to write this kernel as an inner product, i.e. find the mapping function Φ*, such that* $K(\mathbf{x}, \mathbf{z}) = \Phi(\mathbf{x})^T \Phi(\mathbf{z})$*.*

$$\begin{aligned} K(\mathbf{x}, \mathbf{z}) &= (x_1 z_1 + x_2 z_2 + c)^2, \\ &= c^2 + x_1^2 z_1^2 + x_2^2 z_2^2 + 2c x_1 z_1 + 2c x_2 z_2 + 2 x_1 z_1 x_2 z_2, \\ &= \left(c, x_1^2, \sqrt{2c} x_1, \sqrt{2c} x_2, \sqrt{2c} x_1 x_2\right)^T \left(c, z_1^2, \sqrt{2c} z_1, \sqrt{2c} z_2, \sqrt{2c} z_1 z_2\right), \\ &= \Phi(\mathbf{x})^T \Phi(\mathbf{z}), \end{aligned}$$

where $\Phi : \mathbf{x} \mapsto \left(c, x_1^2, \sqrt{2c} x_1, \sqrt{2c} x_2, \sqrt{2c} x_1 x_2\right)$*. Using this kernel, we implcitely project the data in a* 5*-dimensional space.*

Try to apply the same process to show that, with a 3*-dimensional dataset and a polynom of degree* 2*, the dataset is projected into a space of dimension* 6 *for* $c = 0$*.*

Let us now show why the gaussian kernel can project the data into a space of an infinite dimension. Let us consider two instances $\mathbf{x}$ and $\mathbf{z}$ in $\mathbb{R}^d$. We remind that:

$$\|\mathbf{x} - \mathbf{z}\|_2^2 = \|\mathbf{x}\|_2^2 + \|\mathbf{z}\|_2^2 - 2\langle \mathbf{x}, \mathbf{z} \rangle.$$

Thus, we can write[11]:

$$K(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{z}\|_2^2}{2\sigma^2}\right),$$

[11] See the development here for fruther details.

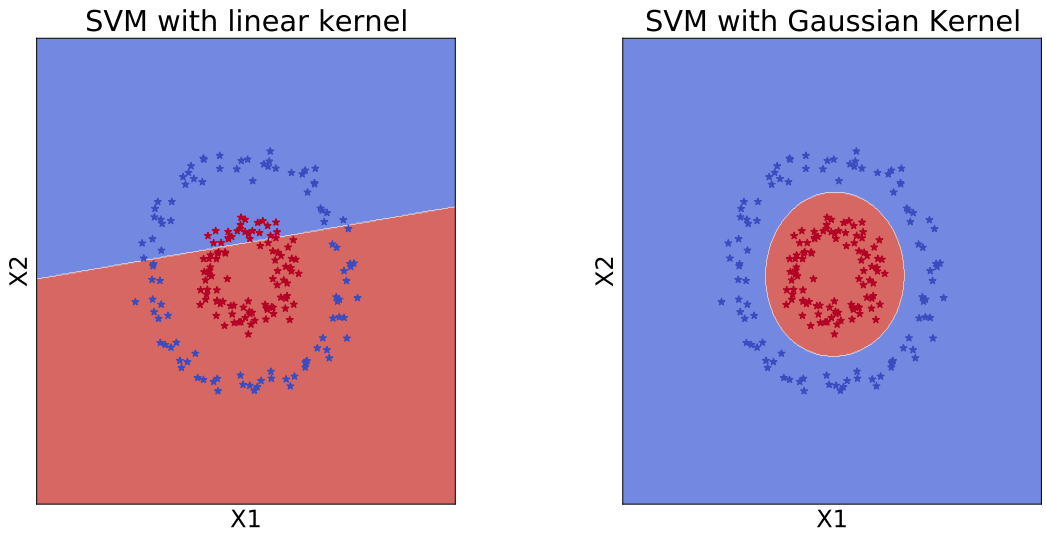


Figure 25: Illustration of the kernel SVM algorithm when the problem is not linearly separable (linear on the left, gaussian on the right) Blue points represent the negative class and the positive class is represented by red ones. The colored area represent the decision boundaries of the learned hypothesis.

$$
\begin{aligned}
&\downarrow \text{ using the previous equation the property of the exponential} \\
&= \underbrace{\exp\left(-\frac{\|\mathbf{x}\|_2^2}{2\sigma^2}\right)}\ \underbrace{\exp\left(-\frac{\|\mathbf{z}\|_2^2}{2\sigma^2}\right)}\ \exp\left(\frac{\langle \mathbf{x}, \mathbf{z}, \rangle}{\sigma^2}\right), \\
&\downarrow \text{ using the power series expansion of the exponential function} \\
&= c(\mathbf{x})c(\mathbf{z}) \sum_{k=0}^{\infty} \frac{\langle \mathbf{x}, \mathbf{z} \rangle^k}{k!\ \sigma^2}, \\
&\downarrow \text{ developping the inner product for} \\
&= \sum_{k=0}^{\infty} \left\langle \sqrt[k]{\frac{c(\mathbf{x})}{\sqrt{k!}\ \sigma}}\mathbf{x}, \sqrt[k]{\frac{c(\mathbf{z})}{\sqrt{k!}\ \sigma}}\mathbf{z} \right\rangle^k .
\end{aligned}
$$

An example of the use of this two kernel is given in Figure 25. Do you think that a polynomial of degree 2 is enough in this case to separate the two classes?

Prediction phase With the linear SVM (hard or soft), to predict the label of a new instance $\mathbf{x}'$, we have to compute

$$h(\mathbf{x}') = \text{sign}\left(\langle \mathbf{w}, \mathbf{x}' \rangle + b\right).$$

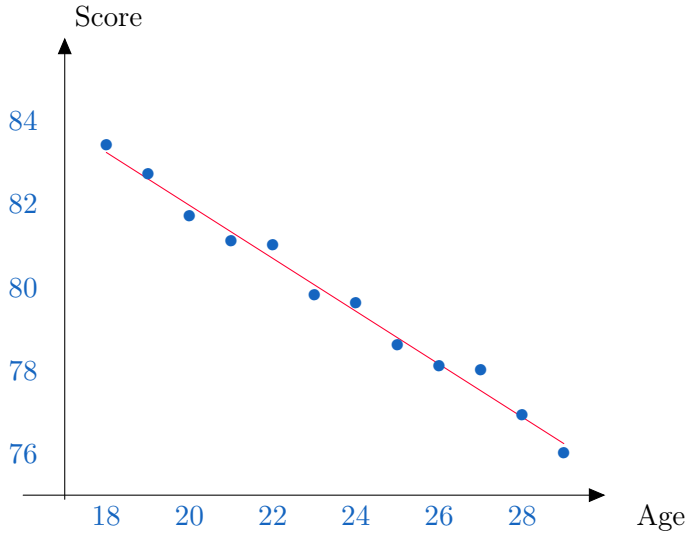


Figure 26: Example of a linear regression where the aim is to predict the score according to the age the person. The solution obtained using a linear regression is represented in red.

With the kernelized version, the hypothesis (or classifier) takes the following form:

$$h(\mathbf{x}') = \text{sign}\left(\sum_{i=1}^{m} \alpha_i K(\mathbf{x}', \mathbf{x}_i)\right).$$

Note that, in this case, we need to compute the similarity of the new example to all the training ones. However, both hypothesis remains linear, they are just working in different spaces.

The next algorithm we will study can be used both for regression or classification task depending on the nature of the output.

7.4 Linear Regression and Logistic Regression

An example of linear model has been introduced before with the support vector machine. In this section, we will present the well known linear regression and its equivalent for classification task, that logistic regression.

Linear Regression Linear regression are mainly used when we aim to predict a real value *e.g.* the price of an house or the score obtained at a given exam as illustrated in Figure 26.

As for the previous algorithm, in its simple formulation, the learned model takes the form of straight line. Given $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m \in \mathbb{R}^d \times \mathbb{R}$, we aim to find the best line which approximates the scatter plot, *i.e.* to learn a hypothesis h of the form:

$$h(\boldsymbol{\theta}, \mathbf{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_d x_d.$$

To learn the parameter $\boldsymbol{\theta}$ of the model we have to minimize a loss, as for SVM. In the case of regression tasks, as stated in Section 6.2, we usually minimize the Mean Square Error (MSE).

Proposition 7.3: Linear Regression

We consider the following probabilistic model for our data.

$$Y = \boldsymbol{\theta} X + \varepsilon,$$

where Y it the predicted variable and X is the set of variables that are used for the prediction and ε represents the error of the model.
We consider a hypothesis h of the form:

$$h(\boldsymbol{\theta}, \mathbf{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_d x_d.$$

Given a set S of m examples, $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m)$ and $\mathbf{y} = (y_1, y_2, \ldots, y_m)$ then the solution of Mean Square Error problem:

$$\min_{\boldsymbol{\theta} \in \mathbb{R}^{d+1}} \|\mathbf{y} - h(\boldsymbol{\theta}, \mathbf{X})\|_2^2 = \min_{\boldsymbol{\theta} \in \mathbb{R}^{d+1}} \sum_{i=1}^{m} (y_i - h(\boldsymbol{\theta}, \mathbf{x}_i))^2$$

is given by:

$$\hat{\boldsymbol{\theta}} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}.$$

Proof. The proof is exactly the same as the one studied during the linear model course. □

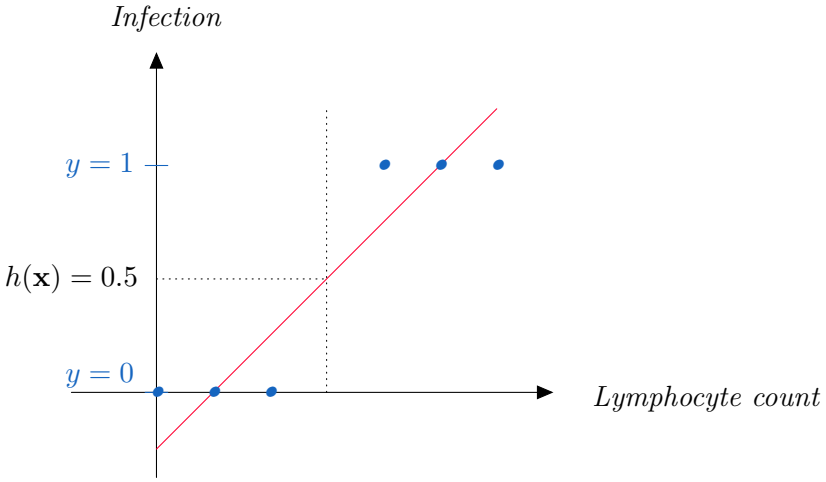
This first model is pretty simple and an analytical solution is available, meaning no training process is needed to learn the model[12].

Let us place ourselves in a slightly different regression context now and consider the following example.

[12] For more information on linear regression and especially on the statistical aspects, the reader can consult the course given by Stéphane Chrétien on Linear Models available here.

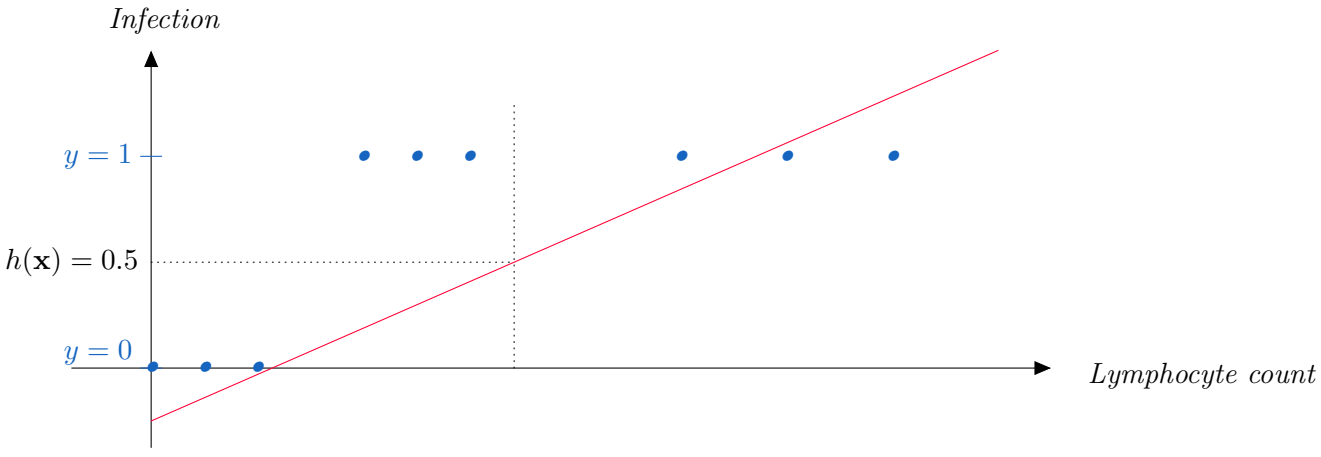
Exemple 7.6. *We are trying to build a regression model capable of determining whether or not an individual has an infection according to his lymphocyte count. The predicted variable can take two values:* 1 *if the person has an infection and* 0 *otherwise.*

At first sight, nothing prevents us from learning a linear model to try to fit our new point cloud, as illustrated below.



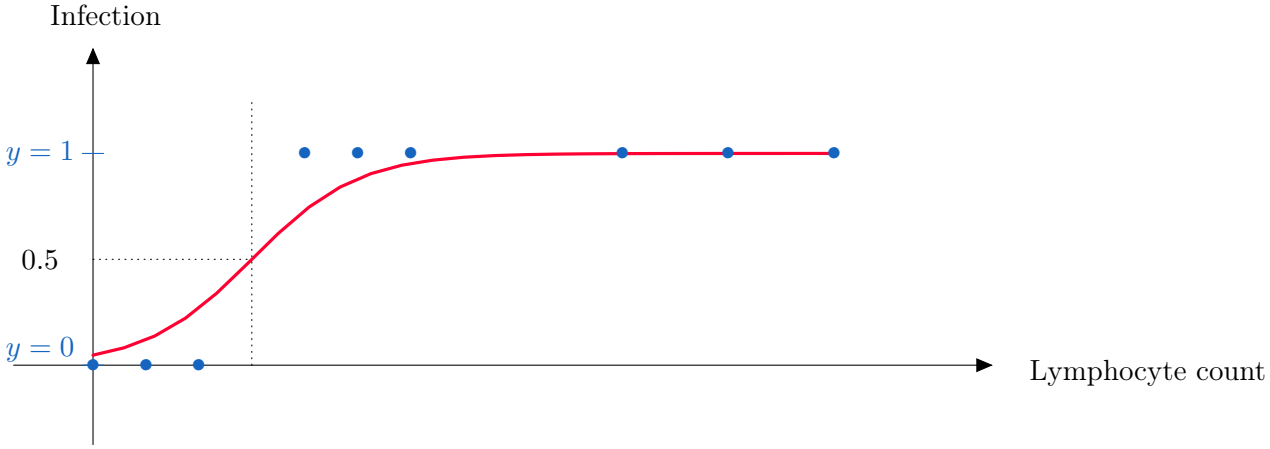
We will then simply have to take a threshold, on the values taken by our hypothesis h, beyond which an individual will be considered as sick, e.g. we consider that an example $\mathbf{x}$ belongs to the positive class when the hypothesis h returns a value greater than 0.5 *(i.e. negative on the left part and positive on the right one). In this example it works well.*

Let us now consider another one where the number of Lymphocytes can be extremely high, meaning that the infection is serious. This new dataset is represented below



This time we see that if we use the same threshold, we are missing some positive instances or infected people.

This example shows that the way we model our problem is not well chosen, we need a different structure, *i.e.* a line which is more adapted to the structure of our data. For instance, we need to have a model which be represented as follows:



Such a model takes its values in $[0, 1]$ and we can thus say that it estimates the probability of having an infection. To transform the values predicted by a linear regression model into probabilities, we use the logistic function, *i.e.* we compute:

$$\frac{1}{1 + \exp\left(-h(\mathbf{x})\right)}.$$

We talk about *linear logistic regression*.

Logistic Regression The Logistic Regression model, also called the *logit model* has been introduced in the middle of the 20^{th} century [Cox, 1958] but the use of *logit* models dates back to the end of the 19^{th} century [Cramer, 2003].

These models are close to SVM in the way that they also aim to learn a linear separator between two classes. However, they differ since are used to estimate the probability that an example belongs to a given class, for instance the positive class: $\eta = Pr(Y = 1 \mid X)$. More precisely, the logistic regression aims to compute the logarithm of the *odds*, i.e. the ratio of the probabilities. Then we estimate the log of this ratio using a linear model:

$$\ln\left(\frac{Pr(y = 1 \mid x)}{Pr(y = 0 \mid x)}\right) = h(\mathbf{w}, b, \mathbf{x}) = b + \langle \mathbf{x}, \mathbf{w} \rangle.$$

Thus, once the parameters of the model are learned, we can compute the probability of being in class 1:

$$Pr(y = 1 \mid x) = \frac{\exp(h(\mathbf{w}, b, \mathbf{x}))}{1 + \exp(h(\mathbf{w}, b, \mathbf{x}))} = \frac{1}{1 + \exp(-h(\mathbf{w}, b, \mathbf{x}))}.$$

Such function is called a *logistic function* and takes its values in $[0, 1]$. An example $\mathbf{x}_i$ is (usually) predicted in class 1 if $Pr(y = 1 \mid x) > 0.5$, i.e. if $h(\mathbf{w}, b, \mathbf{x}) > 0$. Given a task and an objective, we can choose to modify this threshold as we will see in the next chapter.

To estimate the parameters of the model, we maximize the likelihood of the data $\mathfrak{L}(\mathbf{w}, S)$, where S is a set of m examples.

$$\begin{aligned}
\mathfrak{L}(\mathbf{w}, b, S) &= \prod_{i=1}^{m} Pr(Y = y_i \mid X = x_i), \\
&\quad \downarrow \text{ separate } y_i = 0 \text{ and } y_i = 1 \\
&= \prod_{i=1, y_i=1}^{m} Pr(Y = y_i \mid X = x_i) \times \prod_{i=1, y_i=0}^{m} Pr(Y = y_i \mid X = x_i), \\
&\quad \downarrow \text{ Use the fact that the underlying law is a Bernoulli law} \\
&= \prod_{i=1}^{m} \left(\frac{1}{1 + \exp(-h(\mathbf{w}, b, \mathbf{x}_i))} \right)^{y_i} \times \left(\frac{1}{1 + \exp(h(\mathbf{w}, b, \mathbf{x}_i))} \right)^{(1-y_i)}.
\end{aligned}$$

Note that we usually prefer to minimize the negative log-likelihood of the data:

$$\begin{aligned}
\ell(\mathbf{w}, b, S) &= -\ln\left(\mathfrak{L}(\mathbf{w}, b, S)\right), \\
&= -\sum_{i=1}^{m} y_i \ln\left(\frac{1}{1 + \exp\left(-(\langle \mathbf{w}, \mathbf{x}_i \rangle + b)\right)} \right) + (1 - y_i) \ln\left(1 - \frac{1}{1 + \exp\left(-(\langle \mathbf{w}, \mathbf{x}_i \rangle + b)\right)} \right).
\end{aligned}$$

By doing so, we find the logistic loss function introduced before. In the following, for the sake of simplicity, we will set $g(\mathbf{w}, b, \mathbf{x}) = \frac{1}{1 + \exp\left(-(\langle \mathbf{w}, \mathbf{x}_i \rangle + b)\right)}$. Therefore, the optimization problem becomes:

$$\min_{\mathbf{w}, b \in \mathbb{R}^{d+1}} \quad -\frac{1}{m} \sum_{i=1}^{m} y_i \ln\left(g(\mathbf{w}, b, \mathbf{x}_i)\right) + (1 - y_i) \ln\left(1 - g(\mathbf{w}, b, \mathbf{x}_i)\right).$$

We divide the loss by a factor m in order to be consistent with the notion of empirical risk previously defined.

If SVMs and Logistic Regression models are similar geometrically speaking (they both learn a hyperplane) and present similar regularized empirical risk, a closer look in the loss functions shows that the Logistic Regression is sensitive to outliers in the data compared to SVMs and thus can lead to completely different solutions. Indeed, the loss function associated to the Logistic Regression exponentially penalizes the errors.

Compared to the linear model for reel regression, there is no analytical solutions. However, the problem is convex so can we can use a gradient based algorithm to find a solution.

Learning procedure We can compute the gradient of the function ℓ with respect to the vector $\mathbf{w} = (\mathbf{w}, b)$ and we suppose that $\mathbf{x} = (1, \mathbf{x})$ in order to simplify the notation, it avoids to manipulate the offset of the model b. Thus we have

$$\begin{aligned}
\nabla \ell(\mathbf{w}, S) = & \begin{bmatrix} \frac{\partial \ell}{\partial w_1}(\mathbf{w}, S) \\ \frac{\partial \ell}{\partial w_2}(\mathbf{w}, S) \\ \vdots \\ \frac{\partial \ell}{\partial w_{d+1}}(\mathbf{w}, S) \end{bmatrix}, \\
= & -\sum_{i=1}^{m} y_i(1 - g(\mathbf{w}, \mathbf{x}_i))\mathbf{x}_i + (1 - y_i)g(\mathbf{w}, \mathbf{x}_i)\mathbf{x}_i, \\
= & -\sum_{i=1}^{n} \left(y_i - \frac{1}{1 + \exp(-\langle \mathbf{w}, \mathbf{x}_i \rangle)} \right) \mathbf{x}_i.
\end{aligned}$$

We can then apply the gradient descent algorithm using the above expression of the gradient of negative log-likelihood (which is considered as our loss function in this case):

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - \eta \, \nabla \ell(\mathbf{w}^{(k)}, S),$$

$l = 1, 2, \ldots$ and η is the learning rate

Most of the time, we use the **Newton-Raphson** gradient descent algorithm to minimize our loss function, *i.e.* we use the hessian matrix of ℓ in our minimization procedur instead of the learning rate η.

This hessian matrix is given by:

$$\nabla^2\ell(\mathbf{w},S)=\begin{bmatrix}\dfrac{\partial^2\ell}{\partial w_1^2}(\mathbf{w},S) & \cdots & \dfrac{\partial^2\ell}{\partial w_1\partial w_{d+1}}(\mathbf{w},S)\\ \dfrac{\partial^2\ell}{\partial w_2\partial w_1}(\mathbf{w},S) & \cdots & \dfrac{\partial^2\ell}{\partial w_1\partial w_{d+1}}(\mathbf{w},S)\\ \vdots & & \vdots\\ \dfrac{\partial^2\ell}{\partial w_{d+1}\partial w_1}(\mathbf{w},S) & \cdots & \dfrac{\partial^2\ell}{\partial w_{d+1}^2}(\mathbf{w},S)\end{bmatrix}=\sum_{i=1}^{m}g(\mathbf{w},\mathbf{x}_i)\left(1-g(\mathbf{w},\mathbf{x}_i)\right)\mathbf{x}_i^T\mathbf{x}_i,$$

We can express the hessian matrix in more compact form:

$$\nabla^2\ell(\mathbf{w},\mathbf{X})=\mathbf{X}^T\mathbf{G}\mathbf{X},$$

where $\mathbf{X}\in\mathbb{R}^{m\times(d+1)}$ is the design matrix (*i.e.* matrix of the data) and $\mathbf{G}\in\mathbb{R}^{m\times m}$ is the matrix defined by:

$$G=\begin{bmatrix}g(\mathbf{w},\mathbf{x}_1)\left(1-g(\mathbf{w},\mathbf{x}_1)\right) & 0 & \cdots & \cdots & 0\\ 0 & g(\mathbf{w},\mathbf{x}_2)\left(1-g(\mathbf{w},\mathbf{x}_2)\right) & 0 & \cdots & 0\\ \vdots & \ddots & \ddots & \ddots & \vdots\\ & & & & 0\\ 0 & \cdots & \cdots & 0 & g(\mathbf{w},\mathbf{x}_m)\left(1-g(\mathbf{w},\mathbf{x}_m)\right)\end{bmatrix}.$$

Note that, with the above expression, the hessian matrix is expressed as a positive linear combination of Gram matrices and is thus a PSD matrix. The Newton-Raphson algorithm is then:

$$\mathbf{w}^{(k+1)}=\mathbf{w}^{(k)}-\left(\nabla^2\ell(\mathbf{w}^{(k)},S)\right)\ \nabla\ell(\mathbf{w}^{(k)},S),$$

which as faster rate of convergence than the standard gradient descent algorithm with constant step.

Let us now say few words about regularization as we usually find it in both regression and SVM models[13].

Regularization In order to avoid over-fitting, a regularization term of the form $\lambda\|\mathbf{w}\|$ is usually used in regression tasks. Thus, the optimization problem can be rewritten:

$$\min_{\mathbf{w},b\in\mathbb{R}^{d+1}}\ -\frac{1}{m}\sum_{i=1}^{m}y_i\ln\left(g(\mathbf{w},b,\mathbf{x}_i)\right)+(1-y_i)\ln\left(1-g(\mathbf{w},b,\mathbf{x}_i)\right)+\lambda\|\mathbf{w}\|^2.$$

[13] If you look at the SVM problem, the error rate of the algorithm is represented by the mean the value of the slacks variables and the regularization term in L_2 norm is the margin inverse of the model.

In the gaussian linear model, it can be written as

$$\min_{\boldsymbol{\theta}\in\mathbb{R}^{d+1}} \|\mathbf{y} - h(\boldsymbol{\theta}, \mathbf{X})\|_2^2 + \lambda \|\boldsymbol{\theta}\|^2 = \min_{\boldsymbol{\theta}\in\mathbb{R}^{d+1}} \sum_{i=1}^{m} (y_i - h(\boldsymbol{\theta}, \mathbf{x}_i))^2 + \lambda \|\boldsymbol{\theta}\|^2$$

In the two preceding expressions, the standard used is not specified but the standards are very often encountered: the L_1 norm $\|\cdot\|_1^2$ and the L_2 norm $\|\cdot\|_2^2$.
The latter will have an important impact on the model and especially on the parameters learned by the model:

- The L_1 norm is notably used to induce sparsity in the learned hypothesis, i.e. when we want the learned hypothesis to depend on a minimum of parameters. When we use this regularization term, we talk about *Lasso Regression*. This type of regularization is particularly used in high dimensional models, when there are many variables, as it can be the case in genetics. This regularization will allow to highlight the most important variables for the prediction task. However, it has an important drawback which is its non-differentiability at any point.
- The L_2 norm is used to avoid that some parameters take too important points compared to the other parameters of the model. In this case we speak of *Ridge Regression*.

The next algorithms we will present can be used for both regression and classification tasks. It can return the probability of being of a given class, like the Logistic Regression or directly assign the label as the SVM or the k-NN.

7.5 Decision Trees

Decision trees were introduced by [Quinlan, 1986] but the currently used version of Classification and Regression Trees algorithm was well introduced by [Breiman et al., 1984].

Decision trees consist of a series of rules that are successively applied to the dataset in order to separate the data into two or more groups. Here, we will only focus on binary decision trees, i.e. when a decision rule separates the data set in exactly two different sets.

The nature of the tree depends on the output space $\mathcal{Y}$:

- when $\mathcal{Y} \subset \mathbb{R}$, we talk about regression tree,
- when $\mathcal{Y} = \{-1, 1\}$, we talk about classification tree.

Toy dataset

Age	Height	Sex
20	175	F
32	180	M
40	175	M
28	172	M
22	165	F
40	169	F
70	170	F

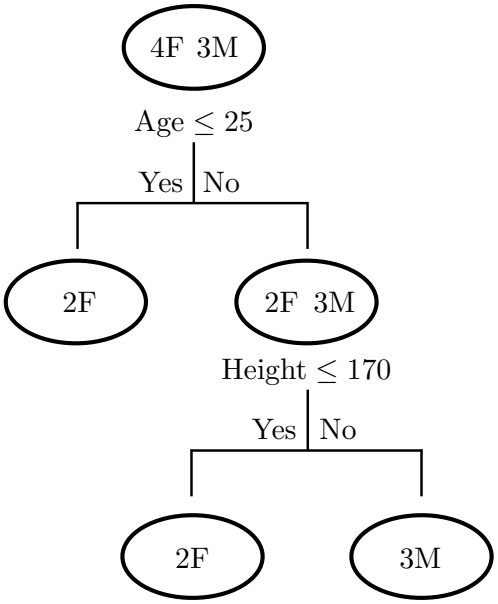


Figure 27: An example of classification tree.

An example of classification tree is provided in Figure 27 with a toy dataset, in which we aim to separate the two classes (male and female) using two descriptors (age and height) using a set of rules. The initial dataset is composed of 4 females and 3 males. The first decision rule: $Age \leq 25$ divides the initial dataset into two groups, one composed of two females and the other one of 3 males and 2 females. The group on the left is *pure* and only contains females, so we do not focus anymore on this one. Now, we apply a second decision rule: $Height \leq 170$ with which we are able to separate the remaining males and females.

Such an algorithm is able to (non linearly) separate a complex dataset when using a large number of decision rules. To see how the decision rules are chosen, we define a criterion to optimize.

For this purpose, we need two tools, a *metric* which evaluates the quality of a node and a measure of improvement after a split, called the *gain* [Safavian and Landgrebe, 1991, Rokach and Maimon, 2005].

The kind of used metrics depends on the type of tree we are dealing with. A list of such metrics are provided by [Rokach and Maimon, 2005] among which:

- the *Variance*, used for regression trees:

$$\frac{1}{n}\sum_{i=1}^{m_N}(y_i - \bar{y})^2,$$

where m_N denotes the number of examples in the node N and $\bar{y}$ the average value of y_i in the node.

- the *Gini impurity* used in classification tree. It measures the impurity of a node by computing the proportion of each classes present in the node. For instance, in binary classification [14], the Gini impurity G_N of a node N is defined by:

$$G_N = \sum_{j=1,-1} p_j(1-p_j) = 2p_1(1-p_1), \tag{9}$$

where p_j denotes the proportion of examples being in class j.

In the binary setting, the Gini impurity is a real value which belongs in $[0, 0.5]$. A value of 0 means that the node is pure, i.e. it contains only examples from one class. A value of 0.5 means that the node contains the same number of examples from both classes.

Let us now illustrate this notion. In the previous example (Figure 27) the Gini impurity of the root is $G_{\text{root}} = 2 \times \frac{4}{7}\left(1 - \frac{4}{7}\right) = \frac{24}{49}$ while it is equal to 0 and $\frac{12}{25}$ respectively on each node after the first split. We have previously said that the node on the left was *pure* because it contains only examples from one class. As this node is pure, its Gini Impurity can not be improved.

The next step consists in choosing the optimal rule to split the dataset into two nodes. This rule is chosen in order to minimize the Gini impurity at the end of the tree. For this purpose, we define the *Gini gain* Γ as follows:

$$\Gamma = G_{\text{root}} - \left(\frac{|N_L|}{|N_L + N_R|}G_{N_L} + \frac{|N_R|}{|N_L + N_R|}G_{N_R}\right),$$

where G_{N_L} and G_{N_R} denote the Gini impurity of the node on the left, respectively on the right.

Figure 28 illustrates the use of the Gini impurity as a metric to build our decision tree on the given example. The arrow between the two dashed lines represents the Gini gain Γ. On this figure, we also see that the Gini function is concave. This concavity ensures the positiveness of the gain by the Jensen Inequality [Jensen, 1906] so that each split leads to a lower classification error. Furthermore, at each step, we choose the feature and its corresponding value which maximizes the gain Γ. The decision rule is then applied and the node is separated into two different nodes until getting pure leaves.

In practice, it is always possible to lead to such perfect leaves. However, building such trees might tend to over-fitting and bad performance in generalization. To overcome this issue, we usually use a pruning strategy which can be controlled by parameters:

[14] The definition can be extended to L classes and the Gini impurety is then defined by $G_N = \sum_{j=1}^{L} p_j(1-p_j)$, where p_j is the proportion of examples of class j in the node N.

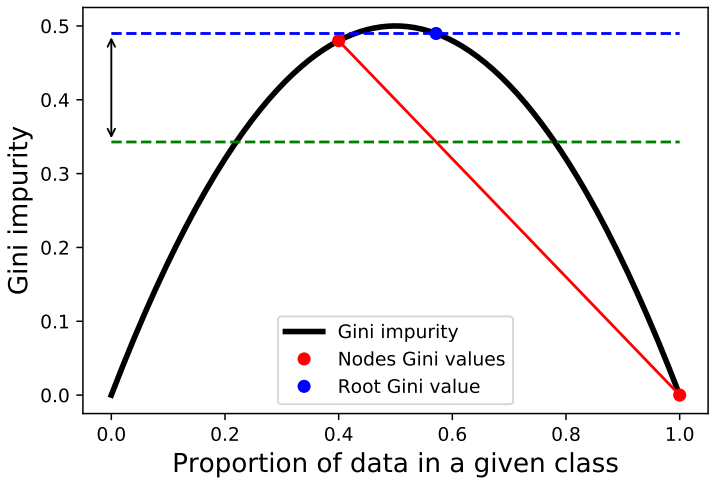


Figure 28: Illustration of the Gini Gain using the example presented in Figure **??**. The dotted green line represent the weighted sum of the Gini impurity of both nodes. The arrow between the two dashed lines represents the Gini gain Γ.

- the *size/depth* of the tree,
- the size of a node: minimum number of examples required in the node to make a new split,
- the size of a leaf: minimum number of examples in both leaves after a split,
- a threshold on the gain: the minimum value of gain required to make a new split.

Further in the document, we will see how we can use decision trees to build stronger algorithms.

We finish the presentation of standards algorithms with Neural Networks.

7.6 Neural Networks

The model presented here is a model that can be used for both classification and regression but also in some unsupervised learning algorithms.

Foundations This algorithm is very freely inspired by nature and more precisely by neurosciences on synaptic models. The first traces of neural networks can be found in works dating from the middle of the 20th century [McCulloch and Pitts, 1943]. This is the first mathematical modeling of a neuron, which is currently known as a perceptron [Rosenblatt, 1958], a representation of which is given in Figure 29.

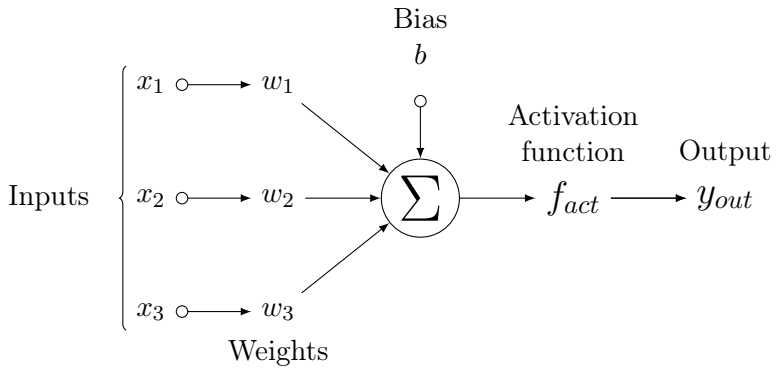


Figure 29: Illustration of a Perceptron with an input space of dimension 3. Each feature is multiplied by a parameter w and a bias b is added. When the sum is computed, the value pass through an activation function f to give the output.

This type of model takes a vector $\mathbf{x} \in \mathbb{R}^d$ as an input and depends on one parameter $(\mathbf{w}, b) \in \mathbb{R}^{d+1}$. The output $h(\mathbf{x})$ is historically computed the as the sign of the inner product of vectors $\mathbf{x}$ and $\mathbf{w}$ to which a constant value b is added, *i.e.*

$$h(\mathbf{x}) = sign\left(\langle \mathbf{w}, \mathbf{x} \rangle + b\right) = sign\left(\sum_{j=1}^{d} w_j x_j + b\right).$$

Thus, this first model was initially intended to do binary classification. The model that is used is affine (or linear if b is equal to 0) and an *activation function* is then applied to the outpout to return the predicted class:

$$h(\mathbf{x}) = \begin{cases} 0 & \text{if} \langle \mathbf{w}, \mathbf{x} \rangle + b < 0, \\ 1 & \text{otherwise.} \end{cases}$$

Note that the rule that is used to predict the class is the same as the one that we have seen with the SVM classifier. The activation function that is used here is called the *heavyside* function, illustrated in Figure 30 (left).

The first algorithm that has been used to learn the parameters $\mathbf{w}$ and b is known as the *Hebb algorithm* and the process is quite simple but it only converges for linearly separable data. The parameters are updated as follows: let us denote $\mathcal{I}$ the set of missclassified instances. Then, for all $(\mathbf{x}_i, y_i)$ such that $i \in I$ compute:

$$\mathbf{w} = \mathbf{w} + \alpha y_i \mathbf{x}_i \quad \text{and} \quad b = b + \alpha y_i,$$

where α is the learning rate. The update rule is repeated as long as I is not empty or it stops after a given number of iteration.

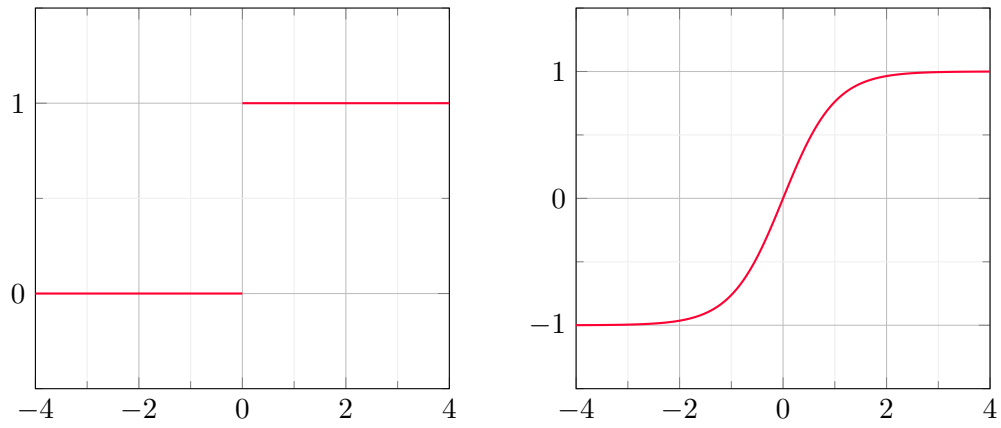


Figure 30: Illustration of two activation functions. On the left, the heavyside function, it takes the value 1 when x is positive and 0 otherwise. On the right, the function *tanh*, defined by $tanh(x) = \dfrac{e^x - e^{-x}}{e^x - e^{-x}}$, takes its values in the range $[-1, 1]$.

Another rule to learn the perceptron parameters, called *law of Widrow-Hoff*, works in a similar manner, but it also takes into account the error observed at the current state:

$$\mathbf{w} = \mathbf{w} + (y_i - h(\mathbf{x}_i))\mathbf{x}_i \quad \text{and} \quad b = b + y_i - h(\mathbf{x}_i).$$

This second update rule can be used with the sign function but we usually use it with the *tanh* function which can be seen as smoother version of the heavyside function with a little offset (see Figure 30 (right)), the sigmoid can also be used. The later has the advantage to be smooth compared the heavyside function where the derivative is equal to 0 almost everywhere and thus more suited for gradient descent algorithm.

Neural Networks The previously presented algorithms are interesting when we are dealing with problems that are linearly separable. But this situation is rarely met in practice and we need to develop more flexible and complex models to achieve performance for non linear classification problems.

For instance, the perceptron algrorithm is able to achieve good performances on the **OR** or **AND** dataset, *i.e.* a linear model is enough to classify the data, but it cannot solve the **XOR** problem. The problems are illustrated in Figure 31 using a two dimensional dataset and we effectively see that cannot separate the third dataset perfectly using a simple linear classifier, we need a more complex model, *i.e.* to learn another representation of data where the problem is linearly separable.

To do so, we are still inspired by neurosciences and the brain architecture where several neurons are connected between them, this what we call a *Neural Network*. But

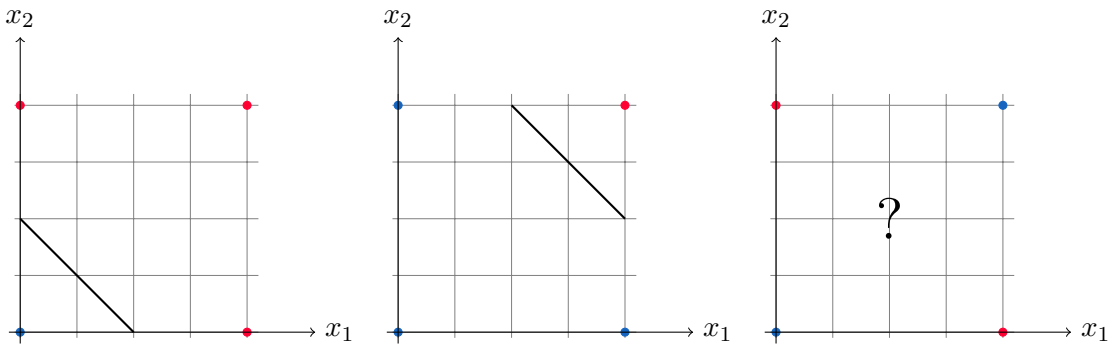


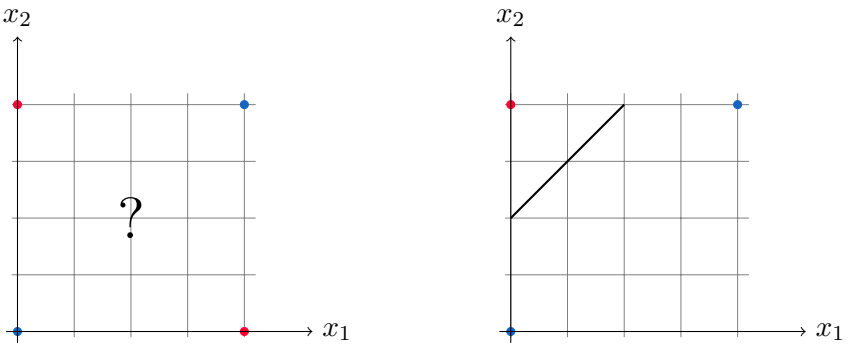
Figure 31: Representation, in a two dimensional space, of the **OR**, **AND** and **XOR** classification datasets respectively. A value x_i equal to 0 means that entry is *FALSE*, it is equal to 1 if it is *TRUE*. The color of the point is used to denote the label of the data which is determined by the logical operator: TRUE and FALSE.

before going on with the architecture, let us go back to our example and see how can solve the **XOR** problem.

To solve this problem we can perform the following transformations and the x_1 and x_2 axes:

- $x_1 \leftarrow x_1 \wedge x_2$
- $x_2 \leftarrow x_1 \vee x_2$

to have the following new representation:



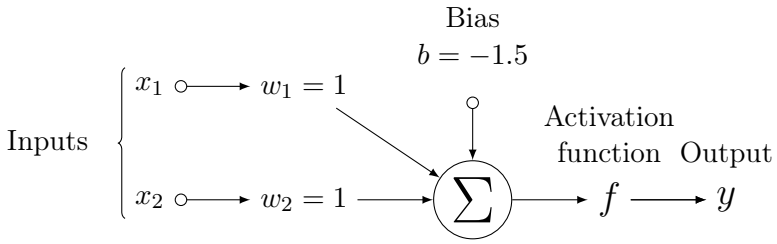
With this transformation, the two initial red points are projected at the same place in the new space because only one of their two entries was in the *TRUE* state. The representation of the blue points is the same since their two entries are the same. In this new representation, the problem is linearly separable and such a representation can be learned by a neural network, more precisely, using a multi layer perceptron.

To find the architecture, we first to do some logical reasoning and write the **XOR** function differently. This function is true if and only if exactly one of the input is true. In other words

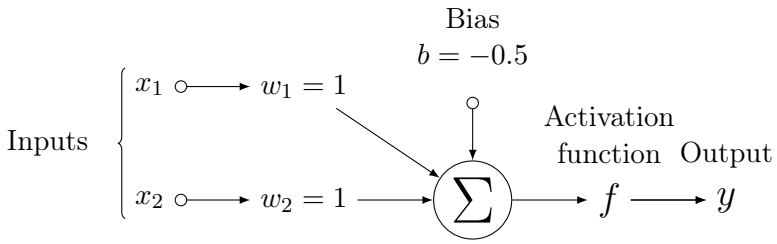
$$XOR(x_1, x_2) = (x_1 \vee x_2) \wedge (\overline{x_1 \wedge x_2}) .$$

It is now enough to translate this expression using several perceptron. For the **XOR** problem, we just how to combine the several perceptron given below. We leave it to the reader to represent the solution of the problem by combining these different perceptrons.

The **AND** perceptron:



The **OR** perceptron:



The **NOT** perceptron:

The activation function that is used is always the heavyside function.

A multiple layer perceptron is represented in Figure 32. The first layer is called *input layer*. Its size is equal to the dimension of the input space and we also add an other neuron for which the entry s always equal to 1 and which represents the bias term b.
The intermediate layers are called hidden layers. The number of layers and their sizes are defined by the user according to his needs and the problem he is facing. Again, to each hidden layer, we can associate a bias parameter that will be used to evaluate the output at the next layer.
The last layer is called the output layer, and its size also depends on the size of the output space. For a binary regression or classification problem, the output is of dimension 1.

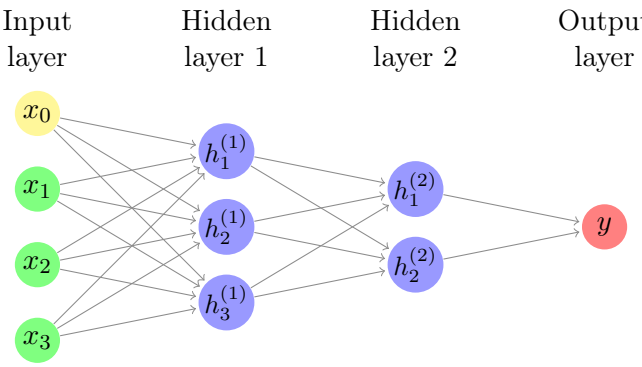


Figure 32: Representation of a multi layer perceptron. On this particular example, the input space is of dimension 3, there are two hidden layers of size 3 and 2 respectively. The output space is of dimension 1.

On the other hand, for a multi-class classification problem, the output layer will have as many neurons as there are classes in the data set.

The neural network presented in Figure 32 is said to be *fully connected*, *i.e.* all the inputs are connected to all of the outputs. However, it is possible to remove some of the connections, by cutting or setting the respective weights equal to 0. We could also imagine connections between two non successive layers.

We will finish this generalization on networks by evoking the number of parameters of a multi-layer neural network.

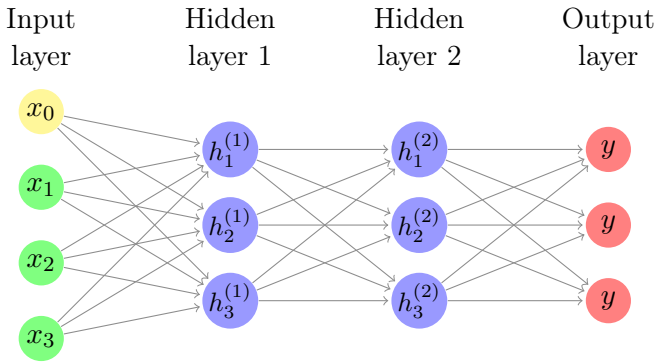
In the example given in Figure 32, we can see the number that the input is of dimension 3, the two hidden layers are respectively of dimension 3 and 2 and the output layer is of dimension 1. Further, at each hidden layers and at the input layer is associated a bias term.

So, in our example, the number of links, which is equal to the number of parameter to learn, is the dimension of input layer plus one multiplied by the dimension of the hidden layer, thus 12 parameters. We also have 8 parameters between the second and the third layer and 3 parameters between the third and the fourth layer. Thus a total of 23 parameters for this network.

More generally, the number of parameters to learn in neural network with K hidden layers

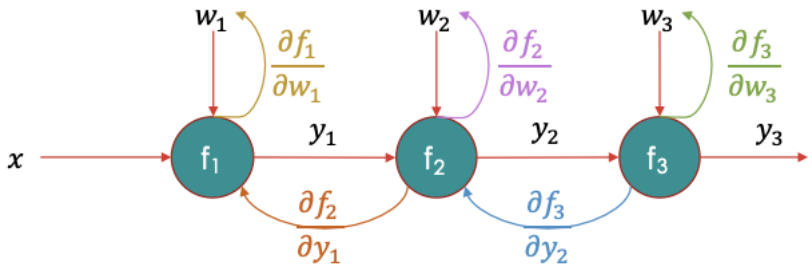
$$\sum_{k=1}^{K-1} \left(d^{(k+1)}\right) \times \left(d^{(k)} + 1\right),$$

where the $d^{(k)}$ denotes the dimension/number of units the k-th layer. In the example below, the neural network has 48 parameters.



This last example shows a multi-layer perceptron than be used for *multi-class classification*. But we will give more precision about how it works when will present different losses we can use in neural networks and after providing information about how to learn the parameters in the next paragraph.

Training and losses We will consider, for the sake of simplicity, the following network with only two hidden layers with a single unit, with a one dimensional input[15].



Where w_i, f_i denote respectively the prameters and the activation function of the i-th layer and y_i the output. The parameters of the network are updated standard gradient descent algorithm using a *Forward - Backward* procedure.

The *Forward* consists in giving data to the network one by one (stochastic) or using minibatch (subset of the data) in order to compute the loss value. The second step is the *Backward* one, where the parameters update using the data and the loss value. The main difficulty will be to trace the error along the entire network in order to update all the parameters. Indeed, if the weights of the last layer are directly linked to the output value, the same cannot be said for the weights of the first hidden layer or the input layer.

[15] This example has been extracted from the Thesis of Damien Fourure.

The gradient of the parameters of the first layers will directly depend on the gradient of the parameters of the layers further downstream. It implies to use the chain rule, to compute the update:

If f and g are two differentiable functions respectively at x and $g(x)$, the derivative of $f \circ g$ is given by

$$\frac{\partial f \circ g}{\partial x}(x) = \frac{\partial f \circ g}{\partial g}(x) \times \frac{\partial g}{\partial x}(x).$$

Let us now use this rule to update the different parameters of our network. We only compute the derivatives of y_3 *w.r.t.* w_1, w_2 and w_3. However, keep in mind that loss is used and this loss depends on y_3, but we skip this for the sake of simplicity.

- Derivative with respect to w_3:

$$\frac{\partial y_3}{\partial w_3} = \frac{\partial f_3(y_2, w_3)}{\partial w_3}.$$

- Derivative with respect to w_2:

$$\begin{aligned}
\frac{\partial y_3}{\partial w_2} &= \frac{\partial f_3(y_2, w_3)}{\partial w_2}, \\
&= \frac{\partial f_3(y_2, w_3)}{\partial y_2} \times \frac{\partial y_2}{\partial w_2}, \\
&= \frac{\partial f_3(y_2, w_3)}{\partial y_2} \times \frac{\partial f_2(y_1, w_2)}{\partial w_2}.
\end{aligned}$$

- Derivative with respect to w_1:

$$\begin{aligned}
\frac{\partial y_3}{\partial w_1} &= \frac{\partial f_3(y_2, w_3)}{\partial w_1}, \\
&= \frac{\partial f_3(y_2, w_3)}{\partial y_2} \times \frac{\partial y_2}{\partial w_1}, \\
&= \frac{\partial f_3(y_2, w_3)}{\partial y_2} \times \frac{\partial f_2(y_1, w_2)}{\partial w_2}, \\
&= \frac{\partial f_3(y_2, w_3)}{\partial y_2} \times \frac{\partial f_2(y_1, w_2)}{\partial y_1} \times \frac{\partial y_1}{\partial w_1}, \\
&= \frac{\partial f_3(y_2, w_3)}{\partial y_2} \times \frac{\partial f_2(y_1, w_2)}{\partial y_1} \times \frac{\partial f_1(x, w_1)}{\partial w_1}.
\end{aligned}$$

Some architectures

Part III

Advanced Supervised Machine Learning and Implementation

blabla

8 Advanced Supervised Algorithms

8.1 Back to Statistical Learning Theory

In Section 5, we focused on the error in generalization and talked about over-fitting or under-fitting for algorithms by focusing on the error. We have seen that in practice we will look for the best parameters that minimize the errors of our algorithm, but we never cared about this particular error. More precisely, what is the potentially minimal value reached by our algorithm? Does this mean that the algorithm is perfect? Is it really the best model I could have learned, or the right class of model?

In this part, we will see that we can decompose our error in order to bring some answers to these questions. This question of decomposition of the error will also be the origin of the development of more complex models (such as the combination of models) which will allow us to reduce some of its components.

The previous examples of the regression setting have shown that a simple model, such as a linear one, is not enough to achieve good performances, *i.e.* it presents a large *bias*. On the opposite, a complex model, a high degree polynomial one, will have a low bias but will not be able to generalize well, it has a high *variance*.

Let us show where these two terms comes from by considering that our data are generated by the following model:

$$y = f(\mathbf{x}) + \boldsymbol{\varepsilon},$$

where the expectation of $\boldsymbol{\varepsilon}$ is equal to 0 and represents some errors. Using a sample $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ generated by the previous, we learn a hypothesis h which is an estimator of f. Because h is learned from a sample S, it is considered as random quantity. We are now interested in the generalization capacities of h over all the distribution of our data according to the mean squared error: $\mathbb{E}[(y - h(\mathbf{x}))^2]$. More precisely we will study the different component of the error.

Proposition 8.1: Error Decomposition

We consider the following data generation model

$$y = f(\mathbf{x}) + \varepsilon,$$

Using a sample $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ generated by the previous, we learn a hypothesis h which is an estimator of f. The generalization error of h according to mean squared error can be written:

$$\mathbb{E}[(y - h(\mathbf{x}))^2] = (\mathbb{E}[h(\mathbf{x})] - f(\mathbf{x}))^2 + \mathbb{E}\left[(\mathbb{E}[h(\mathbf{x})] - h(\mathbf{x}))^2\right] + \mathbb{E}[(y - f(\mathbf{x}))^2].$$

Proof. We will use the *König-Huygens* formula which states that for any random variable X, we have:

$$\mathbb{E}\left[(X - \mathbb{E}[X])^2\right] = \mathbb{E}[X^2] - \mathbb{E}[X]^2.$$

We will now develop the left hand-side of our equation.

$$\begin{aligned}
\mathbb{E}[(y - h(\mathbf{x}))^2] &= \mathbb{E}[y^2 - 2h(\mathbf{x}) + h(\mathbf{x})^2], \\
&\quad \downarrow \text{ linearity of the expectation} \\
&= \underbrace{\mathbb{E}[y^2]} - \mathbb{E}[2yh(\mathbf{x})] + \underbrace{\mathbb{E}[h(\mathbf{x})^2]}, \\
&\quad \downarrow \text{ using } \textit{König-Huygens} \text{ formula} \\
&= \mathbb{E}[y]^2 + \mathbb{E}\left[(y - \mathbb{E}[y])^2\right] - -2\,\mathbb{E}[y]\,\mathbb{E}[h(\mathbf{x})] + \mathbb{E}[h(\mathbf{x})]^2 + \mathbb{E}\left[(h(\mathbf{x}) - \mathbb{E}[h(\mathbf{x})])^2\right], \\
&\quad \downarrow \text{ using the data model generation} \\
&= f(\mathbf{x})^2 + \mathbb{E}\left[(y - f(\mathbf{x}))^2\right] - 2f(\mathbf{x})\,\mathbb{E}[h(\mathbf{x})] + \mathbb{E}[h(\mathbf{x})]^2 + \mathbb{E}\left[(h(\mathbf{x}) - \mathbb{E}[h(\mathbf{x})])^2\right], \\
&\quad \downarrow \text{ using the linearity of the expectation on the remaining terms} \\
&= (\mathbb{E}[h(\mathbf{x})] - f(\mathbf{x}))^2 + \mathbb{E}\left[(\mathbb{E}[h(\mathbf{x})] - h(\mathbf{x}))^2\right] + \mathbb{E}[(y - f(\mathbf{x}))^2].
\end{aligned}$$

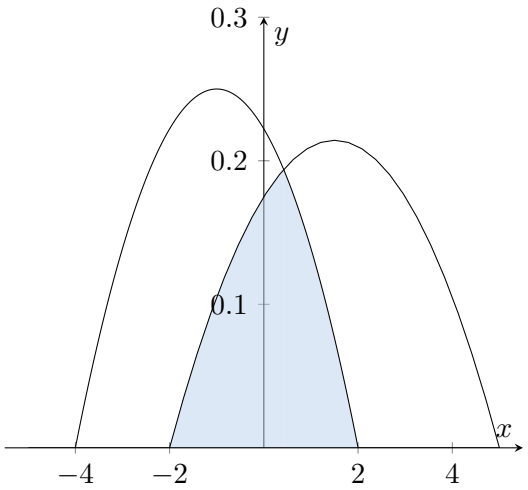
□

In this proposition, the terms on the right hand side respectively represent:

- the squared bias of h: the difference between its mean over the sample S and the value f.
- the second term is the variance of h: how much h varies around its average when the sample S changes. This represents the sensitivity of the model to the data

- the third term represents the Bayes error. It does not depend on the estimator but only on the data distribution.

Exemple 8.1. *Let us consider the two distributions represented the below. Then the Bayes error of a model is the area under both curves, i.e. the space where any classifier is not able to distinguish if the data comes from a distribution or another.*



For instance, you can try to compute the Bayes error of the following two densities for a two classes:

$$d_1 = \begin{cases} \frac{3}{2}x^2 + x & \textit{when } 0 \leq x \leq 1, \\ 0 & \textit{otherwise.} \end{cases} \quad \textit{and} \quad d_2 = \begin{cases} 1 & \textit{when } \frac{3}{4} \leq x \leq \frac{7}{4}, \\ 0 & \textit{otherwise.} \end{cases}$$

Regarding the previous proposition, the quantity we thus aim to minimize is the **excess of risk**, that is:

$$\mathbb{E}[(y - h(\mathbf{x}))^2] - \mathbb{E}[(y - f(\mathbf{x}))^2].$$

If the study has been conducted of the special case of the regression setting, a similar one can done for the general case.

We now suppose more generally that the observations are sampled from a joint distribution $\mathcal{D} = \mathcal{X} \times \mathcal{Y}$ associated to a measure μ. We also consider a loss function ℓ

$$\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R},$$

which quantifies the cost of the error, of a hypothesis h, by predicting $h(\mathbf{x})$ when the true value is y. The function or hypothesis h we are looking for is the one that minimizes the expected error over $\mathcal{D}$, *i.e.*

$$\mathcal{R}(h) = \mathop{\mathbb{E}}_{(\mathbf{x},y)\sim\mathcal{D}}[\ell(y, h(\mathbf{x}))] = \int_{\mathcal{X}\times\mathcal{Y}} \ell(y, h(\mathbf{x}))d\mu.$$

Keep in mind that, in practice, the hypothesis h is learned on a finite sample. We now assume that we minimize the risk over a function/hypothesis space $\mathcal{H}$. If we denote by $\mathcal{R}^\star$ the Bayes risk, we can decompose the **Bayes** regret as:

$$\mathcal{R}(h) - \mathcal{R}^\star = \left(\mathcal{R}(h) - \inf_{g\in\mathcal{H}} \mathcal{R}(g)\right) + \inf_{g\in\mathcal{H}} \mathcal{R}(g) - \mathcal{R}^\star.$$

The first term is the excess of risk of h with respect to the best function in the hypothesis space $\mathcal{H}$ The second term is the **approximation error**, *i.e.* the smallest excess of risk we can achieve using a function in $\mathcal{H}$. This is bias term which does not depend on the data but only the hypothesis space $\mathcal{H}$.

We will now draw the link with the generalization bounds presented in Section 5 and to go a little bit further by bounding the **excess of risk**.

If we denote by h the hypothesis obtained by minimizing the empirical risk over $\mathcal{H}$ using a sample S:

$$h \in \arg\min_{g\in\mathcal{H}} \mathcal{R}_S(g).$$

We will also denote by $h_\mathcal{H}$ the minimizer of the risk $\mathcal{R}$ over the hypotheses space H, *i.e.*

$$h_\mathcal{H} = \arg\min_{g\in\mathcal{H}} \mathcal{R}(g).$$

The **excess of risk**: $\mathcal{R}(h_S) - \inf_{g\in\mathcal{H}} \mathcal{R}(g)$ can be rewritten as the sum of three terms:

$$\mathcal{R}(h) - \mathcal{R}(h_\mathcal{H}) \leq (\mathcal{R}(h) - \mathcal{R}_S(h)) + (\mathcal{R}_S(h) - \mathcal{R}_S(h_\mathcal{H})) + (\mathcal{R}_S(h_\mathcal{H}) - \mathcal{R}(h_\mathcal{H})),$$

where $(\mathcal{R}(h) - \mathcal{R}_S(h))$ is the difference between the true risk and the empirical risk of the hypothesis h_S. This quantity is the one we are interested in, when it comes to study the generalization of the algorithms. $(\mathcal{R}_S(h) - \mathcal{R}_S(h_\mathcal{H}))$ is a non positive term by construction. $(\mathcal{R}_S(h_\mathcal{H}) - \mathcal{R}(h_\mathcal{H}))$ is easier to control as it involves a deterministic function and the law of large numbers applies.

We can show that the excess of risk can be bounded by a quantity $\mathfrak{R}(\mathcal{H})$ which is the *Rademacher complexity* the hypotheses space $\mathcal{H}$. It is measure of richness of the hypotheses space $\mathcal{H}$.

Thus, the Bayes regret can be bounded as:

$$\mathcal{R}(h) - \mathcal{R}^\star \leq \inf_{g \in \mathcal{H}} \mathcal{R}(g) - \mathcal{R}^\star + \mathfrak{R}(\mathcal{H}).$$

This result illustrates a little more generally the bias variance trade-off for risk minimization. It makes explicit the link between complexity and sample size.

8.2 Model combination

In what we have previously seen, we only use one model to answer to a given problem, *i.e.* a linear regression to predict the price of an house given its different characteristic or a logistic regression to predict if someone is infected or not.

But why use only one model? Would not it be more interesting to use several models to create a more powerful one?

The answer is effectively yes, it is more interesting to do so and currently, the combination of models are the one that perform the most on lots of tasks.

But how to create such a combination of models?

A naïve rule Let us imagine we want to create K models using a sample $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ of size m. We can train an hypothesis h_k for k from 1 to K using the same training sample S for each k.

After that, we can combine the different hypotheses into a single one, noted H_K, doing the average to take our final decision:

- for a regression task, the predicted value will correspond to the mean value over all prediction made by hypotheses h_k, *i.e.*

$$H_K(\mathbf{x}) = \frac{1}{K} \sum_{k=1}^{K} h_k(\mathbf{x}).$$

- for a classification task, we can apply the same rule. However, instead of taking the mean value as the output of the combined classifier, we rather take the sign of this mean value as the output, *i.e.*.

$$H_K(\mathbf{x}) = sign\left(\frac{1}{K} \sum_{k=1}^{K} h_k(\mathbf{x}) \right).$$

In the case where the hypotheses h_k return a value that is -1 or 1, our hypothesis H_K can then be seen as an majority vote with equal weights.

On the contrary, if the hypotheses h_k return real values, then we can imagine that it is a weighted majority vote.

Although this rule is simple in practice, it is not very useful.

Indeed, recall that if all our problems are convex, there is a good chance that all the hypotheses h_k are similar if we use the same training set each time.

Based on hyper-parameter One could then be tempted to vary the hypotheses by imposing different hyper-parameter values for each hypothesis, *e.g.* one could impose K different hyper-parameter values in order to obtain K different models.

Again this solution is not satisfactory. On the contrary, it may lead to hypotheses with low predictive power. Moreover, using such a process would call into question the cross-validation process presented earlier which allows to optimize the values of these hyper-parameters.

We can see that, even if our algorithm does not depend on hyper-parameter, it is possible to improve it using a single training set S. These two methods are known as **Bagging** and **Boosting**.

These two methods act differently on the performance of algorithms or more precisely on the different components of the error of an algorithm.

8.2.1 Bagging and Random Forests

Bagging It is a way to combine models that have good performances on the training set. We have previously seen that we can decompose our *Bayes Regret* into the sum of two terms

$$\mathcal{R}(h) - \mathcal{R}^\star \leq \inf_{g \in \mathcal{H}} \mathcal{R}(g) - \mathcal{R}^\star + \mathfrak{R}(\mathcal{H}).$$

What will interest us here is more precisely the variance term in the error decomposition. This value will show us how sensitive the algorithm is to the variation of the training set, if this value is low, our algorithm will be little sensitive to variations of the training set.

Let us show this on a little example illustrated on Figure 33 for a regression task (note this is similar for classification tasks).

We will learn a polynomial regression model of degree 15 on different data from the same distribution. Each model is learned on a training set of size 30. In total, we learn 10 different models which are represented on the graph on the left. We note that

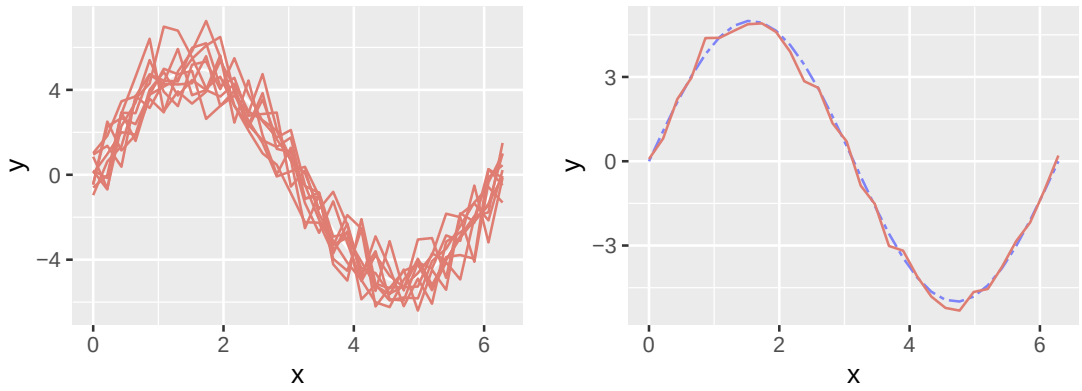


Figure 33: Illustration of the variance reduction phenomenon for the combination of models. On the left, we represent the 10 models learned on different data sets from the same distribution and on the right the model obtained after the combination of the 10 previous models (in red) as well as the regression function (in blue)

the variance of the models is very important, i.e. for the same value on the abscissa, the models return very different values on the ordinate, so it is very sensitive to the training data.

On the other hand, if we average the results of all the 10 models, we obtain the graph on the right. We notice that the latter show much less variation around the true distribution of the data. We have therefore succeeded in reducing the variance by combining several models.

Through bagging, we wish to exploit the fact that the variance of a set of models is smaller than the variance of the models separately. Moreover, the combined model retains a very important predictive power, *i.e.* it remains accurate.

In this example the models learned are on different data each time. But in practice we have only one training set S. So we have to find a way to create several training sets S_k from this set S. This can be done using the Bootstrap method, which is a sampling method based on a random draw. How does it work in practice?

Let us consider of training set S of size m. To create a bootstrap sample S_k of the same size m it will be necessary simply to carry out a draw with replacement of m examples in the set S, *i.e.* we have the same probability to draw each example in the set S.

With this procedure, the same example may occur several times in the same training set. Thus the algorithm, in order to minimize its error, will have to pay more attention to this example and will give it more importance. This sampling method will thus create diversity in the learned models. Indeed, the sets S_t being different, the hypotheses h_t

Algorithm 3: Bagging

Inputs : Training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, number of model T
Output: Hypothesis H_k
1: **for** $t = 1$ to T **do**
2: create a bootstrap sample S_t of size m using S.
3: learn a hypothesis h_t using S_t
4: **end for**
5: set $H_T = \frac{1}{T}\sum_{t=1}^T h_t$ =0

will focus on different regions of the data space.

The Bagging procedure is summarized in Algorithm 3. As one can then guess after this reading, bagging means bootstrap aggregating: generate several samples and hypothesis and then you aggregate the results.

A theoretical analysis We will now try to explain more precisely why the bagging procedure is working well. To do this, let us consider data $(\mathbf{x}, y)$ from a distribution $\mathcal{D}$ where $\mathbf{x}$ s the feature vector and y is the response variable or the value to predict.

In a regression setting, we will then learn a hypothesis h which aims to predict the value y according to $\mathbf{x}$. For the sake of simplicity, we suppose there exists a *true function* r such that $r(\mathbf{x}) = y$ for all $(\mathbf{x}, y) \sim \mathcal{D} = \mathcal{X} \times \mathcal{Y}$. We also consider a training set S of size m. In a bagging setting, remember that we learn a set of hypothesis h_t, $t = 1, \ldots, T$ given a bootstrap sample S_t drawn from S. Thus for any instance $\mathbf{x}$ and any hypothesis h_t

$$h_t(\mathbf{x}) = r(\mathbf{x}) + \varepsilon_t(\mathbf{x}),$$

where $\varepsilon_t(\mathbf{x})$ is the error between the predicted value and the true value of the function at $\mathbf{x}$.

Keep it mind that, in a regression setting, the error we consider is the MSE, *i.e.* the mean squared error. Thus the generalization error of a single hypothesis h is defined by:

$$\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{X}}\left[(h(\mathbf{x}) - r(\mathbf{x}))^2\right] = \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{X}}\left[\varepsilon(\mathbf{x})^2\right].$$

And the generalization error of our averaged classifier $H_K = \frac{1}{T}\sum_{t=1}^T h_t$ is

$$\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{X}}\left[(H_T(\mathbf{x}) - r(\mathbf{x}))^2\right] = \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{X}}\left[\left(\frac{1}{T}\sum_{t=1}^T h_t(\mathbf{x}) - r(\mathbf{x})\right)^2\right],$$

$$\downarrow \text{ applying definition}$$

$$= \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{X}}\left[\left(\frac{1}{T}\sum_{t=1}^{T}\varepsilon_t(\mathbf{x})\right)^2\right].$$

Let us suppose now that our error ε_t are centered (mean value equal to 0) and uncorrelated. We can rewrite the previous expression as:

$$\begin{aligned}
\mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{X}}\left[(H_T(\mathbf{x}) - r(\mathbf{x}))^2\right] &= \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{X}}\left[\left(\frac{1}{T}\sum_{t=1}^{T}\varepsilon_t(\mathbf{x})\right)^2\right], \\
&\downarrow \text{ we develop the square term} \\
&= \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{X}}\left[\left(\frac{1}{T}\sum_{t_1=1}^{T}\varepsilon_{t_1}(\mathbf{x})\right)\left(\frac{1}{T}\sum_{t_2=1}^{T}\varepsilon_{t_2}(\mathbf{x})\right)\right], \\
&\downarrow \text{ we use the fact that } \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{X}}[\varepsilon_{t_1}(\mathbf{x})\varepsilon_{t_2}(\mathbf{x})] = 0 \text{ for all } t_1 \neq t_2. \\
&= \frac{1}{T^2}\sum_{t=1}^{T}\varepsilon_t(\mathbf{x})^2 . \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{X}}\left[(H_T(\mathbf{x}) - r(\mathbf{x}))^2\right], \\
&= \frac{1}{T}\left(\frac{1}{T}\sum_{t=1}^{T}\varepsilon_t(\mathbf{x})^2\right).
\end{aligned}$$

This last equation shows that the generalization error of a bagging hypothesis is just the average error of the set of the T hypotheses divided by T.

Note that it practice, the assumption of uncorrelated error is essentially wrong because several hypotheses h_t are learned using similar bootstrap samples S_t. However, we can still show that the error of bagging hypotheses is no more than average error of the set of hypotheses h_t.

Let us now present a well known algorithm based on the bagging procedure: *Random Forests* algorithm.

Random Forests When we introduced decision trees, we said that the size of the tree, i.e. its depth, depends on the data. The tree will thus grow until we obtain pure leaves.

If we go back to our error decomposition story, the (deep) decision trees thus form hypotheses with a low bias (an error rate that decreases with depth) but with a high variance. They are very sensitive to the data and the structure can vary greatly from one training set to another.

Algorithm 4: Random Forests

Inputs : Training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$, number of trees T, a sample size m' and a number of features p'

Output: Hypothesis H_K

1: **for** $t = 1$ to T **do**
2: create a bootstrap sample S_t of size $m' < m$ using S.
3: Build a decision tree h_t where at each split, a random subsample of p' features are used to split the node.
4: **end for**
5: set $H_T = \frac{1}{T} \sum_{t=1}^{T} h_t$ =0

In fact, we find ourselves in exactly the same case as our regression example presented in Figure 33. We will therefore proceed to a combination of tree models in order to reduce the variance of the trees while maintaining their predictive capacities using bagging. This combination of trees by bagging gives rise to the random forest algorithm founded in the early 21^{th} century [Breiman, 2001].

The idea is then to build several trees based on different training sets S_t that are drawn randomly from S with replacement, *i.e.* using the bootstrap procedure and to combine the results of the different trees. But the random forest algorithm (presented in Algorithm 4 is in fact more sophisticated than that. It is based on the principle of double sampling: sampling both on the examples and on the variables.

This double sampling will make it possible to create a diversity at the sample level and features level. Compared to the standard bagging algorithm, note the boostrap sample is of size $m' < m$ and that the number of used features at each node of a given tree is less than the dimension d of the data.

From an algorithmic point of view, it allows to learn the different trees faster: you have less examples and you the splitting procedure is applied to a less number of features.

The fact to use a sample of size $m' < m$ is also used to compute what is called the *Out Of Bag* error (or score) which is used in most of bagging-based algorithms instead of cross-validation. We do not provide much details here about the way it works but the reader can consider it as an alternative of the cross-validation process.

The presented algorithms is the simplest one. It is not rare to give different weights to the trees according to their performance, remember, this our weighted majority vote.

Decision Trees and Random Forests are used in many applications such as finance, security or social sciences in general. These algorithms have the advantage to be easy to build and their decision is easy to understand (explainable AI), you just to follow the

road of the data in the tree.

This first way to combine hypotheses is essentially used for *strong hypotheses*, those with high predicted value. We will see in the next section that there is another way to combine trees using the so called *Boosting* procedure for *weak hypotheses*.

8.2.2 Boosting

This section is largely inspired by Chapter 7 of the book *Foundations of Machine Learning* [Mohri et al., 2012].

Setting Let us introduce to definitions first of *strong* and *weak* learnability that are given in the context of classification, where the boosting is mainly used. Thus We can use indifferently the terms classifier and hypothesis.

Définition 8.1: Strong Learability [Mohri et al., 2012]

A concept class $\mathcal{C}$ is said to be (strongly) PAC learnable if there exists an algorithm $\mathcal{A}$ and polynomial function *poly* such that for any $\varepsilon > 0$ and $\delta > 0$, for all distributions $\mathcal{X}$ and for any target concept $c \in \mathcal{C}$, the following holds for any sample size $m \geq poly(1/\varepsilon, 1/\delta, d, size(c))$:

$$\underset{S \sim \mathcal{D}^m}{\mathbb{P}} [\mathcal{R}(h_s) \leq \varepsilon] \geq 1 - \delta,$$

where h_S is the hypothesis returned by $\mathcal{A}$ when trained on S. If $\mathcal{A}$ further runs in $poly(1/\varepsilon, 1/\delta, d, size(c))$, then $\mathcal{C}$ is said to be efficiently PAC-learnable.

In this definition, a concept c is a function from $\mathcal{X}$ to $\mathcal{Y}$ that reach a specific target. As an example, a concept may be the set of points inside a triangle.
A concept class is a set of concepts we may wish to learn and is denoted by $\mathcal{C}$. This could, for example, be the set of all triangles in the plane.

According to the previous definition, a concept class $\mathcal{C}$ is thus PAC-learnable if the hypothesis returned by the algorithm after observing a number of points polynomial in $1/\varepsilon$ and $1/\delta$ is approximately correct (error at most ε) with high probability (at least $1 - \delta$).

Définition 8.2: Weak Learability [Mohri et al., 2012]

A concept class $\mathcal{C}$ is said to be weakly PAC learnable if there exists an algorithm $\mathcal{A}, \gamma > 0$ and polynomial function *poly* such that for any $\delta > 0$, for all distributions $\mathcal{X}$ and for any target concept $c \in \mathcal{C}$, the following holds for any sample size $m \geq poly(1/\delta, d, size(c))$:

$$\mathop{\mathbb{P}}_{S \sim \mathcal{D}^m} \left[\mathcal{R}(h_s) \leq \frac{1}{2} - \gamma \right] \geq 1 - \delta,$$

where h_S is the hypothesis returned by $\mathcal{A}$ when trained on S. when such an algorithm exists, it is called a weak learning algorithm, a weak learner or a base classifier.

The idea is quite similar as for strong hypotheses or strong learners. Note that the difference between the two definitions is based on the error of the hypothesis h_S. In the first one, we require this classifier to achieve an error of at most ε for a sufficiently large sample $m(\varepsilon)$. In the second one we just want the same classifier to be slightly better (of a parameter $\gamma > 0$) than the random classifier.

Strong classifiers have been met before when have presented decision trees or over-parameterized neural networks. Weak classifiers are also naturally present in the literature depending on the task, *e.g.* small decision trees with a depth of one or two or linear classifiers as linear SVM when the problem is non linearly separable.

If bagging has worked with strong classifiers, boosting will work with weak ones. It will try to combine them in order to build a strong classifier, but the way it works is different from the bagging procedure. Instead of building bootstrap samples, we will modify the data distribution in order to build a sequence of classifier h_k where h_{t+1} will try to correct the errors made by the previous classifiers h_t.

Adaboost A well-known boosting algorithm of boosting is *Adaboost* [Freund and Schapire, 1999] which iteratively focuses on examples difficult to classify. The algorithm is presented in Algorithm 5 and an illustration is provided on Figure 34

Let us say few words about this algorithm before analyzing it.

At a round t, the i-th training samples has the weight $w_i^{(t)}$. For $t = 1$, all the training samples have the same weights equal to $1/m$. A hypothesis h_t is learned and we can compute its classification error ε_t

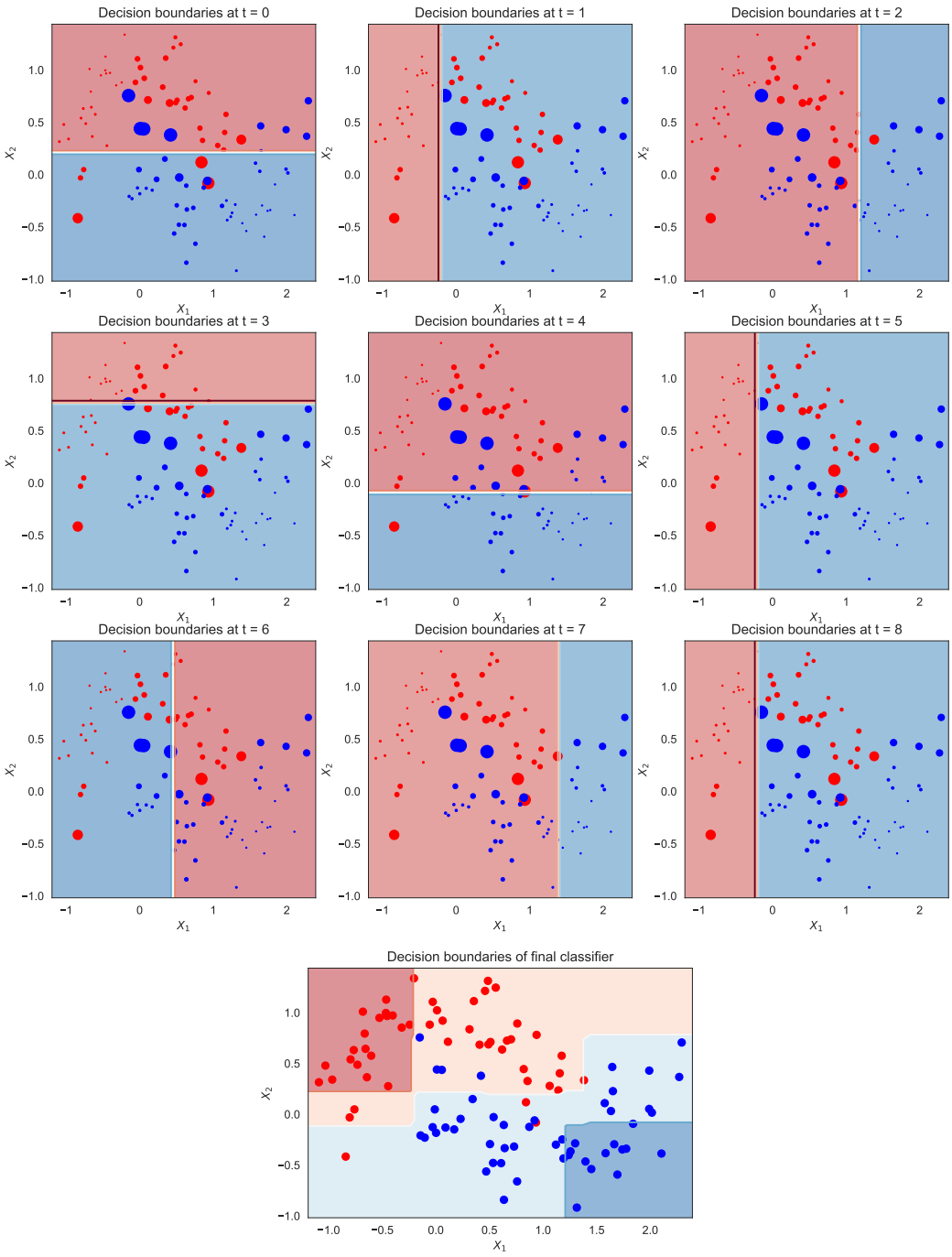


Figure 34: Illustration of the first nine iterations of Adaboost algorithm on a binary classification task. The colors represent the decision boundaries and the size of the points represents their weights in the sample distribution. The figure below represent the decision boundaries of the Adaboost algorithm after 10 rounds.

Algorithm 5: Adaboost

Inputs : Training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, number of model T
Output: Hypothesis H_T

1: **for** $i = 1$ to m **do**
2: the weight of the i-th example $w_i^{(1)}$ is equal to $1/m$
3: **end for**
4: **for** $t = 1$ to T **do**
5: learn a base classifier h_t using S with the weights $\mathbf{w}^{(t)}$.
6: compute the error $\varepsilon_t = \sum_{i=1}^m w_i^{(t)} \mathbb{1}_{\{h_t(\mathbf{x}_i) y_i < 0\}}$ of h_t
7: set $\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_t}{\varepsilon_t} \right)$
8: set $Z_t = 2\sqrt{\varepsilon_t (1 - \varepsilon_t)}$
9: **for** $i = 1$ to m **do**
10: set $w_i^{(t+1)} = w_i^{(t)} \frac{\exp(-\alpha_t y_i h_t(\mathbf{x}_i))}{Z_t}$
11: **end for**
12: **end for**
13: set $H_T = \frac{1}{T} \sum_{t=1}^T \alpha_t h_t$ =0

$$\varepsilon_t = \sum_{i=1}^m w_i^{(t)} \mathbb{1}_{\{h_t(\mathbf{x}_i) y_i < 0\}}$$

Using this value, we can compute the *weight* of the learn classifier α_t

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_t}{\varepsilon_t} \right)$$

Keep in mind that the idea is to learn a hypothesis H_T that be expressed as a linear combination of weak learners h_t. The better the weak learner, the greater the weight.

The remaining of the procedure consist in finding a good reweighting function of the training samples such that, during the next round, the new classifier h_{t+1} is able to correct the mistakes done by the classifier h_t. This is done using the following update rule:

$$w_i^{(t+1)} = w_i^{(t)} \frac{\exp(-\alpha_t y_i h_t(\mathbf{x}_i))}{Z_t}.,$$

where Z_t is normalization factor which ensure that the sum of the weights is equal to 1. We can show (see later for the proof) that $Z_t = 2\sqrt{\varepsilon_t (1 - \varepsilon_t)}$. The reweighting function used will then tend to give more weight to the examples for which the classifier is strongly wrong, and, on the contrary, to decrease the weight of the examples that are

correctly classified.

Proposition 8.2: Theoretical bound Adaboost

The empirical error of the classifier returned by Adaboost verifies :

$$\mathcal{R}_S(H_T) \leq \exp\left[-2\sum_{t=1}^{T}\left(\frac{1}{2}-\varepsilon\right)^2\right].$$

Furthermore, if for all $t \in [\![1,T]\!]$, $\gamma \leq \left(\frac{1}{2}-\varepsilon_t\right)$, then:

$$\mathcal{R}_S(H_T) \leq \exp(-2\gamma^2 T).$$

Proof. First, we recall that the exponential function is an upper bound of the indicator function, *i.e.*

$$\forall(\mathbf{x},y)\ \mathbb{1}_{\{yh(\mathbf{x})<0\}} \leq \exp(-yh(\mathbf{x})).$$

We can then upper the empirical risk $\mathcal{R}_S(H_T)$:

$$\begin{aligned}\mathcal{R}_S(H_T) &= \frac{1}{m}\sum_{i=1}^{m}\mathbb{1}_{\{y_i h(\mathbf{x}_i)<0\}},\\ &\leq \frac{1}{m}\sum_{i=1}^{m}\exp(-y_i H_T(\mathbf{x}_i)).\end{aligned}$$

We can now express this last sum using the normalization factor Z_t and the weights $w_i^{(t+1)}$. Indeed, we have:

$$\begin{aligned}w_i^{(t+1)} &= w_i^{(t)}\frac{\exp(-\alpha_t y_i h_t(\mathbf{x}_i))}{Z_t},\\ &\downarrow \text{ by recursion}\\ &= \frac{1}{m}\times\frac{\exp(-\sum_{s=1}^{t}\alpha_s h_s(x_i))}{\prod_{s=1}^{t}Z_s}\end{aligned}$$

Thus, the empirical risk can be upper bounded as:

$$\mathcal{R}_S(H_T) \leq \frac{1}{m}\sum_{i=1}^{m}\exp(-y_i H_T(\mathbf{x}_i)),$$

$$
\begin{aligned}
&\downarrow \text{ using the definition of } H_T \text{ and the previous recursion} \\
&\leq \frac{1}{m}\sum_{i=1}^{m}\left(m\prod_{t=1}^{T}Z_t\right)w_i^{(T+1)}, \\
&\leq \prod_{t=1}^{T}Z_t.
\end{aligned}
$$

We will now focus on the normalization factor and see how we can write it as a function of the classification error ε_t for all $t \in [\![1, T]\!]$.

$$
\begin{aligned}
Z_t &= \sum_{i=1}^{m} w_i^{(t)}) \exp(\breve{}\alpha_t y_i h_t(\mathbf{x}_i)), \\
&= \sum_{i:y_i h_t(\mathbf{x}_i)=1} w_i^{(t)} \exp(-\alpha_t) + \sum_{i:y_i h_t(\mathbf{x}_i)=-1} w_i^{(t)} \exp(\alpha_t), \\
&= (1-\varepsilon_t)\exp(-\alpha_t) + \varepsilon_t \exp(\alpha_t), \\
&= (1-\varepsilon_t)\sqrt{\frac{\varepsilon_t}{1-\varepsilon_t}} + \varepsilon_t\sqrt{\frac{1-\varepsilon_t}{\varepsilon_t}}, \\
&= 2\sqrt{\varepsilon_t(1-\varepsilon_t)}.
\end{aligned}
$$

As the empirical risk $\mathcal{R}_S(H_T)$ is upper bounded by the product of the normalization factors, we directly have:

$$
\begin{aligned}
\mathcal{R}_S(H_T) &\leq \prod_{t=1}^{T} Z_t, \\
&\leq \prod_{t=1}^{T} 2\sqrt{\varepsilon_t(1-\varepsilon_t)}, \\
&\downarrow \text{ using } 4\varepsilon_t(1-\varepsilon_t) = 1 - 4\left(\frac{1}{2}-\varepsilon_t\right)^2 \\
&\leq \prod_{t=1}^{T} \sqrt{1-4\left(\frac{1}{2}-\varepsilon_t\right)^2}, \\
&\downarrow \text{ for all } x \in \mathbb{R}\ 1-x \leq \exp(-x) \\
&\leq \prod_{t=1}^{T} \exp\left[-2\left(\frac{1}{2}-\varepsilon_t\right)^2\right], \\
&\downarrow \text{ property of the exponential} \\
&\leq \exp\left[-2\sum_{t=1}^{T}\left(\frac{1}{2}-\varepsilon_t\right)^2\right].
\end{aligned}
$$

This ends the first part of the proof. Furthermore, if for all t, $\gamma \leq \left(\frac{1}{2} - \varepsilon_t\right)$:

$$\begin{aligned}
\mathcal{R}_S(H_T) &\leq \exp\left[-2\sum_{t=1}^{T}\left(\frac{1}{2}-\varepsilon_t\right)^2\right], \\
&\quad \downarrow \text{ using the assumption} \\
&\leq \exp\left[-2\sum_{t=1}^{T}\gamma^2\right], \\
&\leq \exp\left[-2T\gamma^2\right].
\end{aligned}$$

□

We did not explain previously where the expression of α_t comes from, but the answer is in the previous proof. Indeed, it is chosen to minimize the upper bound of the empirical error. Thus it is chosen to minimize the function:

$$\varphi : \alpha \mapsto (1-\varepsilon_t)\exp(-\alpha) + \varepsilon_t \exp(\alpha).$$

This function is convex as a convex combination of two convex functions. So, it reaches its minimum for a single value α_t which verifies Euler's Equation:

$$\begin{aligned}
\nabla\varphi(\alpha_t) &= 0, \\
-(1-\varepsilon_t)\exp(-\alpha_t) + \varepsilon_t\exp(\alpha_t) &= 0, \\
(1-\varepsilon_t)\exp(-\alpha_t) &= \varepsilon_t\exp(\alpha_t), \\
\frac{1-\varepsilon_t}{\varepsilon_t} &= \varepsilon_t\exp(2\alpha_t), \\
\alpha_t &= \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\varepsilon_t\right).
\end{aligned}$$

Note also that the proof is available not only for a binary output h but also for type of hypothesis for which the output is in the range $[-1, +1]$.

Link with Gradient Descent We can draw the link between the algorithm Adaboost and a gradient descent algorithm called *coordinate descent*, we just have to change our point of view on the algorithm.

Let us consider a set of base learners $h_1, h_2, \ldots, h_T$ and a function H_T which will depend on vector $\boldsymbol{\alpha}$ and defined by

$$H_T(\boldsymbol{\alpha}) = \sum_{t=1}^{T} \alpha_t h_T$$

Let us also consider a set of m labeled examples $S = \{(\mathbf{x}_i, y_i)\}$. We aim to solve the following optimization problem:

$$\min_{\alpha \in \mathbb{R}^T} \frac{1}{m} \exp(-y_i H_T(\mathbf{x}_i)) = \min_{\alpha \in \mathbb{R}^T} \frac{1}{m} \sum_{i=1}^{m} \exp\left(-y_i \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}_i)\right).$$

We name this objective function F and we see that is exactly the upper bound of the empirical risk of H_T. It is exactly the function minimized by Adaboost. F is also a convex function of α as the sum of convex functions.

The idea of the coordinate descent is to update only one coordinate of the vector we aim to find at each round the gradient descent algorithm. More precisely, starting from a vector $\boldsymbol{\alpha}^{(k)}$ at a given round k of the gradient descent:

$$\boldsymbol{\alpha}^{(k)} = \left(\alpha_1^{(k)}, \alpha_2^{(k)}, \ldots, \alpha_i^{(k)}, \ldots, \alpha_T^{(k)}\right)$$

We change only one coordinate, let us say the i-th coordinate, to this vector to get a new vector $\boldsymbol{\alpha}^{(k+1)}$, at round $k+1$. So that:

$$\forall j \neq t, \quad \alpha_j^{(k+1)} = \alpha_j^{(k)} \quad \text{and} \quad \alpha_t^{(k+1)} = \alpha_t^{(k)} + \rho \times d_l,$$

where ρ is a learning rate and d_l is selected direction or coordinate that is updated.

If we consider the function g_k defined at iteration k by $g_k = \sum_{t=1}^{T} \alpha_t^{(k+1)} h_t$. Then the coordinate descent update coincides with the update $g_{k+1} = g_k + \rho h_l$. Thus, since both algorithms start with $g_0 = 0$, to show that Adaboost coincides with coordinate descent applied to F, it suffices to show at every iteration k, coordinate descent selects the same base hypothesis h_t and step ρ as Adaboost.

We will assume by induction that this holds up to iteration $k-1$, which implies the equality $g_{k-1} = H_T^{(k-1)}$, and will show then that it also holds at iteration k.

Here, at reach round k, we will select the maximum descent direction, that is the direction d_l along which the derivative of the objective function f is the largest in absolute value, and of selecting the best step along that direction, that is of choosing ρ to minimize $F(\boldsymbol{\alpha}^{(k-1)} + \rho d_k)$.

We first need to introduce similar quantities as for the boosting algorithm. We will denote by $\tilde{w}_i^{(k)}$ the weight of the i-th example at iteration k:

$$\tilde{w}_i^{(k)} = \frac{\exp\left(-y_i \sum_{t=1}^{T} \alpha_t^{(k-1)} h_t(\mathbf{x}_i)\right)}{\tilde{Z}_k} = \frac{\exp\left(-y_i g_{k-1}(\mathbf{x}_i)\right)}{\tilde{Z}_k},$$

where $\tilde{Z}_k$ is a normalization factor, such the sum of the weights is equal to 1. Since $g_{k-1} = H_T^{(k-1)}$, we have $\mathbf{w}^{(k)} = \tilde{\mathbf{w}}^{(k)}$.

We can also define the expected error of a base learner h_t with respect to the distribution $\tilde{\mathbf{w}}^{(k)}$ as:

$$\tilde{\varepsilon}_t^{(k)} = \mathop{\mathbb{E}}_{\tilde{\mathbf{w}}^{(k)}} \left[\mathbb{1}_{\{y_i h_t(\mathbf{x}_i)\}}\right].$$

The directional derivative of F at $\boldsymbol{\alpha}^{(k-1)}$ along d_k is denoted by $F'(\boldsymbol{\alpha}^{(k-1)}, d_l)$ and defined by

$$F'(\boldsymbol{\alpha}^{(k-1)}, d_l) = \lim_{\rho \to 0} \frac{F(\boldsymbol{\alpha}^{(k-1)} + \rho d_l) - F(\boldsymbol{\alpha}^{(k-1)})}{\rho}.$$

Since $F(\boldsymbol{\alpha}^{(k-1)} + \rho d_l) = \frac{1}{m} \sum_{i=1}^{m} \exp\left(-y_i \sum_{t=1}^{T} \alpha_t^{(k-1)} h_t(\mathbf{x}_i) - \rho y_i h_l(\mathbf{x}_i)\right)$, the directional derivative can be expressed:

$$\begin{aligned}
F'(\boldsymbol{\alpha}^{(k-1)}, d_l) &= -\frac{1}{m} \sum_{i=1}^{m} y_i h_l(\mathbf{x}_i) \exp(-y_i \sum_{t=1}^{T} \alpha_t^{(k-1)} h_t(\mathbf{x}_i), \\
&\quad \downarrow \text{ we use the relation between } Z_k \text{ and the weights } \tilde{w}_i^{(k)} \\
&= -\frac{1}{m} \sum_{i=1}^{m} y_i h_l(\mathbf{x}_i) \tilde{w}_i^{(k)} \tilde{Z}_k, \\
&\quad \downarrow \text{ we separate the sum according to the sign of } y_i h_l(\mathbf{x}_i) \\
&= \frac{\tilde{Z}_k}{m} \left[\sum_{i: y_i h_l(\mathbf{x}_i) = 1} \tilde{w}_i^{(k)} - \sum_{i: y_i h_l(\mathbf{x}_i) = -1} \tilde{w}_i^{(k)} \right], \\
&\quad \downarrow \text{ using the definition of the error} \\
&= \frac{\tilde{Z}_k}{m} \left[1 - \tilde{\varepsilon}_l^{(k)} - \tilde{\varepsilon}_l^{(k)}\right], \\
&= (2\tilde{\varepsilon}_l^{(k)} - 1) \frac{\tilde{Z}_k}{m}.
\end{aligned}$$

The selected direction is then the one that minimizes $\tilde{\varepsilon}_l^{(k)}$. So the base learner h_l selected at round k is the one with the smallest expected error on the sample S with respect to distribution $\tilde{\mathbf{w}}^{(k)} = \mathbf{w}^{(k)}$ on the sample. This is exactly the choice made by

Adaboost at the l-th round.

It remains to see how the step size is computed along the chosen direction d_l. the step size ρ is the solution of the following convex optimization problem

$$\min_{\rho} F(\boldsymbol{\alpha}^{(k-1)} + \rho d_l).$$

To minimize this quantity, it is enough to look for the value of ρ where its gradient vanishes.

$$\begin{aligned}
\nabla_\eta F(\boldsymbol{\alpha}^{(k-1)} + \rho d_l) &= 0, \\
-\frac{1}{m}\sum_{i=1}^{m} y_i h_l(\mathbf{x}_i) \exp\left(-y_i \sum_{t=1}^{T} \alpha_t^{(k-1)} h_t(\mathbf{x}_i) - \rho y_i h_l(\mathbf{x}_i)\right) &= 0, \\
\sum_{i=1}^{m} y_i h_l(\mathbf{x}_i) \tilde{w}_i^{(k)} \tilde{Z}_k \exp\left(-\rho y_i h_l(\mathbf{x}_i)\right) &= 0, \\
\sum_{i=1}^{m} y_i h_l(\mathbf{x}_i) \tilde{w}_i^{(k)} \exp\left(-\rho y_i h_l(\mathbf{x}_i)\right) &= 0, \\
(1 - \tilde{\varepsilon}_l^{(k)}) \exp(-\rho) - \tilde{\varepsilon}_l^{(k)} \exp(\rho) &= 0, \\
\rho &= \frac{1}{2} \ln\left(\frac{1 - \tilde{\varepsilon}_l^{(k)}}{\tilde{\varepsilon}_l^{(k)}}\right).
\end{aligned}$$

This proves that the step size chosen by coordinate descent coincides with the weight α_k assigned by AdaBoost to the classifier chosen in the k-th round. Thus, coordinate descent applied to exponential objective F precisely coincides with Adaboost and F can be viewed as the objective function that Adaboost seeks to minimize.

8.3 Gradient Boosting

The Adaboost algorithm is based on the exponential loss, however, such a loss is not suited for all settings and it is sometimes better to use other losses depending on the model you want to learn or for a specific application. This motivation leads us to the presentation of a more general boosting algorithm, the *Gradient Boosting* [Friedman, 2000].

Generalities Unlike the well-known Adaboost algorithm [Freund and Schapire, 1999], gradient boosting performs an optimization in the *function* space rather than in the

Algorithm 6: Gradient Boosting [Friedman, 2000]

Inputs : Training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, number of models T, a loss ℓ

Output: a model $H_T = H_0 + \sum_{t=1}^T \alpha_t h_{a^t}(\mathbf{x})$

1: Initial hypothesis $H_0 \quad H^0(\mathbf{x}_i) = \arg\min_{\rho \in \mathbb{R}} \sum_{i=1}^m \ell(y_i, \rho)\ \forall i = 1, \ldots, m$

2: **for** $t = 1$ to T **do**

3: Compute pseudo-residuals: $\tilde{y}_i = -\dfrac{\partial \ell(y_i, H_{t-1}(\mathbf{x}_i))}{\partial H_{t-1}(\mathbf{x}_i)},\ \forall i = 1, ..., m$

4: Fit the residuals: $a_t = \arg\min_{a \in \mathbb{R}^d} \sum_{i=1}^m (\tilde{y}_i - h_a(\mathbf{x}_i))^2$ to learn the new hypothesis

5: Learn the weight of the classifier h_{a^t}:
$\alpha^t = \arg\min_{a \in \mathbb{R}^+} \sum_{i=1}^m \ell(y_i, H_{t-1}(\mathbf{x}_i) + \alpha h_{a^t}(\mathbf{x}_i))$

6: Update $H_t(\mathbf{x}_i) = H_{t-1}(\mathbf{x}_i) + \alpha^t h_{a^t}(\mathbf{x}_i)$

7: **end for**=0

parameter space. At each iteration, a weak learner h_t is learned using the *residuals* (or the errors) obtained by the linear combination of the previous models. The linear combination H_t at time t is defined as follows:

$$H_t = H_{t-1} + \alpha_t h_t \tag{10}$$

where H_{t-1} is the linear combination of the first $t-1$ models and α_t is the weight given to the t-th weak learner. The weak learners are trained on the residuals $r_i = y_i - H_{t-1}(\mathbf{x}_i)$ of the current model. These residuals are given by the negative gradient, $-g_t$, of the used loss function L with respect to the current prediction $H_{t-1}(\mathbf{x}_i)$:

$$r_i = g_t(\mathbf{x}_i) = -\left[\frac{\partial \ell(y_i, H_{t-1}(\mathbf{x}_i))}{\partial H_{t-1}(\mathbf{x}_i)}\right].$$

Once the residuals r_i are computed, the following optimization problem is solved:

$$(h_t, \alpha_t) = \arg\min_{\alpha, h} \sum_{i=1}^m (r_i - \alpha h(\mathbf{x}_i))^2.$$

Finally, the update rule (10) is applied.

This algorithm has been first developed for classification and regression trees, and most of the work and libraries such as **XGBoost** [Chen and Guestrin, 2016] are using decision trees as weak learners. To be considered as weak learners, the learners mainly consist in decision stumps or tree with a small depth. Gradient boosting, on the contrary of Adaboost allows to use custom losses. The procedure is summarized in Algorithm 6.

Exemple 8.2. *Let us show what are the pseudo residuals for two different losses ℓ: the square loss for a regression task and the logistic loss for classification task.*

Using the square loss ℓ, the pseudo residuals are defined by:

$$\begin{aligned}\tilde{y} &= -\frac{\partial \ell(y, H_{t-1}(\mathbf{x}))}{\partial H_{t-1}(\mathbf{x})}, \\ &= -\frac{\partial (y - H_{t-1}(\mathbf{x}))^2}{\partial H_{t-1}(\mathbf{x})}, \\ &= 2(y - H_{t-1}(\mathbf{x})).\end{aligned}$$

The pseudo residual is just twice the difference between the true value and the predicted value.

Using the logistic loss ℓ, the pseudo residuals are defined by:

$$\begin{aligned}\tilde{y} &= -\frac{\partial \ell(y, H_{t-1}(\mathbf{x}))}{\partial H_{t-1}(\mathbf{x})}, \\ &= -\frac{\partial \ln\left(1 + \exp\left(-2yH_{t-1}(\mathbf{x})\right)\right)}{\partial H_{t-1}(\mathbf{x})}, \\ &=\end{aligned}$$

We can also draw a parallel between the gradient boosting algorithm and a gradient descent algorithm. This is more natural since the algorithm involves the gradient. Here, the parallel can be drawn with the gradient descent with optimal steepest descent (or optimal step) as shown by Algorithm 7.

Note that the Gradient boosting, in its original presentation, is based on the use of the mean squared error at the step consisting in fitting the pseudo-residuals. However, we can choose any loss function we want as the exponential one for classification tasks for instance. Furthermore, the reader has to be careful the loss ℓ we aim to optimize is the same as the one used to find the weights of the base learners.

Presentation of XGBoost We will finish this presentation on boosting by returning to our decision trees and presenting XGBoost [Chen and Guestrin, 2016] which is a powerful and very fast boosting algorithm based on trees. We first explain how the weights are computed for each leaf, then we explain the splitting criterion that is used.

Algorithm 7: Gradient Boosting [Friedman, 2000]

Inputs : Training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$, number of models T, a loss ℓ

Output: a model $H_T = H_0 + \sum_{t=1}^T \alpha_t h_{a^t}(\mathbf{x})$

1: Initialization H_0 $\quad H^0(\mathbf{x}_i) = \arg\min_{\rho \in \mathbb{R}} \sum_{i=1}^m \ell(y_i, \rho)\ \forall i = 1, \ldots, m$

2: **for** $t = 1$ to T **do**

3: Compute the gradient: $\tilde{y}_i = -\dfrac{\partial \ell(y_i, H_{t-1}(\mathbf{x}_i))}{\partial H_{t-1}(\mathbf{x}_i)},\ \forall i = 1, ..., m$

4: Learn a model that approximates this gradient $a_t = \arg\min_{a \in \mathbb{R}^d} \sum_{i=1}^m (\tilde{y}_i - h_a(\mathbf{x}_i))^2$ to learn the new hypothesis

5: Find the optimal weight h_{a^t} : $\alpha^t = \arg\min_{a \in \mathbb{R}^+} \sum_{i=1}^m \ell(y_i, H_{t-1}(\mathbf{x}_i) + \alpha h_{a^t}(\mathbf{x}_i))$

6: Update $H_t(\mathbf{x}_i) = H_{t-1}(\mathbf{x}_i) + \alpha^t h_{a^t}(\mathbf{x}_i)$

7: **end for**=0

We consider a loss ℓ (there is no particular assumption on it) and the following optimization problem:

$$\min \sum_{i=1}^m \ell(y_i, \hat{y}_i) + \beta\mathcal{L} + \frac{\lambda}{2}\sum_{j=1}^{\mathcal{L}}(f_j^{(t)})^2. \tag{11}$$

where $\beta\mathcal{L}$ and $\dfrac{\lambda}{2}\sum_{j=1}^{\mathcal{L}}(h_t^{(j)})^2$ are two regularization terms used to control the number of leaves and the weight of each leaf $f_t^{(j)}$ for the learned tree at iteration t.

We recall that the models are learned in an additive manner, so let us denote $\hat{y}^{(t-1)}$, the predicted value by the first $t-1$ functions h_k, i.e. $\hat{y}_i^{(t-1)} = \sum_{k=1}^{t-1} h_t(\mathbf{x}_i) = H_{t-1}(\mathbf{x}_i)$. Let us now study how the next model is learned. For this purpose, we rewrite the quantity (11) to minimize as follows:

$$\sum_{i=1}^m \ell(y_i, \hat{y}_i^{(t-1)} + h_t(\mathbf{x}_i)) + \beta\mathcal{L} + \frac{\lambda}{2}\sum_{j=1}^{\mathcal{L}}(h_t^{(j)})^2. \tag{12}$$

We only use the additive definition of the model.

In practice, [Chen and Guestrin, 2016] only consider a second order approximation of the function they aim to optimize. This second order approximation is done with respect to the predicted value at the previous iteration, i.e. $\hat{y}_i^{(t-1)}$. We will denote by respectively g and f the first and second order derivatives of the function L with respect to $\hat{y}^{(t-1)}$. We can rewrite (12) as follows:

$$\sum_{i=1}^{m}\left[L(y_i,\hat{y}_i^{(t-1)})+h_t(\mathbf{x}_i)g(\mathbf{x}_i)+\frac{1}{2}h_t^2(\mathbf{x}_i)f(\mathbf{x}_i)\right]+\beta\mathcal{L}+\frac{\lambda}{2}\sum_{j=1}^{\mathcal{L}}(f_t^{(j)})^2. \tag{13}$$

Remember that we aim to learn the function $\mathbf{f}_t=(h_t^{(j)})_{j=1,\dots,\mathcal{L}}$. So let us consider a leaf j and denote by I_j the set of index i such that $\mathbf{x}_i$ falls in the leaf l_j. Thus, using (13), the function $h_t^{(j)}$ shall minimize the following quantity V_j for a given index j:

$$V_j \;=\; \sum_{i\in I_j}\left[g(\mathbf{x}_i)h_t^{(j)}(\mathbf{x}_i)+\frac{1}{2}\left(\lambda+f(\mathbf{x}_i)\right)\left(h_t^{(j)}(\mathbf{x}_i)\right)^2\right]. \tag{14}$$

This function is convex and the minimum is given by the solution of *Euler's equation*, i.e. the function $f^{(t)}$ for which the gradient vanishes. This solution is given by:

$$h_t^{(j)}=-\frac{\sum_{i\in I_j}g(\mathbf{x}_i)}{\sum_{i\in I_j}f(\mathbf{x}_i)+\lambda}. \tag{15}$$

Exemple 8.3. *Let us come back to our previous example where we have considered the square loss and the logistic loss and let us see what the weights of the leaves are when we use these two losses.*

For the square loss $\ell(\hat{y}^{(t-1)})=\frac{1}{2}(y_i-(\hat{y}^{(t-1)})^2$. *The gradient* g *with respect to the prediction is*

$$g(\hat{y}^{(t-1)})=\frac{\partial\ell}{\partial\hat{y}^{(t-1)}}(\hat{y}^{(t-1)})=(\hat{y}^{(t-1)}-y).$$

And the second order derivative f *with respect to the prediction is*

$$f(\hat{y}^{(t-1)})=\frac{\partial^2\ell}{\partial(\hat{y}^{(t-1)})^2}(\hat{y}^{(t-1)})=1.$$

So the optimal value of a leaf $h_t^{(j)}$ *is:*

$$h_t^{(j)}=-\frac{\sum_{i\in I_j}g(\mathbf{x}_i)}{\sum_{i\in I_j}f(\mathbf{x}_i)+\lambda}=\frac{\sum_{i\in I_j}(y_i-\hat{y}^{(t-1)})}{\lambda+|I_j|}.$$

Thus, when learning the first tree, with the assumption $\hat{y}_i^{(0)}=0$ *for all* i*, we find that the optimal score in a leaf is equal to the average of the instances values in the leaf. For subsequent iterations, the optimal score of each leaf becomes the average of the*

pseudo-residuals.

Let us now focus on the logistic loss $\ell(\hat{y}^{(t-1)}) = -(y \ln(p(\hat{y}^{(t-1)})) + (1-y)\ln(1 - p(\hat{y}^{(t-1)}))$*, where* p *is the logistic function.*

The gradient g *with respect to the prediction is*

$$g(\hat{y}^{(t-1)}) = \frac{\partial \ell}{\partial \hat{y}^{(t-1)}}(\hat{y}^{(t-1)}) = (p(\hat{y}^{(t-1)}) - y).$$

And the second order derivative f *with respect to the prediction is*

$$f(\hat{y}^{(t-1)}) = \frac{\partial^2 \ell}{\partial (\hat{y}^{(t-1)})^2}(\hat{y}^{(t-1)}) = p(\hat{y}^{(t-1)})(1 - p(\hat{y}^{(t-1)})).$$

So the optimal value of a leaf $h_t^{(j)}$ *is:*

$$h_t^{(j)} = -\frac{\sum_{i \in I_j} g(\mathbf{x}_i)}{\sum_{i \in I_j} f(\mathbf{x}_i) + \lambda} = \frac{\sum_{i \in I_j} (p(\hat{y}^{(t-1)}) - y)}{\lambda + \sum_{i \in I_j} p(\hat{y}^{(t-1)})(1 - p(\hat{y}^{(t-1)}))}.$$

Let us now focus on the splitting criterion. Once the optimal weight is found for each leaf (15), we can compute the optimal value $V_j^\star$ of the loss by using (14), we get:

$$\begin{aligned} V_j^\star &= \sum_{i \in I_j} \left[\underbrace{- g(\mathbf{x}_i) \frac{\sum_{i \in I_j} g(\mathbf{x}_i)}{\sum_{i \in I_j} f(\mathbf{x}_i) + \lambda}}_{} + \underbrace{\frac{1}{2}(\lambda + f(\mathbf{x}_i)) \left(- \frac{\sum_{i \in I_j} g(\mathbf{x}_i)}{\sum_{i \in I_j} f(\mathbf{x}_i) + \lambda} \right)^2}_{} \right], \\ &= - \frac{\left(\sum_{i \in I_j} g(\mathbf{x}_i) \right)^2}{\sum_{i \in I_j} f(\mathbf{x}_i) + \lambda} + \frac{1}{2} \frac{\left(\sum_{i \in I_j} g(\mathbf{x}_i) \right)^2}{\sum_{i \in I_j} f(\mathbf{x}_i) + \lambda}, \\ V_j^\star &= - \frac{1}{2} \frac{\left(\sum_{i \in I_j} g(\mathbf{x}_i) \right)^2}{\sum_{i \in I_j} f(\mathbf{x}_i) + \lambda}. \end{aligned}$$

This formula is used to measure the quality of a leaf. It can be seen as a generalized formula for Gini Impurity for any loss function. Using this new measure, they define their splitting criterion, i.e. the gain associated to a split, as follows:

$$\frac{1}{2} \left[\frac{\left(\sum_{i \in I_L} g(\mathbf{x}_i) \right)^2}{\sum_{i \in I_L} f(\mathbf{x}_i) + \lambda} + \frac{\left(\sum_{i \in I_R} g(\mathbf{x}_i) \right)^2}{\sum_{i \in I_R} f(\mathbf{x}_i) + \lambda} - \frac{\left(\sum_{i \in I} g(\mathbf{x}_i) \right)^2}{\sum_{i \in I} f(\mathbf{x}_i) + \lambda} \right] - \beta,$$

where $I = I_L \cup I_R$ for a binary tree and the parameter β is used to control the number of leaves.

8.4 Metric Learning

See later, maybe next year.

9 Applications and Learning in Practice

References

[Bahnsen et al., 2014] Bahnsen, A. C., Stojanovic, A., Aouada, D., and Ottersten, B. (2014). *Improving Credit Card Fraud Detection with Calibrated Probabilities*, pages 677–685.

[Bartlett and Mendelson, 2003] Bartlett, P. L. and Mendelson, S. (2003). Rademacher and gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 3:463–482.

[Behl et al., 2014] Behl, A., Jawahar, C., and Pawan Kumar, M. (2014). Optimizing average precision using weakly supervised data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1011–1018.

[Bentley, 1975] Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517.

[Bolton and Hand, 2002] Bolton, R. J. and Hand, D. J. (2002). Statistical fraud detection: A review. *Statistical science*, pages 235–249.

[Boser et al., 1992] Boser, B. E., Guyon, I. M., and Vapnik, V. N. (1992). A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, COLT '92, pages 144–152. ACM.

[Bousquet et al., 2004] Bousquet, O., Boucheron, S., and Lugosi, G. (2004). *Introduction to Statistical Learning Theory*, pages 169–207. Springer Berlin Heidelberg.

[Bousquet and Elisseeff, 2002] Bousquet, O. and Elisseeff, A. (2002). Stability and generalization. *Journal of Machine Learning Research*, 2:499–526.

[Boyd and Vandenberghe, 2004] Boyd, S. and Vandenberghe, L. (2004). *Convex Optimization*. Cambridge University Press, New York, NY, USA.

[Branco et al., 2016] Branco, P., Torgo, L., and Ribeiro, R. P. (2016). A survey of predictive modeling on imbalanced domains. *ACM Comput. Surv.*, 49(2):31:1–31:50.

[Breiman, 2001] Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

[Breiman et al., 1984] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification and regression trees*. The Wadsworth statistics/probability series. Wadsworth and Brooks/Cole Advanced Books and Software, Monterey, CA.

[Busa-Fekete et al., 2015] Busa-Fekete, R., Szörényi, B., Dembczynski, K., and Hüllermeier, E. (2015). Online f-measure optimization. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 595–603. Curran Associates, Inc.

[Chapelle, 2007] Chapelle, O. (2007). Training a support vector machine in the primal. *Neural Computation*, 19:1155–1178.

[Chen and Guestrin, 2016] Chen, T. and Guestrin, C. (2016). Xgboost: A scalable tree boosting system. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794. ACM.

[Cohen et al., 2006] Cohen, G., Hilario, M., Sax, H., Hugonnet, S., and Geissbuhler, A. (2006). Learning from imbalanced data in surveillance of nosocomial infection. *Artificial intelligence in medicine*, 37(1):7–18.

[Cortes and Mohri, 2004] Cortes, C. and Mohri, M. (2004). Auc optimization vs. error rate minimization. In *Advances in neural information processing systems*, pages 313–320.

[Cover and Hart, 1967] Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27.

[Cox, 1958] Cox, D. R. (1958). The regression analysis of binary sequences (with discussion). *Journal of the Royal Statistical Society*, 20:215–242.

[Cramer, 2003] Cramer, J. (2003). The origins of logistic regression. *SSRN Electronic Journal*.

[Dudani, 1976] Dudani, S. A. (1976). The distance-weighted k-nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-6(4):325–327.

[Elkan, 2003] Elkan, C. (2003). Using the triangle inequality to accelerate k-means. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, pages 147–153. AAAI Press.

[Ferri et al., 2009] Ferri, C., Hernández-Orallo, J., and Modroiu, R. (2009). An experimental comparison of performance measures for classification. *Pattern Recognition Letters*, 30(1):27–38.

[Frery et al., 2017] Frery, J., Habrard, A., Sebban, M., Caelen, O., and He-Guelton, L. (2017). Efficient top rank optimization with gradient boosting for supervised anomaly detection. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 20–35. Springer.

[Freund and Schapire, 1999] Freund, Y. and Schapire, R. E. (1999). A short introduction to boosting. In *In Proceedings of the Sixteenth IJCAI*, pages 1401–1406. Morgan Kaufmann.

[Friedman et al., 2000] Friedman, J., Hastie, T., and Tibshirani, R. (2000). Additive logistic regression: a statistical view of boosting. *The Annals of Statistics*, 28(2):337–407.

[Friedman, 2000] Friedman, J. H. (2000). Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232.

[Gee, 2014] Gee, S. (2014). *Fraud and fraud detection: a data analytics approach*. John Wiley & Sons.

[Genton, 2002] Genton, M. G. (2002). Classes of kernels for machine learning: A statistics perspective. *Journal of Machine Learning Research*, 2:299–312.

[Hart, 1968] Hart, P. (1968). The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, 14(3):515–516.

[He and Wang, 2008] He, Q. P. and Wang, J. (2008). Principal component based k-nearest-neighbor rule for semiconductor process fault detection. In *2008 American Control Conference*, pages 1606–1611.

[Hoeffding, 1963] Hoeffding, W. (1963). Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 58(301):13–30.

[Jeni et al., 2013] Jeni, L. A., Cohn, J. F., and De La Torre, F. (2013). Facing imbalanced data–recommendations for the use of performance metrics. In *Affective Computing and Intelligent Interaction (ACII), 2013 Humaine Association Conference on*, pages 245–251. IEEE.

[Jensen, 1906] Jensen, J. L. W. V. (1906). Sur les fonctions convexes et les inégalités entre les valeurs moyennes. *Acta Mathematica*, 30:175–193.

[Koltchinskii and Panchenko, 2000] Koltchinskii, V. and Panchenko, D. (2000). Rademacher processes and bounding the risk of function learning. In Giné, E., Mason, D. M., and Wellner, J. A., editors, *High Dimensional Probability II*, pages 443–457. Birkhäuser Boston.

[Konukoglu and Ganz, 2014] Konukoglu, E. and Ganz, M. (2014). Approximate false positive rate control in selection frequency for random forest. *CoRR*, abs/1410.2838.

[Liu and Chawla, 2011] Liu, W. and Chawla, S. (2011). Class confidence weighted knn algorithms for imbalanced data sets. In Huang, J. Z., Cao, L., and Srivastava, J., editors, *Advances in Knowledge Discovery and Data Mining*, pages 345–356, Berlin, Heidelberg. Springer Berlin Heidelberg.

[McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133.

[McDiarmid, 1989] McDiarmid, C. (1989). *On the method of bounded differences*, pages 148–188. London Mathematical Society Lecture Note Series. Cambridge University Press.

[Mercer, 1909] Mercer, J. (1909). Functions of positive and negative type, and their connection with the theory of integral equations. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 209(441-458):415–446.

[Metz, 1978] Metz, C. E. (1978). Basic principles of roc analysis. In *Seminars in nuclear medicine*, volume 8, pages 283–298. Elsevier.

[Mitchell, 1997] Mitchell, T. M. (1997). *Machine learning*, volume 1.

[Mohri et al., 2012] Mohri, M., Rostamizadeh, A., and Talwalkar, A. (2012). *Foundations of Machine Learning*. The MIT Press.

[Quinlan, 1986] Quinlan, J. R. (1986). Induction of decision trees. *Mach. Learn.*, 1(1):81–106.

[Rijsbergen, 1979] Rijsbergen, C. J. V. (1979). *Information Retrieval*. Butterworth-Heinemann, Newton, MA, USA, 2nd edition.

[Rokach and Maimon, 2005] Rokach, L. and Maimon, O. (2005). Top-down induction of decision trees classifiers - a survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 35(4):476–487.

[Rosenblatt, 1958] Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386.

[Safavian and Landgrebe, 1991] Safavian, S. R. and Landgrebe, D. (1991). A survey of decision tree classifier methodology. *IEEE Transactions on Systems, Man, and Cybernetics*, 21(3):660–674.

[Sanderson, 1994] Sanderson, M. (1994). Word sense disambiguation and information retrieval. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '94, pages 142–151, New York, NY, USA. Springer-Verlag New York, Inc.

[Torgo and Lopes, 2011] Torgo, L. and Lopes, E. (2011). Utility-based fraud detection. In *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, pages 1517–1522.

[Torgo and Ribeiro, 2007] Torgo, L. and Ribeiro, R. (2007). Utility-based regression. In *Knowledge Discovery in Databases: PKDD 2007*, pages 597–604. Springer Berlin Heidelberg.

[Valiant, 1984] Valiant, L. G. (1984). A theory of the learnable. *Communication of the ACM*, 27(11):1134–1142.

[Vapnik and Chervonenkis, 1971] Vapnik, V. and Chervonenkis, A. (1971). On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability & Its Applications*, 16(2):264–280.

[Vapnik and Chervonenkis, 1982] Vapnik, V. and Chervonenkis, A. (1982). Necessary and sufficient conditions for the uniform convergence of means to their expectations. *Theory of Probability & Its Applications*, 26(3):532–553.

[Vapnik and Cortes, 1995] Vapnik, V. and Cortes, C. (1995). Support-vector networks. *Machine Learning*, 20:273–297.

[Vapnik, 1995] Vapnik, V. N. (1995). *The Nature of Statistical Learning Theory*. Springer-Verlag, Berlin, Heidelberg.

[von Luxburg and Bousquet, 2004] von Luxburg, U. and Bousquet, O. (2004). Distance-based classification with lipschitz functions. *J. Mach. Learn. Res.*, 5(Jun):669–695.

[Weinberger and Saul, 2009] Weinberger, K. Q. and Saul, L. K. (2009). Distance metric learning for large margin nearest neighbor classification. *Journal of Machine Learning Research*, 10:207–244.

[Zhao et al., 2013] Zhao, M.-J., Edakunni, N. U., Pocock, A. C., and Brown, G. (2013). Beyond fano's inequality: bounds on the optimal f-score, ber, and cost-sensitive risk and their implications. *Journal of Machine Learning Research*, 14:1033–1090.

A Some Results in Probability

This section aims to provide some results in probability, mainly concentration inequalities or some important results.

Théorème A.1: Law of large Numbers; Kolmogorov, 1929

Let us consider $(X_n)_{n\in\mathbb{N}^\star}$ a sequence of independant and identically distributed random variables and $S_n = \sum_{i=1}^n X_i$ Suppose that all of them have a moment of order 1, *i.e.* for all $n \in \mathbb{N}^\star, \mathbb{E}[X_n] < +\infty$.

Then, the random variable S_n/n converges almost surely to $\mathbb{E}[X_1]$

Proposition A.1: Hoeffding Inequality

Let us consider $(X_n)_{n\in\mathbb{N}^\star}$ a sequence of independant, identically distributed and centered (*i.e.* with expectation equal to 0) random variables. Let us also suppose that for all n, X_n is almost surely bounded by a positive constant c_n. We also note $\forall n \in \mathbb{N}^\star,\ S_n = \sum_{i=1}^n X_i$ and $a_n = \sum_{j=1}^n c_j^2$.

Then $\forall \varepsilon > 0,\ \forall n \in \mathbb{N}^\star,\ \mathbb{P}[S_n > \varepsilon] \leq 2\exp\left(\frac{-\varepsilon^2}{2a_n}\right)$.