Mathematics for Machine Learning

NYU Paris, CSCI-UA 9473

Augustin Cosse

Draft, May 26, 2019

These notes provide a summary of the mathematics needed for an introductory class in machine learning. The version is temporary. Please direct any comments or questions to acosse@nyu.edu or acosse@ens.fr

1 Notations

$\mathbb{R}, \mathbb{Z}, \mathbb{N},$	Real, integer and natural numbers
$\langle \cdot, \cdot \rangle$	Inner product, $\langle \boldsymbol{v}, \boldsymbol{w} \rangle = \sum_{i=1}^{n} v_i w_i$, $\langle \boldsymbol{A}, \boldsymbol{B} \rangle = \sum_{i=1}^{n} \sum_{j=1}^{m} A_{ij} B_{ij}$
$\text{Tr}(\boldsymbol{A})$	Trace of a matrix $\text{Tr}(\boldsymbol{A}) = \sum_{i=1}^{n} A_{ii}$
$\mathcal{N}(\mu, \sigma^2)$	Normal distribution with mean μ, variance σ^2 (see below)
$\boldsymbol{\theta}$	vector encoding the parameters of a distribution, e.g. for the Gaussian, $\boldsymbol{\theta} = (\mu, \sigma^2)$
$A \propto B$	A Proportional to B. We will use this symbol often when discussing inference. For example we will often write $p(\theta\|x) \propto P(x\|\theta)p(\theta)$ where we neglect the normalizing constant $p(x)$.
$\delta(x = k)$	Indicator function, $\delta(x = k) = 1$ if $x = k$ and 0 otherwise
$\mathcal{D}$	Usually used to represent a dataset, $\mathcal{D} = \{\boldsymbol{x}_i, y_i\}_{i=1}^{N}$
$\phi(\boldsymbol{x})$	For a given prototype $\boldsymbol{x}$, we usually reserve the notation $\phi(\boldsymbol{x})$ to denote the corresponding feature vector generated from $\boldsymbol{x}$
$\frac{\partial f}{\partial x_i}$	partial derivative of $f(x_1, \ldots, x_N)$ with respect to the variable x_i
$\nabla f(\boldsymbol{x})$	gradient, $\nabla f(\boldsymbol{x}) = (\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n})$
$\boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T$	Eigenvalue decomposition, $\boldsymbol{\Lambda} = \text{diag}(\lambda_1, \ldots, \lambda_N)$
$\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$	Singular value decomposition $\boldsymbol{\Sigma} = \text{diag}(\sigma_1, \ldots, \sigma_N)$
$\mathcal{H}(p)$	Shannon Entropy

Table 1: Notations

1.1 Notions from optimization

When discussing optimization, we will often make use of the notations max, min as well as argmax and argmin. When looking for the minimum *value*, f^*, of a function, we write

$$f^* = \min_x \ f(x) \tag{1}$$

When looking for *the point* x^* at which the function achieves a minimal value (minimal argument), we write

$$x^* = \underset{x}{\text{argmin}} \ f(x) \tag{2}$$

When looking for the solution of an optimization problem (such as the computation of the MLE or MAP) below, a common approach is to use a gradient descent iteration,

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \alpha \nabla f(\boldsymbol{x}_k) \tag{3}$$

By moving in the direction of $-\nabla f(\boldsymbol{x}_k)$, provided that we use a sufficiently small step length (or learning rate) α, we are guaranteed to decrease the value of the function at each step. Whenever the Hessian is available and not too expensive, one can turn to second order methods, the most popular of which is Newton's method, whose iterations are defined as

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \left(\nabla^2 f(\boldsymbol{x}_k)\right)^{-1} \nabla f(\boldsymbol{x}_k) \tag{4}$$

The use of the Hessian enables faster convergence (we can think of second order methods as methods that are using a more global view of the optimization landscape than gradient descent steps) but induces a higher computational cost. In practice we therefore turn to approximations of this Hessian. The corresponding optimization methods are then called quasi-Newton methods. The iterations corresponding to those methods are defined as

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k - \boldsymbol{B}_k^{-1} \nabla f(\boldsymbol{x}_k) \tag{5}$$

where $\boldsymbol{B}_k$ is an approximation of the Hessian which is updated at each step. The most popular quasi Newton method is the BFGS algorithm (see for example [17], chapter 8).

1.2 Statistical Inference

For a given probability model p parametrized by the parameter θ which takes values in $\{\theta_1, \theta_2, \ldots, \theta_N\}$, Bayes's Theorem (see below) gives

$$p(\theta|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\theta)p(\theta)}{p(\boldsymbol{x})} \tag{6}$$

$p(\theta|\boldsymbol{x})$ is called the posterior, $p(\boldsymbol{x}|\theta)$ is called likelihood. Finally $p(\theta)$ is called the prior.

Contents

2 Linear algebra

Given a matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$,

$$\boldsymbol{A} = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \ddots & & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix} \tag{7}$$

we define its transpose as the matrix $\boldsymbol{A}^T \in \mathbb{R}^{n \times m}$ whose j^{th} column is defined as the j^{th} row of $\boldsymbol{A}$, i.e.

$$\boldsymbol{A}^T = \begin{pmatrix} a_{11} & a_{21} & \dots & a_{m1} \\ a_{12} & a_{22} & \dots & a_{n2} \\ \vdots & \ddots & & \\ a_{1n} & \dots & a_{m-1,m} & a_{mn} \end{pmatrix} \tag{8}$$

If $\boldsymbol{A}$ and $\boldsymbol{B}$ are $m \times n$ matrices, we have the following properties

- $(\boldsymbol{A}^T)^T = \boldsymbol{A}$
- $(\boldsymbol{A} + \boldsymbol{B})^T = \boldsymbol{A}^T + \boldsymbol{B}^T$
- $(\alpha \boldsymbol{A})^T = \alpha \boldsymbol{A}^T$
- $(\boldsymbol{A}\boldsymbol{B})^T = \boldsymbol{B}^T \boldsymbol{A}^T$

Definition 1 (PseudoInverse)**.** *Given a matrix $\boldsymbol{A}$, the pseudo inverse is defined as the matrix $\boldsymbol{A}^+$ that satisfies the following conditions*

$$\boldsymbol{A}\boldsymbol{A}^+\boldsymbol{A} = \boldsymbol{A} \tag{9}$$
$$\boldsymbol{A}^+\boldsymbol{A}\boldsymbol{A}^+ = \boldsymbol{A}^+ \tag{10}$$
$$(\boldsymbol{A}\boldsymbol{A}^+)^T = \boldsymbol{A}\boldsymbol{A}^+ \tag{11}$$
$$(\boldsymbol{A}^+\boldsymbol{A})^T = \boldsymbol{A}^+\boldsymbol{A} \tag{12}$$

When the matrix $\boldsymbol{A}^T\boldsymbol{A}$ is invertible, the pseudo inverse can be defined as $(\boldsymbol{A}^T\boldsymbol{A})^{-1}\boldsymbol{A}^T$

2.1 Norms and inner products

Given two vectors $\boldsymbol{v} \in \mathbb{R}^n$ and $\boldsymbol{w} \in \mathbb{R}^n$, one can define an inner product as

$$\boldsymbol{v} \cdot \boldsymbol{w} = \langle \boldsymbol{v}, \boldsymbol{w} \rangle = \sum_{i=1}^{n} v_i w_i \tag{13}$$

Generally speaking, a function $\langle \cdot, \cdot \rangle$ is an inner product if it satisfies the following properties

- $\langle x, x \rangle \geq 0$, $\langle x, x \rangle = 0 \iff x = 0$ (positivity)
- $\langle x, y \rangle = \langle y, x \rangle$ (symmetry)
- $\langle x + y, z \rangle = \langle x, z \rangle + \langle y, z \rangle$ (additivity)
- $\langle \alpha x, y \rangle = \alpha \langle x, y \rangle$, for all $\alpha \in \mathbb{R}$ (homogeneity)

So far, we have discussed inner products for vectors, but it is also possible to define an inner product on matrices. A standard inner product can be defined in a similar way on matrices $\boldsymbol{X}, \boldsymbol{Y} \in \mathbb{R}^{m \times n}$,

$$\langle \boldsymbol{X}, \boldsymbol{Y} \rangle = \mathrm{Tr}(\boldsymbol{X}^T \boldsymbol{Y}) = \sum_{i=1}^{m} \sum_{j=1}^{n} X_{ij} Y_{ij} \tag{14}$$

Here we used the notation $\mathrm{Tr}(\boldsymbol{A})$ to denote the trace of a square matrix $\boldsymbol{A} \in \mathbb{R}^{n \times n}$, i.e. $\mathrm{Tr}(\boldsymbol{X}) = \sum_{i=1}^{n} A_{ii}$.

Another important function is the norm. A norm is a function $f : \mathbb{R}^n \mapsto \mathbb{R}$ and which has the following properties

- $f(\boldsymbol{x}) \geq 0$, $f(\boldsymbol{x}) = 0 \iff \boldsymbol{x} = 0$ (positivity)
- $f(\alpha \boldsymbol{x}) = |\alpha| f(\boldsymbol{x}), \quad \forall \alpha \in \mathbb{R}$ (homogeneity)
- $f(\boldsymbol{x} + \boldsymbol{y}) \leq f(\boldsymbol{x}) + f(\boldsymbol{y})$ (triangle inequality)

Examples of norms include

- The ℓ_2 norm: $\|\boldsymbol{v}\|_2 = \sqrt{\sum_{i=1}^{n} v_i^2}$
- The ℓ_1 norm: $\|\boldsymbol{v}\|_1 = \sum_{i=1}^{n} |v_i|$
- The ℓ_∞ norm: $\|\boldsymbol{v}\|_\infty = \max_i |v_i|$
- More generally, the ℓ_p norms are defined for $p \geq 1$ as $\|\boldsymbol{v}\|_p = \left(\sum_{i=1}^{n} |v_i|^p\right)^{1/p}$

Just as for vectors, one can define norms on matrices. Those norms will satisfy the same properties. The most popular one is the Frobenius norm

$$\|\boldsymbol{A}\|_F = \|\boldsymbol{A}\|_2 = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} |A_{i,j}|^2} = \sqrt{\mathrm{Tr}(\boldsymbol{A}^T \boldsymbol{A})} \tag{15}$$

In fact given any inner product (on vectors or matrices), one can define an "induced" norm by letting $f(\boldsymbol{v}) = \|\boldsymbol{v}\| = \sqrt{\langle \boldsymbol{v}, \boldsymbol{v} \rangle}$

3 Differential Calculus

Given a multivariate scalar function $f(\boldsymbol{x})$, $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$, one defines the gradient of the function as the vector encoding the derivatives of the function f with respect to each of the components of $\boldsymbol{x}$, i.e.,

$$\nabla f(\boldsymbol{x}) = \frac{\partial f}{\partial \boldsymbol{x}} = \left(\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n} \right) \tag{16}$$

Note that when we differentiate a field or vector function, we get a matrix, whose entries correspond to the partial derivatives of each of the components of the field with respect to each of the variables. This matrix is known as the Jacobian matrix. For $\vec{f}(\boldsymbol{x}) = (f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n))$, we have

$$\frac{\partial \vec{f}}{\partial \boldsymbol{x}} \equiv \left(\frac{\partial f_i(\boldsymbol{x})}{\partial x_j} \right)_{ij} = J = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \ddots & & \vdots \\ \frac{\partial f_m}{\partial x_1} & \frac{\partial f_m}{\partial x_2} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix} \tag{17}$$

The Hessian matrix of a multivariate function $f : \mathbb{R}^n \mapsto \mathbb{R}$, is the matrix encoding all second order partial derivatives of f,

$$\nabla^2 f = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \ddots & & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix} \tag{18}$$

Compactly we can write

$$H = \nabla^2 f = \left(\frac{\partial^2 f}{\partial x_i \partial x_j} \right)_{ij} \tag{19}$$

3.1 Matrix differentiation

When considering the derivative of a function $f(\boldsymbol{A})$, $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ with respect to any matrix $\boldsymbol{A}$, we will use the abridged notation $\frac{\partial f(\boldsymbol{A})}{\partial \boldsymbol{A}}$ to denote the $m \times n$ matrix encoding the first order (partial) derivatives of the function f with respect to each of the entries of the matrix $\boldsymbol{A}$, i.e.

$$\frac{\partial f(\boldsymbol{A})}{\partial \boldsymbol{A}} = \begin{pmatrix} \frac{\partial f(\boldsymbol{A})}{\partial a_{11}} & \frac{\partial f(\boldsymbol{A})}{\partial a_{12}} & \cdots & \frac{\partial f(\boldsymbol{A})}{\partial a_{1m}} \\ \frac{\partial f(\boldsymbol{A})}{\partial a_{21}} & \frac{\partial f(\boldsymbol{A})}{\partial a_{2,2}} & \cdots & \frac{\partial f(\boldsymbol{A})}{\partial a_{2m}} \\ \frac{\partial f(\boldsymbol{A})}{\partial a_{31}} & & \ddots & \vdots \\ \frac{\partial f(\boldsymbol{A})}{\partial a_{m1}} & \cdots & \frac{\partial f(\boldsymbol{A})}{\partial a_{m,n-1}} & \frac{\partial f(\boldsymbol{A})}{\partial a_{mn}} \end{pmatrix} \tag{20}$$

4 Statistics and probability

4.1 Probability

Definition 2. *Given a sample space $\mathcal{S}$ and an associated algebra $\mathcal{B}$, a probability function is a function P with a domain $\mathcal{B}$ that satisfies*

- $P(A) \geq 0$ *for all* $A \in \mathcal{B}$
- $P(S) = 1$
- *If* $A_1, A_2, \ldots \in \mathcal{B}$ *are pairwise disjoint, then* $P(\cup_{i=1}^{\infty} A_i) = \sum_{i=1}^{\infty} P(A_i)$

Theorem 1. *If P is a probability function and A is any set in $\mathcal{B}$, then*

- $P(\emptyset) = 0$, *where* $\emptyset$ *is the empty set*
- $P(A) \leq 1$
- $P(A^c) = 1 - P(A)$

Theorem 2. *if P is a probability function and A and B are any sets in $\mathcal{B}$, then*

- $P(B \cap A^c) = P(B) - P(A \cap B)$
- $P(A \cup B) = P(A) + P(B) - P(A \cap B)$
- *If* $A \subset B$, *then* $P(A) \leq P(B)$.

Definition 3. *If A and B are events in S, and $P(B) > 0$, then the conditional probability of A given B, written $P(A|B)$ is*

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \tag{21}$$

Given definition (3), it is easy to see that $P(A \cap B) = P(A|B)P(B) = P(B \cap A) = P(B|A)P(A)$ (this idea is sometimes known as the chain rule). The generalization of this gives Bayes' rule:

Theorem 3 (Bayes' rule)**.** *Let $A_1, A_2, \ldots$ be a partition of a sample space, and let B be any set. Then, for each $i = 1, 2, \ldots$,*

$$P(A_i|B) = \frac{P(B|A_i)P(A_i)}{\sum_{j=1}^{\infty} P(B|A_j)P(A_j)} \tag{22}$$

(the denominator is simply the expansion $P(B) = \sum_{j=1}^{\infty} P(B|A_j)P(A_j)$)

Sometimes, it could happen that the occurence of an event B has no influence on the probability of another event A, i.e.

$$P(A|B) = P(A) \tag{23}$$

If this condition holds, using Bayes's Theorem, it immediately implies $P(B|A) = \frac{P(A|B)P(B)}{P(A)} = P(B)$. From this, we also have $P(A \cap B) = P(A)P(B)$. This last relation is the precise definition of statistical independence

Definition 4 (Independence)**.** *Two events A and B are statistically independent if*

$$P(A \cap B) = P(A)P(B) \tag{24}$$

Definition 5. *A collection of events $A_1, \ldots, A_n$ are mutually independent if for any subcollection $A_{i_1}, \ldots, A_{i_k}$, we have*

$$P\left(\bigcap_{j=1}^{k} A_{i_j}\right) = \prod_{j=1}^{k} P(A_{i_j}) \tag{25}$$

Theorem 4. *If two events A and B are independent, the following pairs of events are also independent*

- *A and B^c*
- *A^c and B*
- *A^c and B^c*

4.2 Random variables

Definition 6 (Random variable)**.** *A random variable is a function from a sample space into the real numbers*

It is important to understand that the set of random variables includes the samples themselves as one can of course choose the random variable to match a sample onto the value of this sample. However, the probability is only defined on the sample space. For example, a simple Gaussian random variable X is a function from the sample space $\mathcal{S} \equiv (-\infty, \infty)$ into this same sample space (or if you prefer the value of the sample). I.e. $X(s_i) = s_i$ in this case. But the definition is more general and one could for example define another random variable $Y = X^2$. Now the random variable is not the identity anymore. It a function which takes samples from the sample space $\mathcal{S}$ and raise them to the power 2. The connection to the sample space is important because the

notion of probability P is defined only on this sample space. I.e we only know how likely an event s_i is, we don't know about $X(s_i)$. It is however possible to define an induced probability on the random variable X, by using P,

$$P_X(X \in A) = P(\{s \in \mathcal{S} \ : \ X(s) \in A\}) \tag{26}$$

In words, we look at the samples s for which $X(s) \in A$ and we define the probability of the event $X \in A$ from the probability of the corresponding sample $\{s \in \mathcal{S} \ : \ X(s) \in A\}$.

We are now ready to introduce the notions of cumulative distribution function (cdf), probability mass function (pmf) and probability density function (pdf).

Definition 7. *The cumulative distribution function or cdf of a random variable X which we denote $F_X(x)$, is defined by*

$$F_X(x) = P(X \leq x), \quad \text{for all } x \tag{27}$$

Definition 8. *The probability mass function (pmf) of a discrete random variable X defined as*

$$f_X(x) = P(X = x), \quad \text{for all } x \tag{28}$$

Definition 9. *The probability density function or pdf, $f_X(x)$ of a continuous random variable X is the function defined as*

$$F_X(x) = \int_{-\infty}^{x} f_X(t)\, dt, \quad \text{for all } x \tag{29}$$

We now introduce the notion of expectation or expected value (a.k.a mean or average) of a random variable. The expected value $\mathbb{E}\{X\}$ of a random variable X can be thought of as an indication of a typical value that the random variable X will take, hence a measure of the "center" of the distribution.

Definition 10. *The expected value of a random variable $h(X)$, which we denote $\mathbb{E}\{h(X)\}$ is defined as*

$$\mathbb{E}h(X) = \begin{cases} \int_{-\infty}^{\infty} h(x) f_X(x)\, dx & \text{if } X \text{ is continuous} \\ \sum_{x \in \mathcal{X}} h(x) f_X(x) = \sum_{x \in \mathcal{X}} h(x) P(X = x) & \text{if } X \text{ is discrete} \end{cases} \tag{30}$$

In the definition above, $\mathcal{X}$ is used to denote the set of all possible outcomes of the random variable X.

Theorem 5. *If X is a random variable and α, β, γ are constants, then for any functions h_1 and h_2 with well defined expectations, we have*

- $\mathbb{E}\{\alpha h_1(X) + \beta h_2(X) + \gamma\} = \alpha \mathbb{E}\{h_1(X)\} + \beta \mathbb{E}\{h_2(X)\} + \gamma$
- *If $h_1(x) \geq 0$ for all x, then $\mathbb{E}\{h_1\} \geq 0$*
- *If $h_1(x) \geq h_2(x)$ for all x, then $\mathbb{E}\{h_1(X)\} \geq \mathbb{E}\{h_2(X)\}$*
- *If $\alpha \leq h_1(x) \leq \beta$ for all x, then $\alpha \leq \mathbb{E}\{h_1(X)\} \leq \beta$.*

A particular class of expectations can be defined for a random variable X by choosing the function $h_1(X)$ to encode the monomials X^α, $\alpha \in \mathbb{N}$. Those expectations are then called moments of the random variable X.

Definition 11. *For each integer $n \in \mathbb{N}$, the n^{th} moment of X, μ'_n is defined a*

$$\mu'_n = \mathbb{E}\{X^n\} \tag{31}$$

Accordingly, the n^{th} central moment of X, μ_n is defined as

$$\mu_n = \mathbb{E}\{(X - \mu)^n\} \tag{32}$$

In particular, note that we have $\mu = \mu'_1 = \mathbb{E}X$.

Definition 12. *The variance of a random variable X is its second central moment, i.e. $\sigma^2 = Var(X) = \mathbb{E}\left\{(X - \mathbb{E}\{X\})^2\right\}$. The (positive) square root of σ^2, σ is called standard deviation.*

For two random variables X and Y, one can also define the correlation and covariance of X and Y,

Definition 13. *The covariance of X and Y is the number defined by*

$$\mathrm{Cov}(X, Y) = \mathbb{E}\{(X - \mu_X)(Y - \mu_Y)\} \tag{33}$$

Definition 14. *The correlation or correlation coefficient of X and Y is the number defined by*

$$\rho_{XY} = \frac{\mathrm{Cov}(X, Y)}{\sigma_X \sigma_Y} \tag{34}$$

If similar values of X and Y occur at the same time, the covariance will be large and positive. If on the contrary, large positive values of X occur with large negative values for Y, the covariance $\mathrm{Cov}(X, Y)$ will be large and negative. Hence we can already say that the sign of $\mathrm{Cov}(X, Y)$ gives information on the type of relation that exists between X and Y.

The correlation coefficient $\rho_{X,Y}$ between X and Y takes value between -1 and 1 with a value of 1 indicating a perfect linear relationship between the two variables. Another important aspect of the correlation coefficient is its connection to statistical independence.

Theorem 6. *If two variables X and Y are independent, then $Cov(X, Y) = 0$ and $\rho_{X,Y} = 0$.*

Proof. The proof follows form the definition of independence and the fact that independence implies $\mathbb{E}\{XY\} = \mathbb{E}X\mathbb{E}Y$ □

4.3 Classical PDFs

We start by listing some important distributions:

- The Gaussian/Normal Distribution is the most popular distribution in statistics. The univariate distribution reads as

$$\mathcal{N}(x|\mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \exp(-\frac{1}{2\sigma^2}|x - \mu|^2) \tag{35}$$

  The multivariate distribution, for a d-dimensional vector $\boldsymbol{x}$, reads as

$$\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Sigma}|^{1/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{\mu})\right\} \tag{36}$$

- The Binomial distribution (which we write $\mathrm{Bin}(k|n, \boldsymbol{\theta})$) is used to represent the probability of observing the positive/successful outcomes (the probability of success is given by θ) that repeats itself k times over n successive trials.

$$\mathrm{Bin}(k|n, \theta) = \binom{n}{k} \theta^k (1 - \theta)^{n-k} \tag{37}$$

  The first factor encodes the number of possibilities of choosing k elements among the n without replacement.

$$\binom{n}{k} = \frac{n!}{(n - k)!k!} \tag{38}$$

  The mean and variance for this distribution are given by $\mu = n\theta$ and $\sigma^2 = n\theta(1 - \theta)$

- The Bernoulli distribution. The Bernoulli distribution is defined on a binary variable. I.e if we consider a single experiment (tossing a coin) with only two possible outcomes which are encoded by the variable $y = \{1, 0\}$ and appear with probability p and $1 - p$ respectively. to encode the probability of an outcome 0 or 1 we can thus just raise the probability θ to the power y, as show below. This gives the Bernoulli pdf

$$\text{Ber}(x|\theta) = \theta^y (1-\theta)^{1-y} \tag{39}$$
$$\tag{40}$$

  This is also sometimes written

$$\text{Ber}(x|\theta) = \theta^{\delta(y=0)} (1-\theta)^{\delta(y=1)} \tag{41}$$
$$\tag{42}$$

  Where $\delta(\mathcal{C}) = 1$ if the predicate $\mathcal{C}$ is satisfied.

- Categorical distribution or Multinoulli distribution. When considering a variable with K possible states (e.g. a dice with 6 faces), we will often encodes those states through dummy variables. This means that for example for a variable that can take three different values, $y = 1$, $y = 2$ or $y = 3$ (we will see later that this applies to features or classes in classification problems), we would encode each of these feature through the 3-tuples $\boldsymbol{t} = (1, 0, 0)$ $(y = 1)$, $\boldsymbol{t} = (0, 1, 0)$ $(y = 2)$ and $\boldsymbol{t} = (0, 0, 1)$ $(y = 3)$. If we use $\theta_j = p_j$, $j = 1, \ldots, 3$ to denote the probability to get each feature, the total probability density function can now read compactly as

$$\text{Cat}(x|\boldsymbol{\theta}) = \text{Mu}(\boldsymbol{x}|\theta) = p(x|\theta) = \prod_{j=1}^{K} \theta_j^{t_j} = \theta_1^{t_1} \ldots \theta_K^{t_K} = \boldsymbol{\theta^t} \tag{43}$$

- The Laplace distribution is characterized by "fatter" tails than the Gaussian distribution (i.e the probability of getting values that are very different from the mean will be higher in the Laplace distribution). The pdf of the Laplace distribution reads as

$$p(x|\mu, \beta) = \frac{1}{2\beta} \exp(-\frac{|x-\mu|}{\beta}) \tag{44}$$

  one way to see this is that in the Laplace distribution the argument of the exponential is the absolute value whether in the Normal distribution, the argument is squared which implies a faster decrease in the pdf for large values of the deviation to the mean, $|x - \mu|$.

4.4 A few important Theorems

Theorem 7 (Weak Law of large numbers)**.** *Let $X_1, X_2, \ldots$ be identically and independently distributed (i.i.d) random variables with $\mathbb{E}X_i = \mu$ and $\text{Var}X_i = \sigma^2$. Define $\overline{X}_n = (1/n)\sum_{i=1}^n X_i$. Then, for every $\varepsilon > 0$,*

$$\lim_{n\to\infty} P(|\overline{X}_n - \mu| < \varepsilon) = 1 \tag{45}$$

That is, $\overline{X}_n$ converges in probability to μ.

Theorem 8 (Strong Law of large numbers)**.** *Let $X_1, X_2, \ldots$ be i.i.d. random variables with $\mathbb{E}X_i = \mu$ and $\text{Var}X_i = \sigma^2 < \infty$ and define $\overline{X}_n = (1/n)\sum_{i=1}^n X_i$. Then for every $\epsilon > 0$,*

$$P(\lim_{n\to\infty} |\overline{X}_n - \mu| < \varepsilon) = 1 \tag{46}$$

that is $\overline{X}_n$ converges almost surely to μ

Theorem 9 (Central Limit Theorem). *Let $X_1, X_2, \ldots$ be a sequence of i.i.d random variables whose mgf's exist in a neighborhood of 0 (that is $M_{X_i}(t)$ exists for $t < |h|$, for some positive h). Let $\mathbb{E}X_i = \mu$ and $\mathrm{Var}X_i = \sigma^2$ (both μ and σ^2 are finite since the mgf exists). Define the empirical mean $\overline{X}_n$ as $\overline{X}_n = (1/n)\sum_{i=1}^n X_i$. Let $G_n(x)$ denote the cdf of $\sqrt{n}(\overline{X}_n - \mu)/\sigma$. Then for any x, $-\infty < x < \infty$,*

$$\lim_{n\to\infty} G_n(x) = \int_{-\infty}^{x} \frac{1}{\sqrt{2\pi}} e^{-y^2/2}\, dy \tag{47}$$

That is, $\sqrt{n}(\overline{X}_n - \mu)/\sigma$ has a limiting standard normal distribution.

4.5 MAP and MLE

This part gives a very basic introduction to the difference between the Frequentist and Bayesian notions of estimators. For more details see [4]

Definition 15. *A point estimator is any function of the sample $W(x_1, \ldots, x_n)$*

The most popular method to derive an estimator on the parameters of a distribution is the method of maximum likelihood. For a given probability distribution $p(x_i|\theta_1, \ldots, \theta_k)$ parametrized by $\boldsymbol{\theta}$, the likelihood function is defined as

$$L(\theta|\boldsymbol{x}) = p(\boldsymbol{x}|\theta) = \prod_{i=1}^{n} p(x_i|\theta_1, \ldots, \theta_n) \tag{48}$$

This function encodes the "likelihood" of observing the samples x_i for a particular choice of parameters θ (we write $L(\theta|\boldsymbol{x})$ because we view it as a function of θ). Given this function, it seems reasonable to look for the parameters $\theta_1, \ldots, \theta_n$ which give the highest probability of observing the sample (i.e the distribution which gives the highest probability of observing the x_i is likely to be the distribution of those x_i). This idea leads to the Maximum likelihood estimator (MLE) whose formal definition is given below

Definition 16. *For each sample point $\boldsymbol{x}$, let $\hat{\theta}(\boldsymbol{x})$ be a parameter value at which $L(\theta|\boldsymbol{x})$ attains its maximum as a function of θ, with $\boldsymbol{x}$ held fixed. A maximum likelihood estimator (MLE) of the parameter θ based on a sample X is $\hat{\theta}(X)$. Mathematically, we write*

$$\theta_{MLE} = \underset{\theta}{\mathrm{argmax}}\; p(\boldsymbol{x}|\theta) \tag{49}$$

The MLE is considered a Frequentist estimator as it only relies on the sample itself without making any prior assumption on the parameters.

The Bayesian approach to statistics is fundamentally different. So far, we assumed that θ was an unknown but fixed quantity. I.e. A random sample $X_1, \ldots, X_n$ was drawn from a population and based on those observations, some knowledge on θ was obtained. In the Bayesian approach, θ is described by a probability distribution (the prior distribution). We thus assume some level of uncertainty on θ. The prior is a subjective distribution which is set by the observer.

The idea is that the observer updates his original prior based on the sample that he draws from the population. The update is done by means of Bayes' rule,

$$p(\theta|\boldsymbol{x}) \propto p(\boldsymbol{x}|\boldsymbol{\theta})p(\theta) \tag{50}$$

We can then estimate the parameter θ by finding the value which maximizes the new prior, $p(\theta|\boldsymbol{x})$. This idea is known as Maximum a posteriori (MAP). The MAP estimator is thus defined as

$$\theta_{\mathrm{MAP}} = \underset{\theta}{\mathrm{argmax}}\; p(\boldsymbol{x}|\theta)p(\theta) \propto p(\theta|\boldsymbol{x}) \tag{51}$$

4.5.1 Linear Gaussian system

By linear Gaussian system, we mean here that we consider both a Gaussian likelihood and a Gaussian prior, and connect the two by means of any linear transformation,

$$p(\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\theta}|\boldsymbol{\mu}_\theta, \boldsymbol{\Sigma}_\theta) \tag{52}$$

$$p(\boldsymbol{x}|\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{A}\boldsymbol{\theta} + \boldsymbol{b}, \boldsymbol{\Sigma}_x) \tag{53}$$

If we write down the probability of $\boldsymbol{x}$, from such a model, we get

$$p(\boldsymbol{x}) = \mathcal{N}(\boldsymbol{x}|\boldsymbol{A}\boldsymbol{\mu}_\theta + \boldsymbol{b}, \boldsymbol{\Sigma}_x + \boldsymbol{A}\boldsymbol{\Sigma}_\theta \boldsymbol{A}^T) \tag{54}$$

4.6 Exponential family

Many of the probability density functions discussed above are part of a general family of distributions known as the exponential family.

A probability density function, $f(\boldsymbol{x}|\boldsymbol{\theta})$ is said to be in the exponential family if it is of the form

$$f(\boldsymbol{x}|\boldsymbol{\theta}) = h(\boldsymbol{x})c(\boldsymbol{\theta}) \exp\left(\sum_{i=1}^{k} w_i(\boldsymbol{\theta})t_i(\boldsymbol{x})\right) \tag{55}$$

$$= \frac{1}{Z(\theta)} h(\boldsymbol{x}) \exp(\boldsymbol{\theta}^T \phi(\boldsymbol{x})) \tag{56}$$

$$= h(\boldsymbol{x}) \exp(\boldsymbol{\theta}^T \phi(\boldsymbol{x}) - A(\boldsymbol{\theta})) \tag{57}$$

where

$$Z(\boldsymbol{\theta}) = \int_{\mathcal{X}^m} h(\boldsymbol{x}) \exp(\boldsymbol{\theta}^T \phi(\boldsymbol{x}))\, d\boldsymbol{x} \tag{58}$$

$$A(\boldsymbol{\theta}) = \log Z(\boldsymbol{\theta}) \tag{59}$$

$Z(\boldsymbol{\theta})$ is called the partition function and $A(\boldsymbol{\theta})$ is called the log partition function. $\boldsymbol{\theta}$ are called the natural parameters or canonical parameters.

5 Probabilistic classifiers

When discussing classification, we often either look at classifiers as geometric objects (i.e separating planes, maximal margin hyperplanes,..) or as probabilistic objects based on Bayes Theorem. In the last framework, we consdider two main groups of classifiers

- Generative classifiers
- Discriminative classifiers

The first class defines a model for the joint probability distribution $p(\boldsymbol{x}, y)$ (or equivalently for the class conditional density $p(\boldsymbol{x}|y)$) and hence provides a way to generate new samples $\boldsymbol{x}$. The second class defines a model for the probability distribution $p(y|\boldsymbol{x})$ (i.e the class posterior) and hence only provides a way to discriminate between classes. Below we discuss the two most popular discriminative models: The Naive Bayes classifier (generative) and the Logistic regression classifier (discriminative)

5.1 Naive Bayes classifier

The Naive Bayes classifier is a generative classifier (i.e we learn a model for $p(\boldsymbol{x}|y)$ or $p(\boldsymbol{x}, y)$). We consider a classification problem in which the feature vectors are D dimensional. Moreover,

each of the D features can take K values. I.e, $\boldsymbol{x} \in \{1, \ldots, K\}^D$. If we assume that the features are independent within a class c (note that this is usually not the case), we can write the "class conditional density" as

$$p(\boldsymbol{x}|y = \mathcal{C}_\ell, \boldsymbol{\theta}) = \prod_{j=1}^{D} p(x_j|y = \mathcal{C}_\ell, \boldsymbol{\theta}_{j\ell}) \tag{60}$$

In the equation above $\theta_{j\ell}$ is the parameter associated to feature j in class $\mathcal{C}_\ell$ (see below).

Possible choices for the probability distribution include

- the Gaussian distribution

$$p(\boldsymbol{x}|y = \mathcal{C}_\ell; \boldsymbol{\theta}) = \prod_{j=1}^{D} \mathcal{N}(x_j|\mu_{j\ell}, \sigma^2_{j,\ell}) \tag{61}$$

  Here $\mu_{j,\ell}$ and $\sigma^2_{j,\ell}$ are the mean and variance for the j^{th} feature in class $\mathcal{C}_\ell$.

- When the features are encoded through binary variables $\beta = \{0, 1\}$, the multivariate Bernoulli distribution (or Bernoulli product model) is often used. In this case, we have

$$p(\boldsymbol{x}|y = \mathcal{C}_\ell, \boldsymbol{\theta}) = \prod_{j=1}^{D} \mathrm{Ber}(x_j|\mu_{j,\ell}) \tag{62}$$

  Again, here $\mu_{j,\ell}$ is the probability of getting feature j in class ℓ.

5.1.1 Deriving the parameters through Maximum Likelihood

We want to train a classification algorithm from a set of pairs $(\boldsymbol{x}_i, c_i)$. The prototypes $\boldsymbol{x}_i$ are represented by D-dimensional feature vectors $x_i = (x_{i1}, \ldots, x_{iD})$. From this, if we assume that the probabilities of observing the features are independent, the probability of observing a given pair $(\boldsymbol{x}_i, c_i)$ reads as

$$p(\boldsymbol{x}_i, c_i|\boldsymbol{\theta}) = \prod_{j=1}^{D} p(x_{ij}|c_i, \boldsymbol{\theta}_j) p(c_i|\boldsymbol{\pi}) \tag{63}$$

Here $p(c_i|\boldsymbol{\pi})$ encodes the probability of observing the class i, $p(c_i|\boldsymbol{\pi}) = (\pi_1^{\delta(c_i=1)}, \ldots, \pi_N^{\delta(c_i=N)})$ encode the probabilities of getting a particular class $\mathcal{C}_1$ to $\mathcal{C}_C$ (we use C to denote the number of classes). Using the δ notation, we can thus write (63) as

$$p(\boldsymbol{x}_i, c_i|\boldsymbol{\theta}) = \prod_{c=1}^{C} \pi_c^{\delta(c_i=c)} \prod_{j=1}^{D} \prod_{c=1}^{C} p(x_{ij}|\boldsymbol{\theta}_{c,j})^{\delta(y_i=c)} \tag{64}$$

Taking the product over all pairs $(x_i, c_i) \in \mathcal{D}$ and then the log, the log likelihood can read as

$$\log(\mathcal{D}|\boldsymbol{\theta}) = \sum_{c=1}^{C} N_c \log(\pi_c) + \sum_{j=1}^{D} \sum_{c=1}^{C} \sum_{i \text{ s.t.} y_i = c} \log p(x_{ij}|\boldsymbol{\theta}_{jc}) \tag{65}$$

Maximizing the log likelihood with respect to the parameters of the model then gives

$$\hat{\pi}_c = \frac{N_c}{N} \tag{66}$$

as an estimate for the class prior (the probability of being in Class c). This estimate is thus the ratio between the number of samples in class c and the total number of training samples.

If we assume that all N features $x_{1i}, \ldots, x_{1,N}$ are binary and that all the vectors within a class follow the same distribution, then a good choice for $p(x_{ij}|\theta_{jc})$ is to take a Bernoulli distribution with parameter θ_{jc}. In this case we can show the MLE for the parameter θ_{jc} is given by the ratio of the number of vectors from class c which have their j^{th} feature equal to 1 over the total number of points from class c,

$$\hat{\theta}_{jc} = \hat{p}(\boldsymbol{x}_{ij} = 1|y = c) = \frac{N_{jc}}{N_c} \tag{67}$$

Once we have computed the parameters of the model, this model can be used to classify new points by relying again on Bayes (neglecting the normalizing constant), we have

$$p(y = c|\boldsymbol{x}, \mathcal{D}) \propto p(y = c|\mathcal{D}) \prod_{j=1}^{D} p(x_j|y = c, \mathcal{D}) \tag{68}$$

$$\propto p(y = c|\mathcal{D}) p(\boldsymbol{x}|y = c, \mathcal{D}) \tag{69}$$

5.2 Logistic regression

Logistic regression is perhaps the most popular discriminative classifier. For data pairs $\{\boldsymbol{x}, y\}$ of prototypes and their associated labels, the probabilistic model is defined as

$$p(y|\boldsymbol{x}, \boldsymbol{w}) = \text{Ber}(y|\text{sigm}(\boldsymbol{w}^T \boldsymbol{x})) \tag{70}$$

where the sigmoid sigm() is defined as

$$\text{sigm}(x) = \frac{1}{1 + \exp(-x)} \tag{71}$$

We thus have

$$p(y = 1|\boldsymbol{x}, \boldsymbol{w}) = \frac{\exp(\boldsymbol{w}^T \boldsymbol{x})}{1 + \exp(\boldsymbol{w}^T \boldsymbol{x})} \tag{72}$$

$$p(y = 0|\boldsymbol{x}, \boldsymbol{w}) = 1 - \frac{\exp(\boldsymbol{w}^T \boldsymbol{x})}{1 + \exp(\boldsymbol{w}^T \boldsymbol{x})} \tag{73}$$

The interesting aspect of logistic regression is that the ratio between the probability of each class (which can be used to do classification) is linear in the parameters of the model. You can indeed verify that

$$\log\left(\frac{p(y = 1|\boldsymbol{x}, \boldsymbol{w})}{p(y = 0|\boldsymbol{x}, \boldsymbol{w})}\right) = \boldsymbol{w}^T \boldsymbol{x} \tag{74}$$

This model in fact falls in the class of "generalized" linear models for that precise reason. This idea easily extends to multiple classes by introducing "log-odd ratios" between the probability of each class and the probability of the last class. For K classes, we have $K - 1$ such ratios (the last probability is fixed through $\sum_i p(\mathcal{C} = i|x_n) = 1$)

$$\log\left(\frac{P(\mathcal{C} = 1|x)}{P(\mathcal{C} = K|x)}\right) = \boldsymbol{w}_1^T \boldsymbol{x} \tag{75}$$

$$\log\left(\frac{P(\mathcal{C} = 1|x)}{P(\mathcal{C} = K|x)}\right) = \boldsymbol{w}_1^T \boldsymbol{x} \tag{76}$$

$$\vdots \tag{77}$$

$$\log\left(\frac{P(\mathcal{C} = K - 1|x)}{P(\mathcal{C} = K|x)}\right) = \boldsymbol{w}_{K-1}^T \boldsymbol{x} \tag{78}$$

The posterior class probabilities are then defined as

$$P(\mathcal{C} = k|\boldsymbol{x}) = \frac{\exp(\boldsymbol{w}_k^T \boldsymbol{x})}{1 + \sum_{\ell=1}^{K-1} \exp(\boldsymbol{w}_\ell^T \boldsymbol{x})}, \quad k = 1, \ldots, K-1 \tag{79}$$

$$P(\mathcal{C} = K|\boldsymbol{x}) = \frac{1}{1 + \sum_{\ell=1}^{K-1} \exp(\boldsymbol{w}_\ell^T \boldsymbol{x})} \tag{80}$$

The model (and the associated parameters) can be learned through Maximum likelihood, by writing the likelihood function for the N observation (noting c_n the class of $\boldsymbol{x}_n$),

$$\ell(\boldsymbol{w}_\ell, \ell = 1, \ldots, K-1) = \sum_{n=1}^{N} \log(p(\mathcal{C} = c_n|\boldsymbol{x})) \tag{81}$$

And then minimizing the function (for example through gradient descent).

6 Optimization

In this section, we discuss some additional details in optimization. Finding the global minimizer of sufficiently complex functions is usually hard because iterative algorithms will get trapped in "local minima". When solving optimization problems, we will often be interested in understanding whether the particular point to which our algorithm converges is a local or a global minimum (resp maximum) of the function. Under some conditions (smoothness of the function), the general characterization of the local extremas of a function is relatively easy as shown by the following two propositions

Proposition 1 (First order necessary conditions)**.** *If $\boldsymbol{x}^*$ is a local minimizer and f is continuously differentiable in an open neighborhood of $\boldsymbol{x}^*$, then $\nabla f(\boldsymbol{x}^*) = 0$.*

Proposition 2 (Second Order Necessary conditions)**.** *If x^* is a local minimizer of f and $\nabla^2 f$ is continuous in an open neighborhood of x^*, then $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive semidefinite.*

Given a function f, there also exist sufficient conditions under which one can guarantee that a point is a minimizer. This idea is summarized by proposition 3 below

Proposition 3 (Second Order Sufficient conditions)**.** *Suppose that $\nabla^2 f$ is continuous in an open neighborhood of x^* and that $\nabla f(x^*) = 0$ and $\nabla^2 f(x^*)$ is positive definite. Then x^* is a strict local minimizer of f.*

6.1 Convexity

Definition 17 (Convex set)**.** *A set of points, $\mathcal{C}$ is convex if the line segment between any two points in the set is included in the set, that is to say for any $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathcal{C}$, and any scalar $\alpha \in \mathbb{R}$ with $0 \leq \alpha \leq 1$, we have*

$$\alpha \boldsymbol{x}_1 + (1 - \alpha)\boldsymbol{x}_2 \in \mathcal{C} \tag{82}$$

When a set is not convex, it is always possible to define the smallest convex set that contains $\mathcal{C}$. This idea in particular plays an important role when replacing non convex function (which therefore cannot be optimized efficiently) by convex approximation whose solution can be found and studied by iterative algorithms.

Definition 18 (convex hull)**.** *Given a set of points $\mathcal{S}$, the convex hull of $\mathcal{S}$ is defined as*

$$\text{conv}\mathcal{S} = \{\alpha_1 \boldsymbol{x}_1 + \ldots, \alpha_k \boldsymbol{x}_k | \boldsymbol{x}_i \in \mathcal{S}, \alpha_i \geq 0, i = 1, \ldots, k, \alpha_1 + \ldots + \alpha_k = 1\} \tag{83}$$

I.e., conv$\mathcal{S}$ *is the set defined from all possible convex combinations of points from $\mathcal{S}$.*

6.2 Constrained problems and Lagrangian

When considering a basic mathematical programming problem of the form

$$\begin{aligned} \min \quad & f_0(\boldsymbol{x}) \\ \text{subject to} \quad & f_\ell(\boldsymbol{x}) \leq 0, \ell = 1, \dots L. \end{aligned} \tag{84}$$

We will use K to denote the set of feasible points for the program (84) (i.e. the set of points $\boldsymbol{x}$ satisfying the constraints in (84)), i.e. $K = \{\boldsymbol{x} \mid f_\ell(\boldsymbol{x}) \leq 0\}$

We can write the Lagrangian by introducing positive multipliers $\lambda_\ell \in \mathbb{R}$ for each of the constraints $f_\ell(x) \leq 0$. As those constraints are negative, the Lagrangian then reads as

$$L = f_0(x) - \sum_{\ell=1}^{L} \lambda_\ell f_\ell(\boldsymbol{x}) \tag{85}$$

The following Theorem then gives necessary conditions (on the Lagrangian) for a point $\boldsymbol{x}$ to be a minimizer from the feasible set, K.

Theorem 10 (Kuhn-Tucker conditions)**.** *Assume that $f_\ell(\boldsymbol{x})$ $(\ell = 1, \dots, L)$ are all differentiable. If the function $f_0(\boldsymbol{x})$ attains a local minimum at a point $\boldsymbol{x}^*$ which belongs the feasible set K, then there exists a vector of multipliers, λ_ℓ^*, $\ell = 1, \dots, L$, such that the following conditions are satisfied*

$$\frac{\partial f_0(\boldsymbol{x}^*)}{\partial x_j} + \sum_{\ell=1}^{L} \lambda_\ell^* \frac{\partial f_\ell(\boldsymbol{x}^*)}{\partial x_j} = 0, \quad (j = 1, \dots, J) \tag{86}$$

$$f_\ell(\boldsymbol{x}^*) \leq 0, \quad (\ell = 1, \dots, L) \tag{87}$$

$$\lambda_\ell^* f_\ell(\boldsymbol{x}^*) = 0, \quad (\ell = 1, \dots, L) \tag{88}$$

$$\lambda_\ell^* \geq 0, \quad (\ell = 1, \dots, L) \tag{89}$$

Proof. We can always replace the inequality constraints in (84), by introducing slack variables y_ℓ. I.e., we always have the equivalence

$$f_\ell(\boldsymbol{x}) \leq 0 \iff f_\ell(\boldsymbol{x}) + y_\ell^2 = 0 \tag{90}$$

for some variables y_ℓ.

For those slack variables, the Lagrangian reads as

$$L = f_0(\boldsymbol{x}) + \sum_{\ell=1}^{L} \lambda_\ell (f_\ell(\boldsymbol{x}) + y_\ell^2) \tag{91}$$

Any minimum of the Lagrangian must satisfy the zero gradient condition, so we have

$$\frac{\partial L}{\partial x_j} = \frac{\partial f_0(\boldsymbol{x}^*)}{\partial x_j} + \sum_{\ell=1}^{L} \lambda_\ell \frac{\partial (f_\ell(\boldsymbol{x}^*) + (y_\ell^*)^2)}{\partial x_j} = 0, \quad (\ell = 1, \dots, L) \tag{92}$$

$$\frac{\partial L}{\partial y_\ell} = 2\lambda_\ell y_\ell = 0 \tag{93}$$

$$\frac{\partial L}{\partial \lambda_\ell} = f_\ell(\boldsymbol{x}) + y_\ell^2 = 0 \tag{94}$$

Condition (93) above is equivalent to the "complementary slackness" condition (88). To see this, simply note that $\lambda_\ell y_\ell = 0$ implies either $\lambda_\ell = 0$ or $y_\ell = 0$.

In the second case, we have $y_i = 0$ and hence $f_\ell(x) + y_\ell^2 = 0$ which implies $f_\ell(\boldsymbol{x}) = 0$. For the reverse implication, simply note that $\lambda_\ell \neq 0$ implies $f_\ell = 0$ and hence $y_\ell^2 = -f_\ell = 0$ which finally gives $y_\ell \lambda_\ell = 0$

If $y_\ell \neq 0$ (equivalently, $\lambda_\ell = 0$), on the other hand, then (94) implies $y_\ell \lambda_\ell = -f_\ell \lambda_\ell = 0$ and the equivalence is straightforward.

Condition (94) is equivalent to (86) (i.e the slack variables can be eliminated from the derivative)

We are thus left with showing that conditions (92) to (94) imply the non negativity of the multipliers λ_ℓ. For a general minimization problem,

$$\begin{aligned} \min \quad & f_0(\boldsymbol{x}) && (95) \\ \text{subject to} \quad & f_i(\boldsymbol{x}) \leq b_i, \quad (i = 1, 2, \ldots, m) && (96) \end{aligned}$$

if we denote the optimal solution corresponding to a given value of the bounds b_i, as $\boldsymbol{x}_0(\boldsymbol{b})$, we have

$$\frac{\partial f_0(\boldsymbol{x}_0(\boldsymbol{b}))}{\partial b_i} = -\lambda_i \tag{97}$$

But on the other any increase in the b_i leads to a larger feasible set and hence the variation in the value of f_0 for such a small increase in b_i must always be negative (i.e we can only do better when we increase the feasible set), following from this, we must have

$$\frac{\partial f_0(x_0(\boldsymbol{b}))}{\partial b_i} \leq 0 \tag{98}$$

which implies $\lambda_i \geq 0$

□

7 Halfspaces and hyperplanes

Given a vector $\boldsymbol{w} \in \mathbb{R}^N$, one can define a(n) (affine) hyperplane as the set of all points $\boldsymbol{x} \in \mathbb{R}$ satisfying the relation $\boldsymbol{w}^T \boldsymbol{x} + b = 0$.

The vector $\boldsymbol{w}$ encodes the normal to the hyperplane, which also means that for any two points $\boldsymbol{x}_1, \boldsymbol{x}_2$ that lie on the hyperplane, we necessarily have

$$\boldsymbol{w}^T \boldsymbol{x}_1 + b = \boldsymbol{w}^T \boldsymbol{x}_2 + b = 0 \tag{99}$$

(i.e the two points belong to the hyperplane) and hence,

$$\boldsymbol{w}^T (\boldsymbol{x}_1 - \boldsymbol{x}_2) = 0 \tag{100}$$

Every hyperplane naturally defines a splitting of the space into two halfspaces. The set of points which are lying above the plane (i.e formally, the points for which $\boldsymbol{w}^T \boldsymbol{x} + b > 0$), and the set of points which are lying under the plane ($\boldsymbol{w}^T \boldsymbol{x} + b < 0$). Every hyperplane can thus be used as a natural classifier,

$$y(\boldsymbol{x}) = \boldsymbol{w}^T \boldsymbol{x} + b \tag{101}$$

where we put the point $\boldsymbol{x}$ in class $\mathcal{C}_0$ if $y(\boldsymbol{x}) > 0$ and in class $\mathcal{C}_1$ if $y(\boldsymbol{x}) < 0$.

7.1 Distance of a point to a hyperplane

When discussing the robustness of a classifier (and in particular when introducing the notion of margin), we will need the notion of distance of a point to a hyperplane.

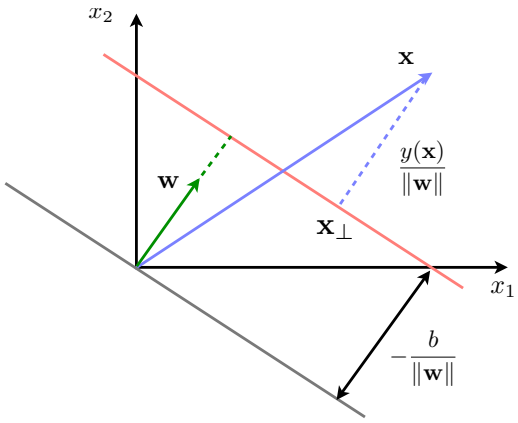

Figure 1: Distance of a point to a hyperplane

Consider Fig 1 below. We consider the plane in red defined as $y(\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x} + w_0$. For any point $\mathbf{x}$ (shown in blue), one can consider the decomposition

$$\mathbf{x} = \mathbf{x}_\perp + d\frac{\mathbf{w}}{\|\mathbf{w}\|} \tag{102}$$

That is we write $\mathbf{x}$ as the combination of its projection onto the plane and a contribution of length d along the normal vector to the plane, $\boldsymbol{w}$. d thus encodes the distance of $\mathbf{x}$ to the hyperplane. Now if we multiply (102) by $\mathbf{w^T}$,

$$\mathbf{w^T}\mathbf{x} = \mathbf{w^T}\mathbf{x}_\perp + \boldsymbol{w}^T d\frac{\mathbf{w}}{\|\mathbf{w}\|} \tag{103}$$

and add the bias, we get

$$\mathbf{w^T}\mathbf{x} + b = \mathbf{w^T}\mathbf{x}_\perp + b + \boldsymbol{w}^T d\frac{\mathbf{w}}{\|\mathbf{w}\|} \tag{104}$$

The left handside is simnply $y(\boldsymbol{x})$. The first term on the right handside is 0 as $\mathbf{x}_\perp$ lies along the plane (hence is orthogonal to $\mathbf{w}$). We are thus left with

$$y(\boldsymbol{x}) = \|\mathbf{w}\|\mathbf{d} \tag{105}$$

from which follows

$$d = \frac{y(\boldsymbol{x})}{\|\mathbf{w}\|} \tag{106}$$

So the signed (because $\mathbf{d}$ can be both positive and negative here) distance, is given by the ratio of the prediction $y(\boldsymbol{x})$ over the norm of the normal vector $\boldsymbol{w}$. We will use this when discussing Maximal margin classifiers.

8 Regression and regularization

The simplest regression model, and one of the most widely used, is the linear regression model. In linear regression we want to learn a plane $(\boldsymbol{w}, b)$ that describes the data as well as possible. The idea here is that if we can show that such a plane is a good representation of the relation between $y(\boldsymbol{x}_n)$ and $\boldsymbol{x}_n$ for the training data we have, then it might give reliable predictions on new data $\boldsymbol{x}_n$

To do so, for a given set of training points, $(\boldsymbol{x}_n, y_n)$, we minimize the sum of squared errors between the predictions $y(\boldsymbol{x}_n)$ from the plane and the exact target values , t_n,

$$\underset{\boldsymbol{w},b}{\operatorname{argmin}} \quad \sum_{n=1}^{N} |\hat{y}(\boldsymbol{x}^n; \boldsymbol{w}, b) - t^n|^2 \tag{107}$$

$$= \underset{\boldsymbol{w},b}{\operatorname{argmin}} \quad \sum_{n=1}^{N} |\boldsymbol{w}^T \boldsymbol{x}^n + b - t^n|^2 \tag{108}$$

$$= \underset{\boldsymbol{w},b}{\operatorname{argmin}} \quad \sum_{n=1}^{N} |\sum_{i=1}^{D} \boldsymbol{x}_i^n \boldsymbol{w}_i + b - t_n|^2 \tag{109}$$

The model remains linear if we replace the original points by a representation in some feature space, $\boldsymbol{\phi}(\boldsymbol{x})$. In this case, the model reads as

$$y(\boldsymbol{x}^n; \boldsymbol{w}; b) = b + \sum_{j=1}^{D} w_j \phi_j(\boldsymbol{x}^n) \tag{110}$$

and, including the bias b in the weight vector $\boldsymbol{w}$, $\tilde{\boldsymbol{w}} = [1, \boldsymbol{w}]$ and letting $\tilde{\boldsymbol{\phi}}(\boldsymbol{x}) = [1,\ \boldsymbol{\phi}(\boldsymbol{x})]$, we have

$$\underset{\boldsymbol{w},b}{\operatorname{argmin}} \sum_{n=1}^{N} |y(\boldsymbol{x}^n; \tilde{\boldsymbol{w}}) - t^n|^2 = \sum_{n=1}^{N} \left|\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n) - t_n\right|^2 \tag{111}$$

The solution to problem (111) can be computed in closed form by setting the derivatives to 0 and then solving for $\boldsymbol{w}$ and b. Setting the derivative of (107) (on the generic feature formulation) to 0, we get

$$\sum_{n=1}^{N} \left(t_n - \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n)\right) \boldsymbol{\phi}(\boldsymbol{x}_n)^T = 0 \tag{112}$$

(Verify this using the matrix derivatives). When solving for $\boldsymbol{w}$, we thus have

$$\sum_{n=1}^{N} t_n \boldsymbol{\phi}(\boldsymbol{x}_n)^T - \boldsymbol{w}^T \left(\sum_{n=1}^{N} \boldsymbol{\phi}(\boldsymbol{x}_n)^T \boldsymbol{\phi}(\boldsymbol{x}_n) \right) = 0 \tag{113}$$

which gives

$$\boldsymbol{w} = (\boldsymbol{\Phi}^T \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T \boldsymbol{t} \tag{114}$$

Here $\boldsymbol{t}$ is the vector concatenating the targets $\boldsymbol{t} = (t_1, \ldots, t_n)$, and $\boldsymbol{\Phi}$ is the matrix whose columns encode the feature vectors $\boldsymbol{\phi}(\boldsymbol{x}_n)$, i.e.

$$\boldsymbol{\Phi} = \begin{pmatrix} \phi_0(\boldsymbol{x}_0) & \phi_1(\boldsymbol{x}_0) & \ldots & \phi_M(\boldsymbol{x}_0) \\ \phi_0(\boldsymbol{x}_1) & \phi_1(\boldsymbol{x}_1) & \ldots & \phi_M(\boldsymbol{x}_1) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\boldsymbol{x}_N) & \phi_1(\boldsymbol{x}_N) & \ldots & \phi_0(\boldsymbol{x}_N) \end{pmatrix} \tag{115}$$

The matrix $\boldsymbol{\Phi}^+ = (\boldsymbol{\Phi}^T \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^T$ is precisely the Moore Penrose Pseudo inverse which was introduced in Definition 1.

This solution highlights an important aspect of linear regression: For the solution of problem (111) to be well defined, we need the Gram matrix $\boldsymbol{\Phi}^T \boldsymbol{\Phi}$ to be invertible.

When this matrix is not invertible, it means that columns or rows, of that matrix are linearly dependent and hence, that a given subset of the feature vectors $\boldsymbol{\phi}(\boldsymbol{x}_n)$ can be written from the

knowledge of all the remaining features. The issue with such redundancy in the representation of the data is that one can define a perfectly valid classifier for the dataset, but for which the weights can vary arbitrarily and in particular, be arbitrarily high as they can cancel each other. Indeed, asssume that for all training points, features $\phi_1(\boldsymbol{x}_n)$ can be obtained as the combination $\phi_2(\boldsymbol{x}_n) + 2\phi_3(\boldsymbol{x}_n)$. Then for any given classifier,

$$y(\boldsymbol{x}) = b + w_1\phi_1 + w_2\phi_2 + w_3\phi_3 \tag{116}$$

adding a term

$$\alpha w\phi_1 - w(\phi_2 + 2\phi_3) \tag{117}$$

will not change the regression of the training points, for any $w \in \mathbb{R}$, since at all those training points we have

$$\phi_1(\boldsymbol{x}_n) = \phi_2(\boldsymbol{x}_n) + 2\phi_3(\boldsymbol{x}_n) \tag{118}$$

In particular, we could take w arbitrarily large and get a classifier that "fits" the training data perfectly well. The issue however is that for new data (or test data), the two classifiers will give very different results. In practice, we therefore want to avoid such situations and we will add a penalty on the coefficients $(w_1, \dots, w_N)$ and bias b.

8.1 Regularizers

There are three popular approaches at regularizing the linear regression problem (111).

- ℓ_2 (Ridge regression). Here we simply minimize the sum of squared weights (squared ℓ_2 norm),

$$\underset{\boldsymbol{w},b}{\text{argmin}} \quad \sum_{n=1}^{N} \left|\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n) - t_n\right|^2 + \lambda \sum_{i=1}^{D} |\boldsymbol{w}_i|^2 \tag{119}$$

- ℓ_1 (LASSO). Here we minimize the sum of the absolute value of the weights,

$$\underset{\boldsymbol{w},b}{\text{argmin}} \quad \sum_{n=1}^{N} \left|\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n) - t_n\right|^2 + \lambda \sum_{i=1}^{D} |w_i| \tag{120}$$

- Finally in Best Subset Selection (which can be throught of as a minimization of the number of non zero weights), we found for each K, the best size-K subset of regression coefficients $w_1, \dots, w_N$.

In all of those approaches, the optimal choice of λ or for best subset selection, the optimal size of the subset of regression coefficients, is fixed through cross validation, by computing the regression coefficients associated to a particular value of λ for a subset $\mathcal{S}_1$ of the whole dataset, then testing the model on the remaining $\mathcal{S} \setminus \mathcal{S}_1$ data. And then selecting the λ that leads to the smallest generalization error. The two formulations (119) and (120) both admit constrained variations,

$$\underset{\boldsymbol{w},b}{\text{argmin}} \quad \sum_{n=1}^{N} \left|\boldsymbol{w}^T\boldsymbol{\phi}(\boldsymbol{x}_n) - t_n\right|^2 \tag{121}$$

$$\text{subject to} \quad \sum_{i=1}^{D} |\boldsymbol{w}_i|^2 \leq t \tag{122}$$

and

$$\underset{\boldsymbol{w},b}{\text{argmin}} \quad \sum_{n=1}^{N} \left| \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}_n) - t_n \right|^2 \tag{123}$$

$$\text{subject to} \quad \sum_{i=1}^{D} |\boldsymbol{w}_i| \leq t \tag{124}$$

respectively.

Finally, note that other regularization terms are possible. In particular, any ℓ_p norm, $p \geq 1$

$$\|\boldsymbol{x}\|_p = \left(\sum_{i=1}^{D} w_i^p \right)^{1/p} \tag{125}$$

can be used as a regularizer.

In the case of the ridge regression formulation (119), it remains possible to derive a closed form solution as follows. We consider the unconstrained formulation, take the derivatives with respect to $(\boldsymbol{w}, b)$ and set those derivatives to 0, we get

$$\boldsymbol{w}^{\text{ridge}} = (\boldsymbol{X}^T \boldsymbol{X} + \lambda \boldsymbol{I})^{-1} \boldsymbol{X}^T \boldsymbol{t} \tag{126}$$

or in the case of feature vectors $\boldsymbol{\phi}(\boldsymbol{x})$,

$$\boldsymbol{w}^{\text{ridge}} = (\boldsymbol{\Phi}^T \boldsymbol{\Phi} + \lambda \boldsymbol{I})^{-1} \boldsymbol{\Phi}^T \boldsymbol{t} \tag{127}$$

Hence you see that we have replaced the matrix $(\boldsymbol{\Phi}^T \boldsymbol{\Phi})$ by a better conditioned version of this matrix as it is now combined to the identity which is invertible.

9 Bias Variance tradeoff

9.1 Expected Loss

Suppose our data set is given by $\mathcal{D} = \{(x, t)\}$ where t are the labels associated to each point $\boldsymbol{x}$. We will assume that the data is distributed according to an ideal function $y(\boldsymbol{x})$ plus some perturbations.

$$t = y(\boldsymbol{x}) + \varepsilon \tag{128}$$

t is thus the observed label. In order to study how well a given regression model is fitting the data, one can look at the expected loss

$$\mathbb{E}[L] = \int \int (h(\boldsymbol{x}) - t)^2 \; p(\boldsymbol{x}, t) \; d\boldsymbol{x} \; dt \tag{129}$$

Note that given the objective (129), the optimal model we can choose for $h(\boldsymbol{x})$ is to solve the minimization problem

$$\min_{h(\boldsymbol{x})} \int \int (h(\boldsymbol{x}) - t)^2 \; d\boldsymbol{x} dt \tag{130}$$

To do this, we set the first order derivatives to 0, and get

$$\frac{\delta \mathbb{E}L}{\delta h(\boldsymbol{x})} = 2 \int (h(\boldsymbol{x}) - t) p(\boldsymbol{x}, t) \, d\boldsymbol{x} \, dt \tag{131}$$

from this we get

$$\int h(\boldsymbol{x}) p(\boldsymbol{x}) \, d\boldsymbol{x} = \int t p(t, \boldsymbol{x}) \, d\boldsymbol{x} \, dt \tag{132}$$

The t disappeared in the left handside because $h(\boldsymbol{x})$ does not depend on t so we can simply integrate over all values of t.

This last equation tells us that in general we should at every value $\boldsymbol{x}$ choose the model $h(\boldsymbol{x})$ corresponding to

$$h(\boldsymbol{x}) = \frac{\int t \, p(\boldsymbol{x}, t) \, d\boldsymbol{x} \, dt}{p(\boldsymbol{x})} = \int t p(t|\boldsymbol{x}) \, dt = \mathbb{E}_t \left\{ t|\boldsymbol{x} \right\} \tag{133}$$

the last line follows from $p(\boldsymbol{x}, t) = p(t|\boldsymbol{x}) p(\boldsymbol{x})$. This is however not always possible.

On the other hand, we always have

$$\begin{aligned} (h(\boldsymbol{x}) - t)^2 &= (h(\boldsymbol{x}) - \mathbb{E}\left\{ t|\boldsymbol{x} \right\} + \mathbb{E}\left\{ t|\boldsymbol{x} \right\} - t)^2 && (134) \\ &= (h(\boldsymbol{x}) - \mathbb{E}\left\{ t|\boldsymbol{x} \right\})^2 + (\mathbb{E}\left\{ t|\boldsymbol{x} \right\} - t)^2 + 2 \left(h(\boldsymbol{x}) - \mathbb{E}\left\{ t|\boldsymbol{x} \right\} \right) \left(\mathbb{E}\left\{ t|\boldsymbol{x} \right\} - t \right) && (135) \end{aligned}$$

If you substitute this last expression into the expected Loss (129) you can see that the cross terms will disappear as it leads to the following four terms

$$\begin{aligned} 2 \int h(\boldsymbol{x}) \mathbb{E}\left\{ t|\boldsymbol{x} \right\} p(\boldsymbol{x}, t) \, d\boldsymbol{x} \, dt &= 2 \int h(\boldsymbol{x}) \mathbb{E}\left\{ t|\boldsymbol{x} \right\} p(\boldsymbol{x}) \, d\boldsymbol{x} && (136) \\ -2 \int h(\boldsymbol{x}) t p(\boldsymbol{x}, t) \, dt \, d\boldsymbol{x} &= -2 \int \mathbb{E}\left\{ t|\boldsymbol{x} \right\} h(\boldsymbol{x}) p(\boldsymbol{x}) \, d\boldsymbol{x} && (137) \\ -2 \int \mathbb{E}\left\{ t|\boldsymbol{x} \right\}^2 p(\boldsymbol{x}, t) \;\; d\boldsymbol{x} \, dt &= -2 \int \mathbb{E}\left\{ t|\boldsymbol{x} \right\}^2 p(\boldsymbol{x}) \, d\boldsymbol{x} && (138) \\ 2 \int \mathbb{E}\left\{ t|\boldsymbol{x} \right\} t p(\boldsymbol{x}, t) \, d\boldsymbol{x} \, dt &= 2 \int \mathbb{E}\left\{ t|\boldsymbol{x} \right\}^2 p(\boldsymbol{x}) \, d\boldsymbol{x} && (139) \end{aligned}$$

So that the average loss reduces to

$$\begin{aligned} \mathbb{E}[L] = \int (t - h(\boldsymbol{x}))^2 \, d\boldsymbol{x} \, dt &= \int (h(\boldsymbol{x}) - \mathbb{E}\left\{ t|\boldsymbol{x} \right\})^2 p(\boldsymbol{x}) \, d\boldsymbol{x} && (140) \\ &+ \int (\mathbb{E}\left\{ t|\boldsymbol{x} \right\} - t)^2 \, dt \, p(\boldsymbol{x}) \, d\boldsymbol{x} && (141) \\ &= \int (h(\boldsymbol{x}) - \mathbb{E}\left\{ t|\boldsymbol{x} \right\})^2 p(\boldsymbol{x}) \, d\boldsymbol{x} && (142) \\ &+ \int \sigma_x^2 \, p(\boldsymbol{x}) \, d\boldsymbol{x} && (143) \end{aligned}$$

Where I used

$$\int (\mathbb{E}\left\{ t|\boldsymbol{x} \right\} - t)^2 p(\boldsymbol{x}, t) \, d\boldsymbol{x} \, dt = \int \mathrm{Var}[t|\boldsymbol{x}] p(\boldsymbol{x}) \, d\boldsymbol{x} \tag{144}$$

Once again, the t is integrated over in the first integral because none of the functions $h(\boldsymbol{x})$ and $\mathbb{E}\{t|\boldsymbol{x}\}$ depend on t anymore.

The final expression of the average loss is then

$$\mathbb{E}[L] = \int (h(\boldsymbol{x}) - \mathbb{E}\{t|\boldsymbol{x}\})^2 \, p(\boldsymbol{x}) \, d\boldsymbol{x} + \int \sigma_x^2 \, p(\boldsymbol{x}) \, d\boldsymbol{x} \tag{145}$$

You see that there is a first term (MSE) that depends on the model and s second term that depends only on the data. Now we will show that the first term can decomposed into a *Variance* and a *bias* contributions.

9.2 Bias Variance Tradeoff

In the derivations above, we haven't make any assumption on the regression model learned $h(\boldsymbol{x})$. In general, when we learn a regression model, we learn it from a subset of the data. We will denote this subset $\mathcal{D}_i$ so that $\mathcal{D}_i \subset \mathcal{D}$ and I will now denote our regression model as $h(\boldsymbol{x}|\mathcal{D}_i)$ to emphasize the dependence on the particular choice of $\mathcal{D}_i$. Now that being said, if we use $\bar{t}(x)$ to denote $\mathbb{E}\{t|\boldsymbol{x}\}$, we have the decomposition

$$\begin{aligned}
&\big(h(\boldsymbol{x};\mathcal{D}_i) - \bar{t}(x)\big)^2 && (146)\\
&= \Big(h(\boldsymbol{x};\mathcal{D}_i) - \mathbb{E}_{\mathcal{D}_i}\{h(\boldsymbol{x};\mathcal{D}_i)\} + \mathbb{E}_{\mathcal{D}_i}\{h(\boldsymbol{x};\mathcal{D}_i)\} - \bar{t}(x)\Big)^2 && (147)\\
&= \Big(h(\boldsymbol{x};\mathcal{D}_i) - \mathbb{E}_{\mathcal{D}_i}\{h(\boldsymbol{x};\mathcal{D}_i)\}\Big)^2 + \Big(\mathbb{E}_{\mathcal{D}_i}\{h(\boldsymbol{x};\mathcal{D})\} - \bar{t}(x)\Big)^2 && (148)\\
&+ 2\Big(h(\boldsymbol{x};\mathcal{D}_i) - \mathbb{E}_{\mathcal{D}_i}\{h(\boldsymbol{x};\mathcal{D}_i)\}\Big)\Big(\mathbb{E}_{\mathcal{D}_i}\{h(\boldsymbol{x};\mathcal{D}_i)\} - \bar{t}(x)\Big) && (149)
\end{aligned}$$

If we take the expectation over all possible datasets $\mathcal{D}_i$, we will get a measure on how well the model $h(\boldsymbol{x})$ performs on average for the different datasets. Below I use $\mathbb{E}_{\mathcal{D}_i}$ to denote the average when sample random datasets $\mathcal{D}_i$ from the full set of samples $\mathcal{D}$.

$$\begin{aligned}
&\mathbb{E}_{\mathcal{D}_i}\big\{h(\boldsymbol{x};\mathcal{D}) - \bar{t}(x)\big\}^2 && (150)\\
&= \mathbb{E}_{\mathcal{D}_i}\Big(h(\boldsymbol{x};\mathcal{D}_i) - \mathbb{E}_{\mathcal{D}_i}\{h(\boldsymbol{x};\mathcal{D}_i)\}\Big)^2 + \mathbb{E}_{\mathcal{D}_i}\Big(\mathbb{E}_{\mathcal{D}_i}\{h(\boldsymbol{x};\mathcal{D})\} - \bar{t}(x)\Big)^2 && (151)\\
&+ 2\mathbb{E}_{\mathcal{D}_i}\Big(h(\boldsymbol{x};\mathcal{D}_i) - \mathbb{E}_{\mathcal{D}_i}\{h(\boldsymbol{x};\mathcal{D}_i)\}\Big)\Big(\mathbb{E}_{\mathcal{D}_i}\{h(\boldsymbol{x};\mathcal{D}_i)\} - \bar{t}(x)\Big) && (152)\\
&= \underbrace{\Big(\mathbb{E}_{\mathcal{D}}\{y(\boldsymbol{x};\mathcal{D})\} - \bar{t}(x)\Big)^2}_{\text{bias}^2} + \underbrace{\mathbb{E}_{\mathcal{D}}\Big[(h(\boldsymbol{x};\mathcal{D}) - \mathbb{E}_{\mathcal{D}}\{h(\boldsymbol{x};\mathcal{D})\})^2\Big]}_{\text{variance}} && (153)
\end{aligned}$$

What this tells you in terms of the complexity of the regression models is that complex models will usually have a large variance because within the class of complex models, the models will usually *vary* a lot. I.e if you have a very complicated model with many many parameters that perfectly fits the data in $\mathcal{D}_i$, then when you will take another dataset $\mathcal{D}_j$, the model that you build for $\mathcal{D}_j$ will be completely different from the model you built with $\mathcal{D}_i$. So complex models usually have larger variance.

The bias, on the other hand is a measure of how well the model "captures" the average behavior of the data $\mathbb{E}\{t|\boldsymbol{x}\}$. In general (except if the data is assumed to be linear which is the framework of the Gauss Markov Theorem discussed below), the simpler models will not be able to capture such behavior. As an example, imagine we have data of the form

$$t = y(x) + \varepsilon = \sin(x) + \varepsilon \tag{154}$$

	simpler models	complex models
bias	large	small
variance	small	large

Table 2: The bias variance trade-off and the model complexity

Then without using non linear terms (and defining our new variables as $x' = \phi(x)$ where $\phi(x)$ encodes the non linearity), it is impossible to design a model of the form $h(x) = \beta_0 + \langle \beta_1, x \rangle$ such that

$$h(\boldsymbol{x}) = \sin(x) \tag{155}$$

This should illustrate the fact that simpler models have a large bias (When we don't make assumptions on the data distribution)

In general we thus have the relation summarized in table 2.

9.3 The particular case of linearly generated data and the Gauss Markov Theorem

So far we haven't made any assumptions on the data. When we don't make assumptions on the data, the relation of table 2 holds in general.

However, when we know that the data has a particular distribution, we can better describe the quality of approximation of given models. This is the framework of the *Gauss Markov Theorem* (GMT).

When the data is distributed according to a linear model, so that

$$t = y(x) + \varepsilon = \alpha_0 + \alpha_1 x + \varepsilon, \tag{156}$$

then the linear model has no bias (note that this might seem counterintuitive compared to table 2 but this is because here we assumed that the data is linear).

Moreover, the *Gauss Markov theorem* states that among all possible linear estimators of the α_0 and α_1 appearing in (for linear data), the linear regression model, which is computed by minimizing the residual sum of squares,

$$\underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{N} (t_i - \beta_0 - \beta_1 x_i)^2 \tag{157}$$

is the best estimator. In the Gauss Markov framework, when we say data, we mean the t_i. What we now call data are the noisy measurements $t_i = \beta_0 + \beta_1 X_i + \varepsilon$. Because we've put ourselves in a framework where we now know how the data is distributed, $t_i(x) = \beta_0 + \beta_1 x_i + \varepsilon$, t_i is enough to recover the data when there is no noise and we thus call t_i the data. In this particular framework, a linear estimator for the coefficients β is an estimator of the form

$$\tilde{\beta} = M\boldsymbol{t} \tag{158}$$

where $\boldsymbol{t} = [t_1, t_2, \ldots, t_N]$ encode the labels from $\mathcal{D}_i$. The estimator provided by the linear regression approach (157) is a particular such estimator as it can read as

$$\hat{\beta}_{\text{RSS}} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{t} \tag{159}$$

$$= (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T(\boldsymbol{X}\beta + \varepsilon) \tag{160}$$

To prove the GMT, we consider any other linear estimator of the coefficients $\boldsymbol{\beta} = \beta_0, \beta_1$, $\tilde{\boldsymbol{\beta}} = \hat{\boldsymbol{\beta}}_{\text{RSS}} + \Delta \boldsymbol{t}$ where Δ is a perturbation.

We first require the linear estimator $\tilde{\boldsymbol{\beta}}$ to be unbiased, which gives

$$\begin{aligned}
\mathbb{E}\left\{\tilde{\boldsymbol{\beta}}\right\} &= \mathbb{E}\left\{\hat{\boldsymbol{\beta}}\right\} + \mathbb{E}\left\{\Delta \boldsymbol{t}\right\} && (161)\\
&= \mathbb{E}\left\{\Delta \boldsymbol{X}\boldsymbol{\beta} + \varepsilon\Delta\right\} && (162)\\
&= \Delta \boldsymbol{X}\boldsymbol{\beta} + 0 && (163)\\
&= 0 && (164)
\end{aligned}$$

The only source of randomness here is the noise ε and we assumed that $\mathbb{E}\varepsilon = 0$. As we haven't made any assumption on $\boldsymbol{\beta}$, the last line implies that for the linear estimator to be unbiased, we must have

$$\Delta \boldsymbol{X} = 0 \tag{165}$$

Now, for the variance, first note that

$$\begin{aligned}
\text{Var}\left\{\hat{\boldsymbol{\beta}}\right\} &= \mathbb{E}\left\{(\hat{\beta} - \mathbb{E}\hat{\boldsymbol{\beta}})^2\right\} && (166)\\
&= \mathbb{E}\left\{\hat{\boldsymbol{\beta}}^2\right\} - \boldsymbol{\beta}^2 && (167)
\end{aligned}$$

as the estimator is unbiased. For any other unbiased estimator, $\tilde{\boldsymbol{\beta}}$ of the form $\tilde{\boldsymbol{\beta}} = \mathbb{E}\left\{\right\}$

$$\begin{aligned}
&\text{Var}\left\{\tilde{\boldsymbol{\beta}}\right\} && (168)\\
&= \mathbb{E}\left\{(\tilde{\boldsymbol{\beta}} - \mathbb{E}\tilde{\boldsymbol{\beta}})(\tilde{\boldsymbol{\beta}} - \mathbb{E}\tilde{\boldsymbol{\beta}})^T\right\} && (169)\\
&= \mathbb{E}\left\{\left(\hat{\boldsymbol{\beta}}_{\text{RSS}} + \boldsymbol{\Delta t} - \boldsymbol{\beta} - \mathbb{E}\left\{\Delta \boldsymbol{t}\right\}\right)\left(\hat{\boldsymbol{\beta}}_{\text{RSS}} + \boldsymbol{\Delta t} - \boldsymbol{\beta} - \mathbb{E}\left\{\Delta \boldsymbol{t}\right\}\right)^T\right\} && (170)\\
&= \mathbb{E}\left\{\left(\hat{\boldsymbol{\beta}}_{\text{RSS}} - \boldsymbol{\beta}\right)\left(\hat{\boldsymbol{\beta}}_{\text{RSS}} - \boldsymbol{\beta}\right)^T\right\} && (171)\\
&+ \mathbb{E}\left\{(\Delta \boldsymbol{t} - \mathbb{E}\left\{\Delta \boldsymbol{t}\right\})(\Delta \boldsymbol{t} - \mathbb{E}\left\{\Delta \boldsymbol{t}\right\})^T\right\} && (172)\\
&- \mathbb{E}\left\{(\hat{\boldsymbol{\beta}}_{RSS} - \boldsymbol{\beta})(\Delta \boldsymbol{t} - \mathbb{E}\left\{\Delta \boldsymbol{t}\right\})^T\right\} && (173)\\
&- \mathbb{E}\left\{(\Delta \boldsymbol{t} - \mathbb{E}\left\{\Delta \boldsymbol{t}\right\})(\hat{\boldsymbol{\beta}}_{RSS} - \boldsymbol{\beta})^T\right\} && (174)
\end{aligned}$$

As $\hat{\boldsymbol{\beta}}_{\text{RSS}}$ is unbiased, the last two (cross) terms disappear and we are left with

$$\begin{aligned}
\text{Var}\left\{\tilde{\boldsymbol{\beta}}\right\} &= \text{Var}\left\{\hat{\boldsymbol{\beta}}_{RSS}\right\} + \mathbb{E}\left\{(\Delta \boldsymbol{t})(\Delta \boldsymbol{t})^T\right\} && (175)\\
&- \mathbb{E}\left\{(\Delta \boldsymbol{t})\right\}\mathbb{E}\left\{(\Delta \boldsymbol{t})\right\}^T && (176)\\
&= \text{Var}\left\{\hat{\boldsymbol{\beta}}_{RSS}\right\} + \Delta \boldsymbol{X}\boldsymbol{\beta}\boldsymbol{\beta}^T\boldsymbol{X}^T\boldsymbol{\Delta}^T && (177)\\
&+ \boldsymbol{\Delta}\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}^T\boldsymbol{\Delta}^T && (178)\\
&= \text{Var}\left\{\hat{\boldsymbol{\beta}}_{RSS}\right\} + \sigma^2\boldsymbol{\Delta}\boldsymbol{\Delta}^T && (179)
\end{aligned}$$

As soon as $\Delta \neq 0$, the diagonal of $\Delta\Delta^T$ is non zero as well. This in particular means that there is one component of $\tilde{\boldsymbol{\beta}}$ for which we always have

$$\mathbb{E}\left\{(\tilde{\beta}_k - \beta_k)^2\right\} > \mathbb{E}\left\{(\hat{\beta}_k - \beta_k)^2\right\} \tag{180}$$

Both estimators have zero bias (we will overestimate and underestimate the truth equally) but the errors we make will often be larger when we don't use the RSS estimator.

In conclusion, you can thus see that any linear estimator will only increase the variance with respect to $\hat{\boldsymbol{\beta}}_{\text{RSS}}$ (This is because we are in the particular framework of linearly generated data)

For this reason, the estimator $\hat{\beta}_{\text{RSS}}$ is sometimes called the BLUE (Best Linear Unbiased Estimator)

9.3.1 Increasing the bias to reduce generalization under the assumption of linearly generated data

That being said, the fact that the BLUE is the best estimator among all unbiased estimators with respect to linearly generated data, does not mean that it is not possible to do better in terms of future predictions. And this is because a similar bias variance tradeoff as the one discussed in section 9.2 applies.

Imagine that we take a new point x_0 and we apply our estimator $\tilde{\boldsymbol{\beta}}$ (*now we don't assume that this estimator is unbiased anymore*, this is just a general linear estimator) to this point to get an estimated label y, we can then write the expected mean square prediction error as before

$$\mathbb{E}_\varepsilon\left\{\left(\langle\tilde{\boldsymbol{\beta}},\boldsymbol{x}_0\rangle-\langle\boldsymbol{\beta},\boldsymbol{x}_0\rangle\right)^2\right\} \tag{181}$$

$$=\mathbb{E}_\varepsilon\left\{\left(\langle\tilde{\boldsymbol{\beta}},\boldsymbol{x}\rangle-\mathbb{E}\left\{\langle\tilde{\boldsymbol{\beta}},\boldsymbol{x}\rangle\right\}+\mathbb{E}\left\{\langle\tilde{\boldsymbol{\beta}},\boldsymbol{x}\rangle\right\}-\langle\boldsymbol{\beta},\boldsymbol{x}\rangle\right)^2\right\} \tag{182}$$

$$=\mathbb{E}_\varepsilon\left\{\left(\langle\tilde{\boldsymbol{\beta}},\boldsymbol{x}\rangle-\mathbb{E}\left\{\langle\tilde{\boldsymbol{\beta}},\boldsymbol{x}\rangle\right\}\right)^2\right\} \tag{183}$$

$$+\mathbb{E}_\varepsilon\left\{\left(\mathbb{E}\left\{\langle\tilde{\boldsymbol{\beta}},\boldsymbol{x}\rangle\right\}-\langle\boldsymbol{\beta},\boldsymbol{x}\rangle\right)^2\right\} \tag{184}$$

$$+2\mathbb{E}_\varepsilon\left\{\left(\langle\tilde{\boldsymbol{\beta}},\boldsymbol{x}_0\rangle-\mathbb{E}_\varepsilon\left\{\tilde{\boldsymbol{\beta}},\boldsymbol{x}_0\right\}\right)\left(\mathbb{E}\left\{\langle\tilde{\boldsymbol{\beta}},\boldsymbol{x}_0\rangle\right\}-\langle\boldsymbol{\beta},\boldsymbol{x}_0\rangle\right)\right\} \tag{185}$$

The last term vanishes as the second factor is deterministic and the first factor has mean 0. We thus once again have

$$\mathbb{E}_\varepsilon\left\{\left(\langle\tilde{\boldsymbol{\beta}},\boldsymbol{x}_0\rangle-\langle\boldsymbol{\beta},\boldsymbol{x}_0\rangle\right)^2\right\}=\underbrace{\mathbb{E}_\varepsilon\left\{\left(\langle\tilde{\boldsymbol{\beta}},\boldsymbol{x}\rangle-\mathbb{E}\left\{\langle\tilde{\boldsymbol{\beta}},\boldsymbol{x}\rangle\right\}\right)^2\right\}}_{\text{variance}} \tag{186}$$

$$+\underbrace{\mathbb{E}_\varepsilon\left\{\left(\mathbb{E}\left\{\langle\tilde{\boldsymbol{\beta}},\boldsymbol{x}\rangle\right\}-\langle\boldsymbol{\beta},\boldsymbol{x}\rangle\right)^2\right\}}_{\text{bias}^2} \tag{187}$$

This shows that even in the framework of linearly generated data, despite the fact that $\hat{\boldsymbol{\beta}}$ is the BLUE estimator, it does not mean that this estimator will be the best (on average over the noise) at predicting a new value of $t=\langle\boldsymbol{x},\boldsymbol{\beta}\rangle$. There might be an estimator $\tilde{\boldsymbol{\beta}}$ with a non zero bias (on linear data) that will have smaller variance.

That is to say, if I generate many values $t_k=\langle\beta,x_k\rangle+\varepsilon_k$, learn my estimator $\hat{\beta}$ from those values, then this estimator will be unbiased but when I will want to get the prediction for a new point x_0, the average prediction might be better if I lower the variance and increase the bias a little. Again this might sound counter intuitive with respect to table 2 but remember that we assumed linearly distributed data here.

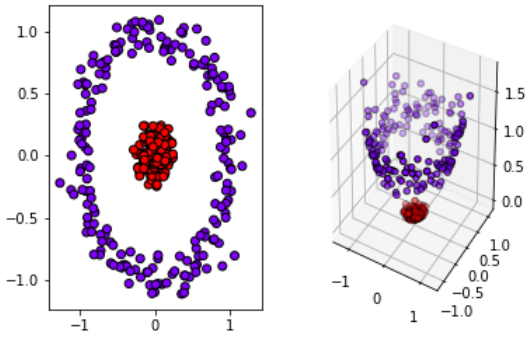

Figure 2: From non linearly separable data to linearly separable data via higher dimensional feature space

10 Kernels

Let $\beta_0 \in \mathbb{R}$ to denote the bias and $\boldsymbol{\beta}_1 \in \mathbb{R}^D$ to denote the vector of weights. In practice, we often have to deal with prototypes, $\{\boldsymbol{x}_\mu\}_{\mu=1}^N$ which are not linearly separable in their original space. One approach then consists in introducing features and to "map" the original prototypes $\boldsymbol{x}_\mu$ into a space (the feature space) in which those prototypes become linearly separable. We use ϕ to denote the underlying transformation so that for a given datapoint $\boldsymbol{x}$, $\phi(\boldsymbol{x})$ denotes the feature vector of $\boldsymbol{x}$.

Example 1. *Consider the dataset shown in Fig. 2 (left). In the original $\mathbb{R}^2$ space, the data given by pairs (x_1, x_2) is clearly not linearly separable. However, it is possible to introduce a transformation $\phi : \boldsymbol{x} \mapsto \phi(\boldsymbol{x})$ defined as $\phi(\boldsymbol{x}) = \phi(x_1, x_2) = (x_1, x_2, x_1^2 + x_2^2)$.*

Here, introducing an additional dimension, and setting this dimension to be equal to the radius of the points in the original space, results in the purple points being placed above the red points, thus leaving some space for a separating plane.

Relying on feature vectors to learn classifiers is at the core of machine learning. However, such an approach often requires to map the data to a much higher dimensional space (i.e higher than the number of prototypes $\boldsymbol{x}_\mu$). In this case, it is more interesting to avoid computing the feature vectors explicitely and to rely instead on a measure of similarity between the points in the feature space. After all, the important information here is really how the points compare to each other in the feature space. This information is encoded through kernels

Mathematically, a kernel is nothing else than a function that measure similarity between points. This is summarized by the definition below

Definition 19 (Kernel)**.** *We define a kernel to be the real valued function of two arguments $\boldsymbol{x}$ and $\boldsymbol{x}'$ from a space $\mathcal{X}$. Typically this function is taken to be symmetric $k(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}', \boldsymbol{x})$ and non negative, $k(\boldsymbol{x}, \boldsymbol{x}') \geq 0$. Those two properties enable us to interpret the kernel $k(\boldsymbol{x}, \boldsymbol{x}')$ as a measure of similarity between points.*

Examples of popular kernels are listed below

- The Gaussian kernel
$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}')\boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{x}'))$$
- When the covariance matrix $\boldsymbol{\Sigma}$ is isotropic/spherical, we get the isotropic kernel
$$k(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|^2}{2\sigma^2})$$

- The cosine similarity:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \frac{\boldsymbol{x}^T \boldsymbol{x}'}{\|\boldsymbol{x}\|_2 \|\boldsymbol{x}'\|_2}$$

- The (non stationnary) polynonial kernel:

$$k(\boldsymbol{x}, \boldsymbol{x}') = (\gamma \boldsymbol{x}^T \boldsymbol{x}' + r)^M$$

  where $r > 0$. This kernel corresponds to a feature vector $\phi(\boldsymbol{x})$ that contains all the monomials up to degree M.

The isotropic kernel is an example of Radial basis function (a function whose distance decreases or increases monotonically with respect to a central point). Those functions are multivariate (i.e they are defined on $\boldsymbol{x} \in \mathbb{R}^D$) but they reduce to a scalar function of the Euclidean norm $\|\boldsymbol{x}\|^2$ of their argument $\boldsymbol{x}$, i.e.,

$$F(\boldsymbol{x}) = f(\|\boldsymbol{x}\|_2) = \phi(r), \quad \boldsymbol{x} \in \mathbb{R}^D \tag{188}$$

Examples of radial basis functions include

- The Gaussian RBF: $F(\|\boldsymbol{x} - \boldsymbol{x}'\|) = \exp(-\alpha^2 \|\boldsymbol{x} - \boldsymbol{x}'\|^2)$
- The Multiquadratic RBF: $F(\|\boldsymbol{x} - \boldsymbol{x}'\|) = \sqrt{1 + \alpha^2 \|\boldsymbol{x} - \boldsymbol{x}'\|^2}$

As soon as one has access to the feature vector $\phi(\boldsymbol{x})$ (for example when this feature is finite dimensional), one can build a valid kernel by defining $k(\boldsymbol{x}, \boldsymbol{x}')$ as the inner product of the feature vectors of $\boldsymbol{x}$ and $\boldsymbol{x}'$, i.e.,

$$k(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x})^T \phi(\boldsymbol{x}') \tag{189}$$

10.1 From linear regression to kernel regression

When the dataset is not linearly separable in the original space, we turn to a formulation on the feature space and try to find the plane defined by $(\beta_0, \boldsymbol{\beta}_1)$ that minimizes $J(\boldsymbol{\beta})$, i.e.,

$$\min J(\boldsymbol{\beta}) = \frac{1}{2} \sum_{n=1}^{N} \left\{ \boldsymbol{\beta}_1^T \boldsymbol{\phi}(\boldsymbol{x}) + \beta_0 - t_n \right\}^2 + \frac{\lambda}{2} \|\boldsymbol{\beta}\|^2 \tag{190}$$

Here the separating plane is defined through the vector of parameters $\boldsymbol{\beta} = (\beta_0, \boldsymbol{\beta}_1)$, $\beta_0 \in \mathbb{R}$, $\boldsymbol{\beta}_1 \in \mathbb{R}^D$. The solution for β can be obtained by computing the derivative of $J(\boldsymbol{\beta})$ with respect to $\boldsymbol{\beta}$ and settting it to zero. For pairs $\{(\boldsymbol{x}_n, t_n)\}_{n=1}^N$, we have

$$\frac{\partial J(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}} = 0 \Rightarrow \boldsymbol{\beta} = -\frac{1}{\lambda} \sum_{n=1}^{N} \left\{ \boldsymbol{\beta}^T \tilde{\boldsymbol{\phi}}(\boldsymbol{x}_n) - t_n \right\} \tilde{\boldsymbol{\phi}}(\boldsymbol{x}_n) \tag{191}$$

Now introducing the notation a_n for the quantities

$$a_n = -\frac{1}{\lambda} \left\{ \boldsymbol{\beta}^T \tilde{\boldsymbol{\phi}}(\boldsymbol{x}) - t_n \right\} \tag{192}$$

Then the solution for $\boldsymbol{\beta}$ can read as

$$\boldsymbol{\beta} = \sum_{n=1}^{N} a_n \boldsymbol{\phi}(\boldsymbol{x}_n) = \boldsymbol{\Phi}^T \boldsymbol{a} \tag{193}$$

Where $\boldsymbol{\Phi}$ is the matrix whose i^{th} row is given by $\boldsymbol{\phi}^T(x_i)$. Now substituting this expression into (191), we get

$$J(\boldsymbol{a}) = \frac{1}{2}\boldsymbol{a}^T\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{a} - \boldsymbol{a}\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{t} + \frac{1}{2}\boldsymbol{t}^T\boldsymbol{t} + \frac{\lambda}{2}\boldsymbol{a}^T\boldsymbol{\Phi}\boldsymbol{\Phi}^T\boldsymbol{a} \tag{194}$$

From the discussion above, you see that the product $\boldsymbol{\Phi}\boldsymbol{\Phi}^T$ is actually encoding all inner products $\langle\boldsymbol{\phi}(\boldsymbol{x}_n), \boldsymbol{\phi}(\boldsymbol{x}_m)\rangle$ and we can thus replace the product $\boldsymbol{\Phi}\boldsymbol{\Phi}^T$ with the Kernel $K(\boldsymbol{x}, \boldsymbol{x}')$. Doing this yields an objective, or energy function that only depends on the kernel $K(\boldsymbol{x}, \boldsymbol{x}')$,

$$J(\boldsymbol{a}) = \frac{1}{2}\boldsymbol{a}^T\boldsymbol{K}\boldsymbol{K}\boldsymbol{a} - \boldsymbol{a}^T\boldsymbol{K}\boldsymbol{t} + \frac{1}{2}\boldsymbol{t}^T\boldsymbol{t} + \frac{\lambda}{2}\boldsymbol{a}^T\boldsymbol{K}\boldsymbol{a} \tag{195}$$

We can then solve for $\boldsymbol{a}$ or simply substitute the expression (193) we found for $\boldsymbol{\beta}$ in (192), to find

$$\boldsymbol{a} = (\boldsymbol{K} + \lambda\boldsymbol{I})^{-1}\boldsymbol{t} \tag{196}$$

Once we have this expression (note that so far everything can be expressed with the Kernel), the classifier simply reads as

$$y(\boldsymbol{x}) = \boldsymbol{\beta}^T\boldsymbol{\phi}(\boldsymbol{x}) = \boldsymbol{a}^T\boldsymbol{\Phi}\boldsymbol{\phi}(\boldsymbol{x}) = \sum_{n=1}^{N} a_n\boldsymbol{K}(\boldsymbol{x}_n, \boldsymbol{x}) \tag{197}$$

11 Support Vector Machines (SVM)

Support vector machine, which are also known as sparse kernel machines, or Maximal margin classifiers, extend the notion of separating hyperplane to find a separating plane that reduces as much as possible the "risk" of misclassifying new points. To achieve this, the separating plane is defined as the plane maximizing the distance to its closest point(s).

Using the discussion from section 7.1, we can then write the derivation of such maximal margin hyperplane as

$$\underset{\mathbf{w},\mathbf{b}}{\text{argmax}} \left\{ \frac{1}{\|\mathbf{w}\|} \min_n \left[t_n(\mathbf{w}^\mathbf{T}\phi(\boldsymbol{x}_\mathbf{n}) + \mathbf{b}) \right] \right\} \tag{198}$$

That is we look for the $\boldsymbol{w}$ and b (which define the plane) such that the distance of the closest point (minimization on $\boldsymbol{x}_n$) to the plane is maximized.

Formulation (198) is invariant under any rescaling of the pair $(\mathbf{w}, \mathbf{b})$. I.e as $\mathbf{w}$ and/or b both appear at the numerator and at the denominator, replacing those quantities by $(\alpha\mathbf{w}, \alpha\mathbf{b})$ does not affect the solution. In fact any such pair, for any α will be a valid solution.

In order to simplify the formulation, we can therefore fix the value of α and decide to take this value equal to the value satisfying

$$t_n(\mathbf{w}^\mathbf{T}\phi(\boldsymbol{x}_\mathbf{n}) + \mathbf{b}) = \mathbf{1} \tag{199}$$

for the closest point $\boldsymbol{x}_n$. By assumption, any other point, must satisfy $t_n(\mathbf{w}^\mathbf{T}\phi(\boldsymbol{x}_\mathbf{n}) + \mathbf{b}) \geq \mathbf{1}$, and the problem reads as

$$\begin{aligned} \underset{\mathbf{w},\mathbf{b}}{\text{argmin}} \quad & \frac{1}{2}\|\mathbf{w}\|^\mathbf{2} \\ \text{subject to} \quad & t_n(\mathbf{w}^\mathbf{T}\phi(\boldsymbol{x}_\mathbf{n}) + \mathbf{b}) \geq \mathbf{1} \end{aligned} \tag{200}$$

Note that we have used $\underset{x}{\text{argmax}}\|\boldsymbol{x}\|^{-1} = \underset{x}{\text{argmin}}\|\boldsymbol{x}\|$. Problem (200) is a quadratic program that is convex, hence, has a single basin of attraction and can be solved efficiently.

To solve this problem, we write is as an unconstrained formulation (Lagrangian) by introducing multipliers, λ_i for each constraints (see section 6.2)

$$L(\mathbf{w}, \mathbf{b}, \lambda) = \frac{\mathbf{1}}{\mathbf{2}}\|\mathbf{w}\|^{\mathbf{2}} - \sum_{\mathbf{n=1}}^{\mathbf{N}} \lambda_{\mathbf{n}} \left\{\mathbf{t_n}(\mathbf{w^T}\phi(\boldsymbol{x_n}) + \mathbf{b}) - \mathbf{1}\right\} \tag{201}$$

To find the solutions for $\mathbf{w}$ and b, we compute the derivatives of L with respect to those variables and set the derivatives to 0. From this, we get

$$\mathbf{w} = \sum_{n=1}^{N} \lambda_n t_n \phi(\boldsymbol{x}_n) \tag{202}$$

$$0 = \sum_{n=1}^{N} \lambda_n t_n \tag{203}$$

Now recall that the general form of the classifier we are after is given by

$$y(\boldsymbol{x}) = \mathbf{w^T}\boldsymbol{\phi}(\boldsymbol{x}) + \mathbf{b} \tag{204}$$

If we substitute the expression for $\mathbf{w}$ that we derived from the zero derivatives in the expression of the classifier, we get

$$\begin{aligned} y(\mathbf{x}) &= \sum_{n=1}^{N} \lambda_n t_n \langle \boldsymbol{\phi}(\boldsymbol{x}_n), \boldsymbol{\phi}(\boldsymbol{x}) \rangle + b \\ &= \sum_{n=1}^{N} \lambda_n t_n k(\boldsymbol{x}, \boldsymbol{x}_n) + b \end{aligned} \tag{205}$$

Now besides the maximization of the margins, the other key idea of SVM is the fact that we don't need to keep track of all the training samples in (205). Indeed from Theorem 10, we know that the solution $(\mathbf{w}, \mathbf{b})$ of problem (200) has to satisfy the conditions

$$\lambda_n \geq 0 \tag{206}$$

$$t_n y(\boldsymbol{x}_n) - 1 \geq 0 \tag{207}$$

$$\lambda_n (t_n y(\boldsymbol{x}_n) - 1) = 0 \tag{208}$$

From the last condition in particular, we can see that for every training sample, $n = 1, \ldots, N$, either we have $\lambda_n = 0$ or we have $t_n y(\boldsymbol{x}) - 1 = 0$. The second case corresponds to points that are located the closest to the maximal margin hyperplane as the distance to each points to the hyperplane is always lower bounded by

$$\frac{t_n y(\boldsymbol{x}_n)}{\|\mathbf{w}\|} \geq \frac{1}{\|\mathbf{w}\|} \tag{209}$$

In other words, for every training point that appears in (205):

- Either that point does not contribute to the expression (205) of the classifier (as $\lambda_n = 0$)
- Or it is lying on the margins.

Using this idea to simplify (205), if we introduce the notation $\mathcal{S}$ to denote the set of "support vectors", that is the points that are lying on the margins, or located the closest to the plane, the expression of the classifier reduces to

$$y(\boldsymbol{x}) = \sum_{n \in \mathcal{S}} \lambda_n t_n k(\boldsymbol{x}, \boldsymbol{x}_n) + b \tag{210}$$

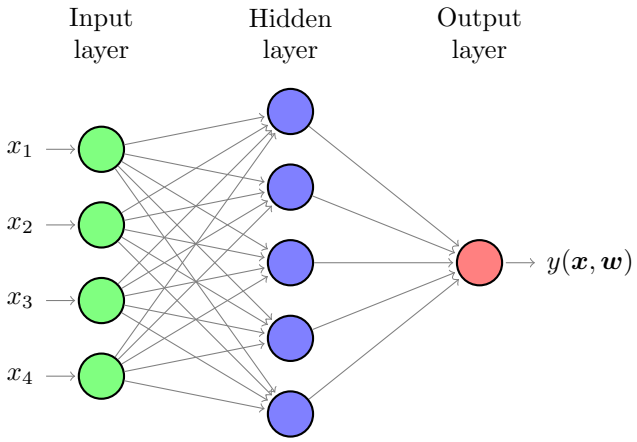

Figure 3: Graphical representation of a one layer neural network. The arrows represent multiplication by the weights w_{ij}. The hidden units (shown in blue) represent the application of the activation function.

Furthermore, we can use condition (208) for the support vectors, (i.e. those for which we have $t_n(y(\boldsymbol{x}_n) - 1) = 0$) to write b. indeed note that substituting (210) into condition (208) gives

$$t_n \left(\sum_{n \in \mathcal{S}} \lambda_n t_n k(\boldsymbol{x}, \boldsymbol{x}_n) + b \right) = 1, \quad \text{for every } \boldsymbol{x}_n \in \mathcal{S} \tag{211}$$

To obtain a robust estimate of b, we can then just sum all those equation and divide by N_S. This gives

$$b = \frac{1}{N_{\mathcal{S}}} \sum_{n \in \mathcal{S}} \left(t_n - \sum_{n' \in \mathcal{S}} \lambda_{n'} t_{n'} k(\boldsymbol{x}', \boldsymbol{x}_n) \right) \tag{212}$$

12 Neural Networks

The general formulation of a (two hidden layers) neural network is as follows,

$$y_k = y_k(\boldsymbol{x}, \boldsymbol{w}) = \sigma^{(2)} \left(\sum_{j=1}^{M} w_{kj}^{(2)} \sigma^{(1)} \left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) + w_{k0}^{(2)} \right) \tag{213}$$

When training neural networks, we usually don't take all the training samples into account but rather rely on a small subset of those training samples at each gradient iteration. These subsets are known as minibatches. There are three main approaches at training a network

- Batch gradient descent. Here all the training samples are taken into account and the weight are updated only after all the error has been evaluated at all the training samples (Batch gradient descent is just the regular gradient descent)
- Stochastic gradient descent (SGD). Here, the algorithm computes the error for a given training sample and update the weights immediately after (i.e the weights are updated for each training sample)
- Finally, the tradeoff between batch gradient descent and SGD, MiniBatch gradient descent updates the weight after evaluating the error/loss on a subset of the training samples.

Among the possible regularization approaches for neural networks, the most popular are the following

- Activity regularization. For a given network, we call activations, the outputs of the activation functions, i.e $\sigma^{(\ell)}(\cdot)$. Placing a regularizer on the activation will result in training a network in which only a fraction of the neurons fire at the same time (at least for data similar to the training data).

- Weight regularization. Here we add a penalty (e.g. ℓ_2 or ℓ_1 norm) on the weights.

- Early stopping. When training neural networks, one often observe a decrease in both the training and validation error for the first iterations. Then, after a sufficient number of gradient iterations, the validation error starts increasing [6]. One way to avoid overfitting is then to get the values of the parameters at each iteration and return the values of the parameters that give the lowest validation error (we stop the iterations when no improvement on the validation error has been obtained for some iterations)

- Dropout, which shares some similarities with activity regularizationin fact it can be considered as some sort of subset selection alternative to the ℓ_1 or ℓ_2 penalties used in activity regularization, can be considered as optimizing over a family of "sparse" networks. When training a network with dropout, we select some random subset of the neurons and only update the weights associated to those neurons. Mathematically, when performing SGD, we thus compute the forward pass (i.e the output of the network) by considering a sparse version of the original network [21]. We choose a probability p. Recall that for a traditional neural network, the activity of each unit can be defined as

$$z_i^{(\ell+1)} = \langle \boldsymbol{w}_i^{(\ell+1)}, \boldsymbol{y} \rangle + b_i^{(\ell+1)} \tag{214}$$
$$y_i^{(\ell+1)} = \sigma(z_i^{\ell+1}) \tag{215}$$

  With dropout propagation through the network now reads as

$$r_j^{(\ell)} \sim \text{Bernoulli}(p)^1 \tag{216}$$
$$\tilde{\boldsymbol{y}}^{(\ell)} = \boldsymbol{r}^{(\ell)} \odot \boldsymbol{y}^{(\ell)} \tag{217}$$
$$z_i^{(\ell+1)} = \boldsymbol{w}_i^{(\ell+1)} \tilde{\boldsymbol{y}}^{(\ell)} + b_i^{(\ell+1)} \tag{218}$$
$$y_i^{(\ell+1)} = f(z_i^{(\ell+1)}) \tag{219}$$

  So that we select only some of the activations y_j and only update the weights associated to those activations.

- Weight Sharing. The idea of weight sharing is especially meaningful in convolutional neural networks (CNN) where filters are applied to an image to extract information from this image. As the name indicates, weights sharing requires all the neurons within a particular subgroup to have identical weights. Weights sharing is implicitely present in convolutional neural. In those network, each neuron can be thought as computing a local average across the neighborhood of a pixel. The point of such averages is to extract features from a part of the image. Usually, when using convolutional neural networks, we are interested in detecting a particular object in the input images. Hence the features we are looking for should be the same regardless of the part of the image on which the filter is applied. AS a consequence, the weights are constants across each layers. Equality of the weights can be enforced in a hard (exact equality such as in CNNs) or in a soft way. Soft weight sharing is done by assuming a Gaussian distribution of the weights within a given group of neurons.

  In the simplest framework, we assume that all the weights w_i in the group $\mathcal{G}$ are all relatively close to some mean value μ (with variance σ^2) and we for example assume the distribution of those weights to be Gaussian

$$p(w_i) = \mathcal{N}(w_i|\mu, \sigma^2) \tag{220}$$

  We can make this model slightly more general by assuming that the weights have a high probability to take a few distinct values. In this case, we assume for example that the

distribution of those weights is given by a mixture of Gaussian, i.e.,

$$p(w_i) = \sum_{j=1}^{M} \pi_j \mathcal{N}(w_i|\mu_j, \sigma_j^2) \tag{221}$$

We then use this distribution as a regularization in the objective used to train the network. I.e, just as we did for the MAP and MLE estimators, we want the training step to select weights that have a high probability in the model (221). This corresponds to maximizing (221) over the weights, or (as we did for the MAP and MLE) to minimizing the negative log likelihood

$$\Omega(w) = -\log(\sum_{j=1}^{M} \mathcal{N}(w|\mu_j, \sigma_j^2)) \tag{222}$$

When there are several ($N_{\mathcal{G}}$) weights in the group $\mathcal{G}$, we simply sum over the contribution of each weight,

$$\Omega(\boldsymbol{w}) = -\sum_{i=1}^{N_{\mathcal{G}}} \log \left(\sum_{j=1}^{M} \pi_j \mathcal{N}(w_i|\mu_j, \sigma_j^2) \right) \tag{223}$$

The loss function that is minimized when training the network is then given by

$$\tilde{E}(\boldsymbol{w}) = E(\boldsymbol{w}) + \lambda \Omega(\boldsymbol{w}) \tag{224}$$

12.1 Backpropagation

Consider the two hidden layers NN of (213). The extension of this simpler model to L layers can read as

$$y_k = y_k(\boldsymbol{x}, \boldsymbol{w}) = \sigma^{(L)} \left(\sum_{j=1}^{M} w_{kj}^{(L)} \sigma^{(L-1)} \left(\dots \sigma^{(1)} \left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \right) \dots \right) + w_{k0}^{(L)} \right) \tag{225}$$

In order to train such a network, we minimize the loss

$$\sum_{n=1}^{N} \sum_{k=1}^{K} (y_k(\boldsymbol{x}_n) - t_k^n)^2 \tag{226}$$

if we put ourselves in a stochastic gradient descent framework, where we only consider a single sample at each iteration, we can drop the sum over n in (226) and consider the minimization

$$L = \sum_{k=1}^{K} (y_k(\boldsymbol{x}_n) - t_k^n)^2 \tag{227}$$

In order to apply one SGD iteration on this objective, and to update the weights

$$w_{ij}^{\ell} \leftarrow w_{ij}^{\ell} - \eta \frac{\partial L}{\partial w_{ij}^{\ell}} \tag{228}$$

we need to compute the expression of the derivatives $\frac{\partial L}{\partial w_{ij}^{\ell}}$. To do so, we will rely extensively on the chain rule.

Computing the derivative with the outer weights (the weights that are the closes to the output of the network) is easy. But computing the derivatives with respect to the most inner weights, i.e. w^1_{ij}, can be quite involved.

Backpropagation is an efficient way to compute the gradient of neural networks. First note that

$$\frac{\partial L}{\partial y_k} = 2(y_k - t_k) \tag{229}$$

We then want to compute all derivatives of the form

$$\frac{\partial L}{\partial w^\ell_{kj}} \tag{230}$$

For this, we use the chain rule a first time, splitting (230) into

$$\frac{\partial L}{\partial w^\ell_{kj}} = \frac{\partial L}{\partial a^\ell_k}\frac{\partial a^\ell_{kj}}{\partial w^\ell_{kj}} \tag{231}$$

that is we start from the weight we want to compute and we go up one step towards the output of the network by considering the quantitites

$$a^\ell_k = \sum_{j=1}^{M} w^\ell_{k,j} z^{\ell-1}_j \tag{232}$$

where $z^{\ell-1}_j = \sigma^{\ell-1}(\ldots)$.

Note that the second factor in (231) is simply $z^{\ell-1}_j = \sigma^{\ell-1}(\ldots)$ which is computed when we pass the data through the network. To get the first factor, we apply the chain rule again, going one more level towards the output of the network, y_k. We have

$$\frac{\partial L}{\partial a^\ell_k} = \sum_j \frac{\partial L}{\partial a^{\ell+1}_j}\frac{\partial a^{\ell+1}_j}{\partial a^\ell_k} \tag{233}$$

This last equation is the equation that really encodes backpropagation. To see this, first note that from (225), we have

$$a^{\ell+1}_j = \sum_m w^\ell_{jm} z_m \tag{234}$$

$$= \sum_m w^\ell_{jm} \sigma^\ell(a_m) \tag{235}$$

and we can thus write

$$\frac{\partial a^{\ell+1}_j}{\partial a^\ell_k} = (\sigma^\ell)' w^\ell_{jm} \tag{236}$$

Putting this back into (233), we get

$$\frac{\partial L}{\partial a^\ell_k} = \sum_j \frac{\partial L}{\partial a^{\ell+1}_j}(\sigma^\ell)' w^\ell_{jm} \tag{237}$$

Now let $\delta_k^\ell = \frac{\partial L}{\partial a_k^\ell}$. From (237), we can write

$$\delta_k^\ell = \sum_j \delta_j^{\ell+1} (\sigma^\ell)' w_{jm}^\ell \tag{238}$$

From (238), you see that the $\delta^{\ell+1}$ can be "backpropagated" through the network by multiplication with the weights, to obtain the δ^ℓ (we move from the output a^L to the most inner part of the network, a^1, hence the term "backpropagation"). Once we have computed all the δ^ℓ, the derivatives are simply given by using (231) and noting that $\frac{\partial a_k^\ell}{\partial w_{kj}^\ell} = z_j^{\ell-1}$ (Eq. (232)).

The "backpropagation" algorithm can thus read as follows

- First send the training data through the network and compute all the $z_j^\ell(\dots)$ which are the outputs of every neuron.
- Then compute all the intermediate derivatives $\left[\sigma^\ell(a^{\ell-1})\right]'$ a the activations $a^{\ell-1}$ given by your training point.
- Once you have all those values, start backpropagating the δ with

$$\delta^L = \frac{\partial L}{\partial a_k^L} = \frac{\partial (y_k - t_k)^2}{\partial y_k} \frac{\partial y_k}{\partial a_k^L} \tag{239}$$

$$= 2(y_k - t_k) \left[\sigma^L(a_k^L)\right]' \tag{240}$$

  and then

$$\delta_k^\ell = \sum_j \delta_j^{\ell+1} (\sigma^\ell)' w_{jm}^\ell \tag{241}$$

- Finally, once you have all the δ_k^ℓ, compute the derivatives as

$$\frac{\partial L}{\partial w_{kj}^\ell} = \frac{\partial L}{\partial a_k^\ell} z_j^\ell \tag{242}$$

13 Clustering

13.1 Hierarchical clustering

The simplest approach to clustering are the so-called hierarchical clustering algorithms which either start from a single cluster and then divide this cluster iteratively (*Divisive*) or start from the individual prototypes and then merge those prototypes with each other based on their relative similarity. We detail those algorithms below

13.1.1 Agglomerative Clustering

In *agglomerative clustering*, each of the prototypes are merged following a given criterion, until they form a simgle cluster. Depending on the deisred number of clusters, the proceudre

In each of those approaches, the combination of subclusters is motivated by the minimization of a particular notion of dissimilarity.

- In *Single Linkage* (SL), we merge at each step, we consider as dissimilarity the distance between the closest pair of points $x_i \in S_1$ and $x_j \in S_2$, and merge the two clusters that minimize this dissimiliarity,

$$d_{SL}(S_1, S_2) = \min_{\substack{x_i \in S_1 \\ x_j \in S_2}} d(x_i, x_j) \tag{243}$$

- In *Complete Linkage* (CL), the dissimilarity of any pair of clusters S_1 and S_2 is defined as the distance between the furthest pair of points

$$d_{CL}(S_1, S_2) = \max_{\substack{x_i \in S_1 \\ x_j \in S_2}} d(x_i, x_j) \tag{244}$$

- Finally, an tradeoff between the two approaches can be obtained by taking the dissimilarity between any two sub-clusters to be the average of the inter-cluster distances. This approach is known as *Group Average (GA)*

$$d_{GA}(S_1, S_2) = \frac{1}{N_{S_1} N_{S_2}} \sum_{i \in S_1} \sum_{j \in S_2} d(x_i, x_j) \tag{245}$$

  where we let N_{S_1} and N_{S_2} denote the number of prototypes in sub-cluster S_1 and S_2

The iterative merging of sub-clusters defines a dendrogram which can be cut (i.e the procedure can be stopped) at any level depending on the desired number of clusters. Each of the aforementioned approach also has an interpretation in terms of the diameter of each cluster.

13.1.2 Divisive Clustering

As the name indicates, *Divisive clustering* is based on recursively splitting a parent cluster C into two child subclusters C_1 and C_2 following a particular criterion. At each step, one cluster is chosen to be subdivided. This cluster is often the one with the largest diameter $\delta = \max_{i,j \in \mathcal{S}_\infty} d(\boldsymbol{x}_i, \boldsymbol{x}_j)$ or with the largest average dissimilarity,

$$\overline{d} = \frac{1}{N_C} \sum_{i,j \in C} d(\boldsymbol{x}_i, \boldsymbol{x}_j) \tag{246}$$

The division of the parent cluster can be done for example by applying K-means to that cluster with two centroids and returning the corresponding two subclusters, An alternative, due to Macnaughton Smith et al. relies on removing from the clusters all the points that have a sufficiently high dissimilarity with respect to all the other points in the original cluster. If we let C_0 to denote the parent cluster, the methods then defines the first subcluster C_1 by removing from C_0 the point with the largest average dissimilarity

$$x^* = \operatorname*{argmax}_x \frac{1}{N_{C_0} - 1} \sum_{x_j \in C_0} d(x_i, x) \tag{247}$$

Here N_{C_0} is used to denote the number of prorotypes in the parent cluster C_0. The first subcluster C_1 is thus defined as the singleton $C_1 = \{x\}$ and the algorithm proceeds by removing the next point with the highest average dissimilarity from $C_0 \setminus \{x^*\}$ and moving it to C_1. This continues until there is no more prototype in $C_2 = C_0 \setminus C_1$ that is more similar to the points in C_1 than it is to the points in C_2. I.e, the subdivision stops whenever there is no more y for which th difference

$$D_{C_2} - D_{C_1} = \frac{1}{|C_2| - 1} \sum_{x_j \in C_2} d(y, x_j) - \frac{1}{|C_1|} \sum_{x_j \in C_1} d(x_j, y) \tag{248}$$

is non negative.

13.2 K-means

K-means is one of the most popular and most straightforward clustering algorithm. In K-means the general idea is to find within the data, K clusters that minimize the pairwise distance between the prototypes belonging to the cluster. If we let $\mathcal{C}_1, \ldots, \mathcal{C}_K$ to denote the K clusters in which we want to group our prototypes $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N$, and use $\mathcal{C}$ to denote the mapping which associates

a cluster $k = 1, \dots, K$ to each prototype $\boldsymbol{x}_\ell$, that is if the cluster of $\boldsymbol{x}_\ell$ is $\mathcal{C}_k$, we write $\mathcal{C}(\boldsymbol{x}_\ell) = k$ or $\mathcal{C}(\boldsymbol{x}_\ell) = \mathcal{C}_k$. K-means minimizes the following objective

$$\min_{\mathcal{C}} \sum_{k=1}^{K} \sum_{(i,j)\in\mathcal{C}_k} \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2 \tag{249}$$

That is, we only take into account the intra-cluster distance (the distance between points belonging to the same cluster). The main advantage of K-means lies in the use of the Euclidean distance. By using the properties of this distance, one can in fact show that the cost (249) can be reduced to finding the clustering which minimizes the sum of the distances points of a same cluster and their center of mass. To see this, first write the Euclidean distance in terms of inner products,

$$\frac{1}{2}\sum_{k=1}^{K}\sum_{i\in\mathcal{C}_k}\sum_{j\in\mathcal{C}_k}\|x_i - x_j\|^2 = \frac{1}{2}\sum_{k=1}^{K}\sum_{i\in\mathcal{C}_k}\sum_{j\in\mathcal{C}_k}\langle x_i, x_i\rangle + \langle x_j, x_j\rangle - 2\langle x_i, x_j\rangle \tag{250}$$

$$= \frac{1}{2}\sum_{k=1}^{K}\sum_{i\in\mathcal{C}_k} 2\langle x_i, x_i\rangle N_k - \frac{1}{2}\sum_{k}\sum_{i\in\mathcal{C}_k} 2\langle x_i, \sum_{j\in\mathcal{C}_k} x_j\rangle \tag{251}$$

$$= \left(\sum_{k=1}^{K} N_k \sum_{i\in\mathcal{C}_k}\langle x_i, x_i\rangle - \sum_{k=1}^{K} N_k \sum_{i\in\mathcal{C}_k}\langle x_i, \sum_{j\in\mathcal{C}_k} x_j\rangle \frac{1}{N_k}\right) \tag{252}$$

$$= \sum_{k=1}^{K} N_k \sum_{i\in\mathcal{C}_k}\left(\langle x_i, x_i\rangle - \langle x_i, \sum_{j\in\mathcal{C}_k} x_j\rangle \frac{1}{N_k}\right) \tag{253}$$

$$\tag{254}$$

Then add and subtract the quantity $Q \equiv \sum_{k=1}^{K}\sum_{i\in\mathcal{C}_k}\frac{1}{N_k}\sum_{j\in\mathcal{C}_k}\langle \boldsymbol{x}_i, \boldsymbol{x}_j\rangle$,

$$\sum_{k=1}^{K} N_k \sum_{i\in\mathcal{C}_k}\left(\langle x_i, x_i\rangle - \langle x_i, \sum_{j\in\mathcal{C}_k} x_j\rangle \frac{1}{N_k}\right) + Q - Q \tag{255}$$

$$= \sum_{k=1}^{K} N_k \sum_{i\in\mathcal{C}_k}\left(\langle x_i, x_i\rangle - \langle x_i, \sum_{j\in\mathcal{C}_k} x_j\rangle \frac{1}{N_k} + \frac{1}{N_k^2}\langle \sum_{j\in\mathcal{C}_k} x_j, \sum_{j\in\mathcal{C}_k} x_j\rangle\right) \tag{256}$$

$$- \sum_{k=1}^{K}\sum_{i\in\mathcal{C}_k}\left(\frac{1}{N_k}\sum_{j\in\mathcal{C}_k}\langle x_j, x_i\rangle\right) \tag{257}$$

$$= \sum_{k=1}^{K} N_k \sum_{i\in\mathcal{C}_k}\left(\langle x_i, x_i\rangle - 2\langle x_i, \frac{1}{N_k}\sum_{j\in\mathcal{C}_k} x_j\rangle + \langle \frac{1}{N_k}\sum_{j\in\mathcal{C}_k} x_j, \frac{1}{N_k}\sum_{j\in\mathcal{C}_k} x_j\rangle\right) \tag{258}$$

$$= \sum_{k=1}^{K} N_k \sum_{i\in\mathcal{C}_k} \|x_i - \frac{1}{N_k}\sum_{j\in\mathcal{C}_k} x_j\|^2 \tag{259}$$

Formulation (259) is interesting because it gives us a heuristic to look for the optimal assignment. From (259) it indeed seems reasonable to start with a set of centroids, define the assignment by relating each prototype to its closest centroid and then update the centroid by defining them as the center of mass of each of the newly generated clusters. This idea is summarized below.

- *Assignment.* Assign each prototype to the cluster represented by the nearest centroid. I.e, for a given set of centroids, y_k, $k = 1, \dots, K$, if we let $\mathcal{C}(\boldsymbol{x}_\ell)$ to denote the cluster of the ℓ^{th} prototype $\boldsymbol{x}_\ell$, then $\mathcal{C}(\boldsymbol{x}_k)$ is updated as follows

$$\mathcal{C}(\boldsymbol{x}_\ell) \leftarrow \min_k \|\boldsymbol{c}_k - \boldsymbol{x}_\ell\| \tag{260}$$

- *Update of the centroids.* Once the cluster of each point has been determined, the main advantage of K-means is that the use of the Euclidean distance enables a straightforward update of each centroid as the center of mass of the newly formed clusters:

$$\boldsymbol{c}_\ell = \frac{1}{N_\ell} \sum_{k \in \mathcal{C}_\ell} \boldsymbol{x}_k \tag{261}$$

It is worth noting that K-means might sometimes return empty clusters (some of the centroids might be moved towards a region of space without any prototypes). In this case, one can simply restart the algorithm by replacing this centroid with for example the prototype $\boldsymbol{x}^*$ from the dataset that is located the furthest away from all the other (well defined) centroids. Proceeding like this will eliminate the point $\boldsymbol{x}^*$ that contributes the most to the error (259).

An alternative approach could be to restart the algorithm by keeping the non zero clusters and resampling a centroid from the cluster that has the largest intra-cluster SSE. This will also reduce the error (259).

The whole process can obviously be repeated when multiple empty clusters are returned.

Several initialization approaches have been proposed for K-means. The popular ones are listed below (see [18] for more details).

- *Random partitioning.* The approach divides the dataset in K distinct clusters chosen at random.
- The *Forgy* or *Lloyd-Forgy* approach [13, 5]. The method picks K feature vectors at random to define the centroids and assigns all the remaining feature vectors to the nearest centroid. There is not recalculation of the centroids after each assignment.
- *K-means++.* The first centroid is selected at random. The next centroid is selected as the point that is the farthest from the currently selected. The selection continues until K centroids are defined.
- *MacQueeen approach* [15]. Choose K instances of the database at random. Assign, following some order, the rest of the dataset to each of the centroids. After each assignment, recalculate the centroids.
- *Kaufman approach* [9]. The Kaufman initialization is a deterministic initialization which places the centroids in the areas where there is a higher density of prototypes $\boldsymbol{x}_i$. The Kaufman initialization is shown in Fig. 1.

Set $\overline{\boldsymbol{x}}_0$ to be the median feature vector (i.e the one that is the most cetrally located);
Set $\mathcal{S} = \{\overline{\boldsymbol{x}}_0\}$ and let $\mathcal{S}^c$ denote the set of remaining feature vectors.;
for *all the remaining feature vectors* $\boldsymbol{x}_i \in \mathcal{S}^c$ **do**
    Compute $\overline{d}_i \equiv \min_{k \in \mathcal{S}} d(\overline{\boldsymbol{x}}_k, \boldsymbol{x}_i)$;
    Set $C_{i,j} \equiv \max(\overline{d}_i - d(\boldsymbol{x}_j, \boldsymbol{x}_i), 0)$
end
Choose the next centroid $\overline{\boldsymbol{x}}_{k+1}$ as the one that maximizes $\max_i C_{ij}$;
if *there are* K *points in* $\mathcal{S}$ **then**
    Stop
end
else
    Set $\mathcal{S} \leftarrow \mathcal{S} \cup \overline{x}_{k+1}$ and go back to step 1.
end

Algorithm 1: Kaufman initialization

Convergence of K-means is studied in [14, 3]. Among the initialization procedures,

13.3 K-medoid

K-medoid is the restriction of K-means to centroids that are part of the original set of prototypes, $\mathcal{D}$. K-medoid is more flexible in the sense that it can be applied with any given distance, but the price to pay is a higher computational cost. Indeed, more general distances are not guaranteed to lead to losses, such as (259), for which the optimal centroid for a given assignment can be obtained directly as the center of mass of each cluster. In the K-medoid framework, the update of the centroids requires screening all the points within a given cluster and selecting the one that minimizes its distance to all the others, thus increasing the cost from $O(N)$ to $O(N^2)$. The algorithm is summarized by the two steps below.

- *Initialization.* Choose K points at random as the initial centroids $\{m_k\}_{k=1}^K$.
- *Assignment.* Divide the dataset according to those centroids by assigning each point as

$$\mathcal{C}(\boldsymbol{x}_i) = \underset{k}{\operatorname{argmin}}\, D(\boldsymbol{x}_i, \boldsymbol{m}_k) \tag{262}$$

- *Update of the centroids.* Update the centroids by choosing them among the feature vectors as

$$m_k \leftarrow \underset{\boldsymbol{x} \mid \mathcal{C}(\boldsymbol{x})=k}{\operatorname{argmin}} \sum_{\boldsymbol{x} \mid \mathcal{C}(\boldsymbol{x})=k} D(\boldsymbol{x}, \boldsymbol{x}_k) \tag{263}$$

K-medoid then iterates over the two steps above until the assignement stop changing.

13.4 The EM Algorithm

Both K-means and K-medoid relies on hard assignment. I.e., a point is classified as belonging or not to a given cluster. Such a hard approach leads to a lack of robustness in the sense that it does not come with a clear quantification of the uncertainty associated to the classification of each prototype. The expectation maximization (EM) approach works in a similarly way but defines the clusters based on probability distributions. The advantage of turning to probability distributions, is that each assignment now comes with a level of confidence encoded by the probabilities. In the EM algorithm, the cluster of a point is defined as the probability distribution that gives the highest probability of occurence for that point.

14 Linear latent variable models

In all the models discussed below, we are interested in finding representations of the form $x(\theta)$ for a dataset $\{\boldsymbol{x}_k\}_{k=1}^N$. That is we would like to understand whether most of the data can be generated from a small number of meaningful parameters.

As usual in machine learning, most of the models that we consider below have a statistical interpretation that falls within either of the Bayesian or Frequentist framework.

14.1 Factor Analysis (FA)

The simplest latent variable model assumes that the prototypes can be represented from the linear combination of a few (common) sources of variation,

$$\boldsymbol{x}_i = \boldsymbol{W}\boldsymbol{\theta}_i, \quad i = 1, \dots, N, \quad \boldsymbol{x}_i \in \mathbb{R}^D \tag{264}$$

$\boldsymbol{W}$ is a matrix of size $D \times K$ with $K \ll N$. Assuming a sufficiently low dimensional latent representation however is not enough to make the decomposition uniquely identifiable. In particular,

for a given set of prototypes $\{\boldsymbol{x}_i\}_{i=1}^N$, and any rotation matrix $\boldsymbol{R}$[2] one always has

$$\boldsymbol{x}_i = \boldsymbol{W}\boldsymbol{R}\boldsymbol{R}^T\boldsymbol{\theta} = \boldsymbol{W}\boldsymbol{\theta} \tag{265}$$

which already gives two possible decompositions for $\boldsymbol{x}_i$. The Factor Analysis model also leads to scaling ambiguities as for any $\alpha \in \mathbb{R}$, one can always write

$$\boldsymbol{x}_i = (\alpha\boldsymbol{W})(\alpha^{-1}\boldsymbol{\theta}_i) \tag{266}$$

To ensure unique recovery of the latent factors together with the factor loading matrix $\boldsymbol{W}$, we need to enforce additional constraints that will prevent the ambiguities discussed above to occur. The most commonly used solutions are listed below and detailed in the subsequent subsections.

- Requiring $\boldsymbol{W}$ to be orthonormal, i.e., $\boldsymbol{W}^T\boldsymbol{W} = \boldsymbol{I}$. This is the approach followed by Principal Component Analysis (PCA) which is discussed in section 14.2.
- Require $\boldsymbol{W}$ to be lower triangular. I.e in this case, the first feature in $\boldsymbol{x}_i$ only depends on the first factor θ_1, the second feature is a linear combination of the first two latent factors, θ_1, θ_2 and so on. This approach usually also requires the weights from the factor loading matrix to be positive.
- Use of non-Gaussian priors for the latent factors. This, as we will see, leads to Independent Component Analysis (ICA).
- Consider sparsity promoting priors on the weights. An alternative to the lower triangular factor loading matrix is to avoid pre-specifying the location of the non zero entries in $\boldsymbol{W}$ but to encourage the entries to be zero by adding an ℓ_1 regularization term.

The idea underlying Factor Analysis can be used inside a statistical framework. If we consider measurements that are generated from the latent model following a Gaussian distribution

$$\boldsymbol{x}_i = \boldsymbol{\mu} + \boldsymbol{W}\boldsymbol{z}_i + \varepsilon_i \tag{267}$$

Where the $\varepsilon_i \in \mathbb{R}^d$ follows a multivariate Normal distribution with covariance $\boldsymbol{\Psi}$,

$$p(\boldsymbol{x}_i) = \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{d/2}|\boldsymbol{\Psi}|^{1/2}} \exp\left\{-\frac{1}{2}(\boldsymbol{x}_i - \boldsymbol{\mu} - \boldsymbol{W}\boldsymbol{z}_i)^T\boldsymbol{\Psi}^{-1}(\boldsymbol{x}_i - \boldsymbol{\mu} - \boldsymbol{W}\boldsymbol{z}_i)\right\} \tag{268}$$

and follow a Frequentist approach, therefore focusing exclusively on the (log) likelihood, we have

$$-\log(p(\boldsymbol{x}_i|\boldsymbol{\mu}, \boldsymbol{W}, \boldsymbol{z}_i)) = -\log(\mathcal{N}(\boldsymbol{\mu} + \boldsymbol{W}\boldsymbol{z}_i, \Psi)) \tag{269}$$

$$= -\log(-\frac{1}{(2\pi)^{d/2}|\boldsymbol{\Psi}|^{1/2}} \exp(-(\boldsymbol{x}_i - \boldsymbol{\mu} - \boldsymbol{W}\boldsymbol{z}_i)^T\boldsymbol{\Psi}^{-1}(\boldsymbol{x}_i - \boldsymbol{\mu} - \boldsymbol{W}\boldsymbol{z}_i))) \tag{270}$$

$$= \log((2\pi)^{d/2}|\boldsymbol{\Psi}|^{1/2}) + (\boldsymbol{x}_i - \boldsymbol{\mu} - \boldsymbol{W}\boldsymbol{z}_i)^T\boldsymbol{\Psi}^{-1}(\boldsymbol{x}_i - \boldsymbol{\mu} - \boldsymbol{W}\boldsymbol{z}_i)) \tag{271}$$

In the particular case where $\boldsymbol{\Psi} = \sigma^2\boldsymbol{I}$ (which corresponds to independent features), we get

$$-\log(p(\boldsymbol{x}_i|\boldsymbol{\mu}, \boldsymbol{W}, \boldsymbol{z}_i)) = d\log(2\pi) + \frac{d}{2}\log(\sigma) + \frac{1}{\sigma^2}\|\boldsymbol{x}_i - \boldsymbol{\mu} - \boldsymbol{W}\boldsymbol{z}_i\|^2 \tag{272}$$

When $\sigma \to 0$, minimizing (272) (i.e maximizing the log likelihood), corresponds to the Factor Analysis minimization problem discussed above,

$$\min_{\boldsymbol{W}, \boldsymbol{z}_i} \|\boldsymbol{x}_i - \boldsymbol{\mu} - \boldsymbol{W}\boldsymbol{z}_i\|^2 \tag{273}$$

[2]A *rotation matrix* is an orthogonal matrix with unitary determinant, i.e. $\boldsymbol{R}^T\boldsymbol{R} = \boldsymbol{R}\boldsymbol{R}^T = \boldsymbol{I}$

Minimizing (273) can thus be understood as neglecting the σ, or assuming no uncertainty on the prototypes. Without additional assumption on either $\boldsymbol{W}$ or $\boldsymbol{z}_i$, solving this problem is hopeless. As we already mentioned, one possibility to make the problem better posed is to add assumptions on $\boldsymbol{W}$. On top of those assumptions, one can take a Bayesian perspective on the problem and consider a prior on $\boldsymbol{z}_i$. Multiplying our likelihood with such a prior gives us the possibility to average out the dependence on the $\boldsymbol{z}_i$ (in the sense that $p(\boldsymbol{x}_i) = \int p(\boldsymbol{x}_i|\boldsymbol{z}_i)p(\boldsymbol{z}_i)\, d\boldsymbol{z}_i$). If we take a Gaussian prior for the $\boldsymbol{z}_i$ for example, $p(\boldsymbol{z}_i) = \mathcal{N}(\boldsymbol{z}_i|\boldsymbol{\mu}_0, \boldsymbol{\Psi}_0)$, we can average out the dependence as

$$p(\boldsymbol{x}_i|\boldsymbol{\theta}) = \int \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{W}\boldsymbol{z}_i + \boldsymbol{\mu}, \boldsymbol{\Psi})\mathcal{N}(\boldsymbol{z}_i|\boldsymbol{\mu}_0, \boldsymbol{\Psi}_0)\, d\boldsymbol{z}_i \tag{274}$$

Using (54), we can then write this probability distribution as

$$p(\boldsymbol{x}_i|\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{x}_i|\boldsymbol{W}\boldsymbol{\mu}_0 + \boldsymbol{\mu}, \boldsymbol{\Psi} + \boldsymbol{W}\boldsymbol{\Psi}_0\boldsymbol{W}^T) \tag{275}$$

Note that is not an MAP estimation as we did not use priors on our parameters $\boldsymbol{W}, \boldsymbol{\mu}, \boldsymbol{\Psi}$ and $\boldsymbol{\mu}_0, \boldsymbol{\Psi}_0$. Moreover, as already explained, this procedure removes the dependence in the latent factors $\boldsymbol{z}_0$ but does not solve the ill posedness of the problem (i.e we still have too many parameters). The next sections address this last difficulty.

14.2 Principal Component Analysis (PCA)

Principal Component Analysis is the restriction of FA to orthonormal factor loading matrices $\boldsymbol{W}$. In principal component analysis, one is interested in finding a subset of vectors $\{v_\ell\}_{\ell=1}^L$ that capture most of the information present in the dataset $\{\boldsymbol{x}_k\}_{k=1}^N$. Mathematically, we therefore want to find a model of the form $\{\boldsymbol{\mu}, \boldsymbol{W}, \boldsymbol{\lambda}\}$ where $\boldsymbol{W}$ has orthonormal columns, i.e. $\boldsymbol{W}^*\boldsymbol{W} = \boldsymbol{I}$, and where any point from $\{\boldsymbol{x}_k\}_{k=1}^N$ can read as

$$\boldsymbol{x}_k \approx \boldsymbol{\mu} + \boldsymbol{W}\boldsymbol{\lambda} \tag{276}$$

To find such a model, a natural approach is to minimize the sum of the deviations between the prototypes and their approximations $\boldsymbol{\mu} + \boldsymbol{W}\boldsymbol{\lambda}$. This can be done by minimizing the ℓ_2, as follows

$$\min_{\mu,\lambda,\boldsymbol{V}} \sum_{i=1}^N \|\boldsymbol{x}_i - \boldsymbol{\mu} - \boldsymbol{W}\boldsymbol{z}_i\|^2 \tag{277}$$

Again, formulation (277) can be viewed as following from the maximization of the log-likelihood provided that the prototypes $\boldsymbol{x}_i$ follow a Normal distribution with mean $\boldsymbol{\mu} + \boldsymbol{W}\boldsymbol{z}_i$.

Solving problem (277) for $\boldsymbol{z}_i$ and μ gives

$$\mu = \frac{1}{N}\sum_{i=1}^N \boldsymbol{x}_i \tag{278}$$

$$\boldsymbol{z}_i = \boldsymbol{W}^*(\boldsymbol{x}_i - \boldsymbol{\mu}) \tag{279}$$

Given expressions (278) and (279), the minimization problem (277) can then be written as

$$\min_{\boldsymbol{W}} \sum_{i=1}^N \|\boldsymbol{x}_i - \overline{\boldsymbol{x}} - \boldsymbol{W}\boldsymbol{W}^*(\boldsymbol{x}_i - \overline{\boldsymbol{x}})\|^2 \tag{280}$$

The matrix $\boldsymbol{W}\boldsymbol{W}^*$ that appears in this expression is a projection matrix (i.e a matrix for which $\boldsymbol{P}^2 = \boldsymbol{P}$). $\boldsymbol{W}\boldsymbol{W}^*$ in particular maps every point from the dataset $\{\boldsymbol{x}_i\}_{i=1}^N$ onto the subspace spanned by the columns of $\boldsymbol{W}$. Indeed, note that we have

$$\boldsymbol{W}\boldsymbol{W}^*\boldsymbol{x}_i = \sum_{\ell=1}^L \boldsymbol{w}_\ell\langle \boldsymbol{w}_\ell, \boldsymbol{x}_i\rangle \tag{281}$$

Now if we develop the norm in (280) and use $\sum_{k=1}^{N} w_k^2 = \langle \boldsymbol{w}, \boldsymbol{w} \rangle$, we have

$$\min_{\boldsymbol{W}} \sum_{i=1}^{N} \|\boldsymbol{x}_i - \overline{\boldsymbol{x}} - \boldsymbol{W}\boldsymbol{W}^*(\boldsymbol{x}_i - \overline{\boldsymbol{x}})\|^2 \tag{282}$$

$$= \min_{\boldsymbol{W}} \sum_{i=1}^{N} \langle \boldsymbol{x}_i - \boldsymbol{W}\boldsymbol{W}^*\boldsymbol{x}_i, \boldsymbol{x}_i - \boldsymbol{W}\boldsymbol{W}^*\boldsymbol{x}_i \rangle \tag{283}$$

$$= \min_{\boldsymbol{W}} \sum_{i=1}^{N} \langle \boldsymbol{x}_i, \boldsymbol{x}_i \rangle + \sum_{i=1}^{N} \langle \boldsymbol{W}\boldsymbol{W}^*\boldsymbol{x}_i, \boldsymbol{W}\boldsymbol{W}^*\boldsymbol{x}_i \rangle - 2\sum_{i=1}^{N} \langle \boldsymbol{W}\boldsymbol{W}^*\boldsymbol{x}_i, \boldsymbol{x}_i \rangle \tag{284}$$

$$= \min_{\boldsymbol{W}} \sum_{i=1}^{N} \|\boldsymbol{x}_i\|^2 + \sum_{i=1}^{N} \boldsymbol{x}_i^* \boldsymbol{W}\boldsymbol{W}^*\boldsymbol{x}_i - 2\sum_{i=1}^{N} \boldsymbol{x}_i^* \boldsymbol{W}\boldsymbol{W}^*\boldsymbol{x}_i \tag{285}$$

Between (284) and (285), we used $\boldsymbol{W}^*\boldsymbol{W} = \boldsymbol{I}$. The first term in the last line can be treated as a constant as it does not depend on $\boldsymbol{W}$ (i.e we can disregard it). We are thus left with

$$\min_{\boldsymbol{W}} -\sum_{i=1}^{N} \boldsymbol{x}_i^* \boldsymbol{W}\boldsymbol{W}^*\boldsymbol{x}_i \tag{286}$$

which can be written as a maximization problem

$$\max_{\boldsymbol{W}} \sum_{i=1}^{N} \boldsymbol{x}_i^* \boldsymbol{W}\boldsymbol{W}^*\boldsymbol{x}_i \tag{287}$$

If we were looking for the first component only, the problem would reduce to

$$\max_{\boldsymbol{w}_1} \sum_{i=1}^{N} \boldsymbol{x}_i^* \boldsymbol{w}_1 \boldsymbol{w}_1^* \boldsymbol{x}_i \tag{288}$$

$$= \max_{\boldsymbol{w}_1} \boldsymbol{w}_1^* \sum_{i=1}^{N} \boldsymbol{x}_i \boldsymbol{x}_i^* \boldsymbol{w}_1 \tag{289}$$

$$= \max_{\boldsymbol{w}_1} \boldsymbol{w}_1^* \boldsymbol{X}^* \boldsymbol{X} \boldsymbol{w}_1 \tag{290}$$

In the expression above, we again used $\boldsymbol{X}$ to denote the matrix encoding the prototypes $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathbb{R}^D$,

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}_1^T \\ \vdots \\ \boldsymbol{x}_N^T \end{bmatrix} \tag{291}$$

For any given matrix $\boldsymbol{X}^*\boldsymbol{X}$, problem (290) will return the eigenvector of $\boldsymbol{X}^*\boldsymbol{X}$ corresponding to the largest eigenvalue (why?). To find the first column of $\boldsymbol{W}$, we can thus take the eigenvalue decomposition of $\boldsymbol{X}^*\boldsymbol{X}$ and retain the first eigenvector. In fact this approach continues to work for any number of components. That is to say, if we want to find the best subspace of dimension q, it suffices to take the eigenvalue decomposition of $\boldsymbol{X}^*\boldsymbol{X}$ and to retain the first L largest eigenvectors. This idea is summarized below

Theorem 11 (see for example [2]). *Suppose that we are given a dataset* $\{\boldsymbol{x}_i\}_{i=1}^N$ *and that we want to find an orthonormal set of* L *basis vectors* $\boldsymbol{w}_j$ *that best approximate the data, or equivalently, that minimize the average reconstruction error*

$$e(\boldsymbol{W}, \boldsymbol{z}) \equiv \frac{1}{N} \sum_{i=1}^{N} \|\boldsymbol{x}_i - \boldsymbol{W}\boldsymbol{z}_i\|^2 \tag{292}$$

Then the optimal solution for $\boldsymbol{W}$ *is obtained by building the empirical covariance matrix,* $\hat{\boldsymbol{\Sigma}} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}_i \boldsymbol{x}_i^T$*, and setting* $\boldsymbol{W} = \boldsymbol{V}_L$ *where* $\boldsymbol{V}_L$ *contains the* L *eigenvectors with largest eigenvalues of* $\hat{\boldsymbol{\Sigma}}$*. The columns of* $\boldsymbol{V}_L$ *are often called principal directions.*

Equivalently one could work with the singular value decomposition (SVD) of $\boldsymbol{X}$, and retain the first L right singular vectors obtained from the matrix $\tilde{\boldsymbol{V}}$ in this SVD $\boldsymbol{X} = \tilde{\boldsymbol{U}}\boldsymbol{\Sigma}\tilde{\boldsymbol{V}}^*$. We then have $\boldsymbol{X}^* \approx \tilde{\boldsymbol{V}}_L(\boldsymbol{\Sigma}\tilde{\boldsymbol{U}}^*)_L$ where $(\boldsymbol{\Sigma}\tilde{\boldsymbol{U}}^*)_L$ is used to denote the first L rows of $\boldsymbol{\Sigma}\tilde{\boldsymbol{U}}^*$ and encodes the λ_k. In other words, to solve the PCA problem for a set of prototypes $\{\boldsymbol{x}_k\}_{k=1}^N$,

(i) Center the dataset, $\boldsymbol{x}_k \leftarrow \boldsymbol{x}_k \frac{1}{N}\sum_{k=1}^{N} \boldsymbol{x}_k$

(ii) Build the matrix $\boldsymbol{X}^* = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]$ from the centered prototypes

(iii) Compute the SVD of $\boldsymbol{X}$, $\boldsymbol{X} = \tilde{\boldsymbol{U}}\boldsymbol{\Sigma}\tilde{\boldsymbol{V}}^*$

(iv) Define the matrix $\boldsymbol{V}$ from the first L columns of $\boldsymbol{V}$

(v) Define the weight vectors $\boldsymbol{\lambda}_k$ from the k^{th} column of the matrix obtained after retaining the first L rows of $\boldsymbol{\Sigma}\boldsymbol{U}^*$.

14.2.1 Probabilistic PCA

Just as for Factor Analysis, one can interpret PCA through the lens of Frequentist statistics, as a particular MLE for which the variance of the noise would be arbitrarily small and hence, the $\log(\sigma)$ negligible. When the noise ε_i cannot be neglected, or more generally if we want to follow a more rigorous statistical approach, one can again consider a prior on the latent factors, $\boldsymbol{z}_i$. We then get as in FA

$$p(\boldsymbol{x}_i|\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{W}\boldsymbol{\mu}_0 + \boldsymbol{\mu}, \boldsymbol{\Psi} + \boldsymbol{W}\boldsymbol{\Psi}_0\boldsymbol{W}^T) \tag{293}$$

If we consider N observations and take $\Psi_0 = \boldsymbol{I}$ and $\boldsymbol{\Psi} = \sigma^2\boldsymbol{I}$ and $\boldsymbol{\mu}_0 = 0$, writing down the log-likelihood gives (you can verify this as an exercise)

$$\log(p(\boldsymbol{x}_i|\boldsymbol{\theta})) = -\frac{d}{2}\log(|\boldsymbol{W}\boldsymbol{W} + \sigma^2\boldsymbol{I}|) - \frac{1}{2}\sum_{i=1}^{N} \boldsymbol{x}_i \left(\boldsymbol{W}\boldsymbol{W}^T + \sigma^2\boldsymbol{I}\right)^{-1} \boldsymbol{x}_i \tag{294}$$

You can check that the maxima of the likelihood are given by

$$\boldsymbol{W}^* = \boldsymbol{V}(\boldsymbol{\Lambda} - \sigma^2\boldsymbol{I})^{1/2}\boldsymbol{R} \tag{295}$$

Where $\boldsymbol{R}$ is any orthogonal matrix (see the discussion on the uniqueness of $\boldsymbol{W}$ in section 14.1) and $\boldsymbol{V}$ and $\boldsymbol{\Lambda}$ follow from the eigenvalue decomposition of the empirical covariance matrix $\boldsymbol{S} = \frac{1}{N}\sum_{i=1}^{N} \boldsymbol{x}_i\boldsymbol{x}_i^T = \frac{1}{N}\boldsymbol{X}^T\boldsymbol{X}$. The approach in probabilistic PCA consists resolving the ambiguity on $\boldsymbol{W}$ by setting $\boldsymbol{R} = \boldsymbol{I}$. We then get $\boldsymbol{W}^* = \boldsymbol{V}(\boldsymbol{\Lambda} - \sigma^2\boldsymbol{I})^{1/2}$. In particular, this recovers the traditional PCA solution when $\sigma \to 0$ (note that this last approach can hence be considered as slightly more rigorous as the case $\sigma = 0$ is well defined.)

It is however also worth noting that in this slightly more rigorous framework (statistically speaking), we do not recover an orthogonal matrix when $\sigma^2 > 0$.

14.3 Independent Component Analysis (ICA)

In Independent Component Analysis (ICA), one is interested in finding a description of the data by means of independent factors. ICA has led to several interesting applications to source separation as well as biomedical signal processing, with a particular emphasis on EEG source separation.

Consider two signals $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ of length T (we use $s_{1,t}$ to denote the t^{th} sample from $\boldsymbol{s}_1$) that are mixed together through the matrix $\boldsymbol{W} \in \mathbb{R}^{2\times 2}$. The mixings are encoded in two distinct recordings that are measured at microphones x_1 and microphone x_2,

$$\begin{bmatrix} x_{1,t} \\ x_{2,t} \end{bmatrix} = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \end{bmatrix} \begin{bmatrix} s_{1,t} \\ s_{2,t} \end{bmatrix} + \begin{bmatrix} \varepsilon_{1,t} \\ \varepsilon_{2,t} \end{bmatrix} \tag{296}$$

Following this model, at any time, the first and second microphones thus record the signals

$$x_{1,t} = w_{11}s_{1,t} + m_{12}s_{2,t} + \varepsilon_{1,t} \tag{297}$$
$$x_{2,t} = w_{21}s_{1,t} + w_{22}s_{2,t} + \varepsilon_{2,t} \tag{298}$$

Here the weights w_{ij} can be understood as encoding the distances from the microphones to the speakers. If the matrix $\boldsymbol{W}$ is given, it is of course easy to recover the sources by, for example ℓ_2 norm minimization. Perhaps more surprisingly, is the fact that the recovery of the sources remain possible even when the mixing coefficients are unknown.

The original application of ICA was to the so-called coktail party problem, where one wants to recover independent speeches from their mixtures. Among the notorious successes of ICA, one must also mention the demixing of brain activities from EEG recordings on the scalp, as well as feature extraction (e.g. the identification of shocks from frequent fluctuations in finance [1]).

To solve problem (296), it is possible to take a Maximum Likelihood viewpoint on the problem. Let $\boldsymbol{x} = \boldsymbol{W}\boldsymbol{s} = f(\boldsymbol{s})$. Then note that in the scalar case, if we use p_x to denote the probability density function (see definition 9) and P_x to denote the cumulative mass function (see definition 7), we can always write

$$p_x(x) = \frac{d}{dx}P_x(x) = \frac{d}{dx}P_s(f^{-1}(x)) = \frac{ds}{dx}\frac{d}{ds}P_s(s) = \frac{ds}{dx}p_s(s) \tag{299}$$

as $p_x(x)$ is a probability density (i.e it has to be non negative) we set

$$p_x(x) = p_s(s)\left|\frac{ds}{dx}\right| \tag{300}$$

When considering multiple dimensions such as in ICA, the change of variable formula (300) extends as

$$p_x(\boldsymbol{x}) = p_s(\boldsymbol{s})\left|\det\left(\frac{\partial \boldsymbol{s}}{\partial \boldsymbol{x}}\right)\right| \tag{301}$$

Where $\frac{\partial \boldsymbol{s}}{\partial \boldsymbol{x}}$ is the Jacobian matrix (see (17)). With this in mind, we can write

$$p_x(\boldsymbol{W}\boldsymbol{s}) = p_s(\boldsymbol{s})\left|\det(\boldsymbol{W}^{-1})\right| \tag{302}$$

If we let $\boldsymbol{V} = \boldsymbol{W}^{-1}$, in particular we get

$$p_x(\boldsymbol{x}) = p_x(\boldsymbol{W}\boldsymbol{s}) = p_s(\boldsymbol{V}\boldsymbol{x})\left|\det(\boldsymbol{V})\right| \tag{303}$$

The maximum likelihood approach looks for the parameters that are maximizing the probability of observing the data (i.e. the prototypes $\boldsymbol{x}_i$, $i = 1, \ldots, N$). Taking the log likelihood (note that the probability to get $\boldsymbol{x}$ only depends on $\boldsymbol{V}$ and $\boldsymbol{x}$ from (303)) gives

$$\log(p(\boldsymbol{x}|\boldsymbol{V})) = N\log|\det(\boldsymbol{V})| + \sum_{s=1}^{D}\sum_{t=1}^{N} p_s(\boldsymbol{v}_s^T\boldsymbol{x}_t) \tag{304}$$

So far, we have introduced an MLE approach to Independent Component Analysis but it is not clear why the maximum likelihood estimator should work.

In ICA we want to recover independent components s_1 and s_2. By the Central Limit Theorem (Theorem 9), we know that a sum of (a sufficiently large number of) identically distributed, independent random variables tend to have a Gaussian distribution. In particular, from this Theorem, the random source s_i can be expected to "look" less Gaussian than any combination $a_{11}s_1 + a_{12}s_2$.

When considering the set of weighted sum of the form $\boldsymbol{v}^T\boldsymbol{x} = \boldsymbol{v}^T\boldsymbol{W}\boldsymbol{s}$, the one vector $\boldsymbol{v}$ that should therefore correspond to a maximally non Gaussian random variable is the one that returns only one of the (non gaussian) independent sources (as opposed to a linear combination of those sources). But this optimal vector $\boldsymbol{v}$ leading to the maximally independent sources (or maximally non Gaussian signal) precisely corresponds to a row (lets say the first one) of $\boldsymbol{A}^{-1}$. Indeed, for this row, which we label $\boldsymbol{v} = (\boldsymbol{A}^{-1})_{1,:}$, we have $\boldsymbol{v}^T\boldsymbol{x} = (\boldsymbol{A}^{-1})_{1,:}\boldsymbol{A}\boldsymbol{s} = s_1$.

When we want to recover independent sources from linear combinations of those sources, maximizing the "non Gaussian" nature of $\boldsymbol{w}^T\boldsymbol{z}$ over the vectors $\boldsymbol{v}$ thus seems to be a relatively good idea. How can we then maximize this "non-Gaussianity" of $\boldsymbol{w}^T\boldsymbol{z}$? and how does this connect to the MLE which is used in the FastICA algorithm of for example scikit-learn?

To understand this, consider the following measure of (non-)gaussianity known as negentropy,

$$\text{negentropy}(z) = \mathcal{H}(\mathcal{N}(\mu, \sigma^2)) - \mathcal{H}(z) \tag{305}$$

In the definition above, $\mathcal{H}(X)$ represents the entropy of a random variable X (you can view it as a measure of the uncertainty associated to the random variable),

$$\mathcal{H}(X) = -\sum_{k=1}^{K} p(X = k)\log_2 p(X = k) \tag{306}$$

We also used $\mu = \mathbb{E}\{z\}$ as well as $\sigma^2 = \text{var}[z]$. But what is probably more important in (305) is that the negentropy can also be understood the deviation of a random variable to the closest Gaussian distribution.

Following from the discussion above, to find the vector w such that the $w^T z$ is as far as possible from a Gaussian distribution, one could thus maximize the negentropy, i.e., for measurements $j = 1, \ldots, K$,

$$\max_{\boldsymbol{V}} \sum_{j=1}^{K} \mathcal{H}(\mathcal{N}(\mu_j, \sigma_j^2)) - \mathcal{H}(z_j) \tag{307}$$

Often, when we want to recover independent component, a good approach is to center and whiten the data (see [7]). Centering means that we replace the measurements $\boldsymbol{x}$ by the difference $\boldsymbol{x} - \mathbb{E}\boldsymbol{x}$. Centering the measurements also corresponds to centering the sources as we have $\boldsymbol{x} - \mathbb{E}\boldsymbol{x} = \boldsymbol{A}\boldsymbol{z} - \boldsymbol{A}\mathbb{E}\boldsymbol{z} = \boldsymbol{A}(\boldsymbol{z} - \mathbb{E}\boldsymbol{z})$. After estimating the centered principal components, one can simply get the original principal components by adding back the mean $\boldsymbol{A}^{-1}\mathbb{E}\boldsymbol{x}$.

Another transformation that is often applied for computational purposes, is the whitening of the data. Whitening the data means that we apply a linear transform to the measurements vector $\boldsymbol{x}$ to make the components of this vector uncorrelated, i.e, whitening transforms $\boldsymbol{x}$ into a vector $\tilde{\boldsymbol{x}}$ satisfying $\mathbb{E}\{\tilde{\boldsymbol{x}}\tilde{\boldsymbol{x}}^*\} = \boldsymbol{I}$.

Whitening can be done by first computing the eigenvalue decomposition of the sample covariance $\mathbb{E}\left\{\boldsymbol{x}\boldsymbol{x}^T\right\} = U\lambda U^T$ and then transforming $\boldsymbol{x}$ into $\tilde{\boldsymbol{x}}$ as

$$\tilde{\boldsymbol{x}} = \boldsymbol{U}\boldsymbol{\Lambda}^{-1/2}\boldsymbol{U}^T\boldsymbol{x} \tag{308}$$

Whitening also has a consequence on the mixing matrix $\boldsymbol{A}$. Indeed, the whitening turns the measurements into

$$\tilde{\boldsymbol{x}} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T\boldsymbol{A}\boldsymbol{s} \tag{309}$$

which is therefore equivalent to changing replacing the mixing matrix $\boldsymbol{A}$ with $\tilde{\boldsymbol{A}}$ defined as $\tilde{\boldsymbol{A}} = \boldsymbol{U}\boldsymbol{\Lambda}^{-1/2}\boldsymbol{U}^T$. From the whitening transformation (308), we get

$$\mathbb{E}\left\{\tilde{\boldsymbol{x}}\tilde{\boldsymbol{x}}^T\right\} = \tilde{\boldsymbol{A}}\mathbb{E}\left\{\boldsymbol{s}\boldsymbol{s}^T\right\}\tilde{\boldsymbol{A}}^T = \tilde{\boldsymbol{A}}\tilde{\boldsymbol{A}}^T = \boldsymbol{I} \tag{310}$$

The mixing matrix is now orthogonal,

$$\begin{aligned}\tilde{\boldsymbol{A}}^T\tilde{\boldsymbol{A}} &= \boldsymbol{U}\boldsymbol{\Lambda}^{-1/2}\boldsymbol{U}^T\boldsymbol{A}\boldsymbol{A}^T\boldsymbol{E}\boldsymbol{\Lambda}^{-1/2}\boldsymbol{E}^T &(311)\\ &= \boldsymbol{U}\boldsymbol{\Lambda}^{-1/2}\boldsymbol{U}^T\mathbb{E}\left\{\boldsymbol{x}\boldsymbol{x}^T\right\}\boldsymbol{E}\boldsymbol{\Lambda}^{-1/2}\boldsymbol{E}^T &(312)\\ &= \boldsymbol{I}. &(313)\end{aligned}$$

So in particular, whitening means that we now only need to estimate $n(n-1)/2$ parameters instead of the initial n^2.

Once the data has been centered and whitened, the first term in (307) is constant, and maximizing the negentropy thus reduces to the maximization

$$\max_{\boldsymbol{V}} \sum_j -\mathcal{H}(z_j) = \sum_j \mathbb{E}\log(p(x_j)) \tag{314}$$

which is precisely the MLE. In other words, we can hope the MLE to recover the sources because it corresponds to maximizing non gaussianity of $\boldsymbol{w}^T\boldsymbol{x}$ and we know from the Central limit Theorem that the more independent sources s_i we have in a linear combination $\boldsymbol{w}^T\boldsymbol{A}\boldsymbol{s}$, the more Gaussian this combination will be.

So the intuition behind the MLE is that is corresponds to maximizing the deviation between the z_j and the closest Gaussian random variable.

14.3.1 Fast ICA

We now follow the notations in [16] and introduce the FastICA algorithm. Let $G(z_j) = -\log(p(z_j))$ and $g(z_j) = \frac{d}{dz_j}G(z_j)$. Maximizing the log likelihood while asking for an orthogonal matrix A (note that when $\boldsymbol{A}$ is orthogonal, the inverse is given by $\boldsymbol{A}^T$), lead to the following minimization

$$\min_{\boldsymbol{v}} \mathbb{E}\left\{G(\boldsymbol{v}^T\boldsymbol{x}) + \lambda(1-\boldsymbol{v}^T\boldsymbol{v})\right\} \tag{315}$$

We simply replaced the sum in (303) by the expectation and neglect the first term which is constant when $\boldsymbol{V}$ is orthogonal (determinant is 1).

The gradient and Hessian are given by

$$\begin{aligned}\nabla f(\boldsymbol{v}) &= \mathbb{E}\left\{\boldsymbol{x}g(]bsv^T\boldsymbol{x})\right\} - 2\lambda\boldsymbol{v} &(316)\\ \nabla^2 f(\boldsymbol{v}) &= \mathbb{E}\left\{\boldsymbol{x}\boldsymbol{x}^T g'(\boldsymbol{v}^T\boldsymbol{x})\right\} - 2\lambda\boldsymbol{I} &(317)\end{aligned}$$

To minimize a function $f(x)$, recall that the Newton steps are defined by

$$x \leftarrow x - (\nabla^2 f(x))^{-1}\nabla f(x) \tag{318}$$

Now if we make the simplification

$$\mathbb{E}\left\{\boldsymbol{x}\boldsymbol{x}^T g'(\boldsymbol{v}^T\boldsymbol{x})\right\} \approx \mathbb{E}\left\{\boldsymbol{x}\boldsymbol{x}^T\right\}\mathbb{E}\left\{g'(\boldsymbol{v}^T\boldsymbol{x})\right\} = \mathbb{E}\left\{g'(\boldsymbol{v}^T\boldsymbol{x})\right\} \tag{319}$$

in the case of problem (315), the Newton iterations can be written as

$$\boldsymbol{v} \leftarrow \boldsymbol{v} - \frac{\mathbb{E}\left\{\boldsymbol{x}g(\boldsymbol{v}^T\boldsymbol{x}) - 2\lambda\boldsymbol{v}\right\}}{\mathbb{E}\left\{g'(\boldsymbol{v}^T\boldsymbol{x})\right\} - 2\lambda} \tag{320}$$

The expectation can be computed by Monte Carlo methods. Once the iterations have converged, one can define the optimal $\boldsymbol{v}^*$ as $\boldsymbol{v}/\|\boldsymbol{v}\|$ to get the unit vector.

In the developments above we have assumed that the distribution $p(z_j)$ was known. In practice, as long as the distribution of the sources is not Gaussian it seems [16] that it is not critical for the shape of this distribution to be exactly known. Possible choices are the Laplace distribution or other smoother super-Gaussian distribution although it seems that in practice, it is common to just use $G(z) = \sqrt{z}$ or $G(z) = \log\cosh z$

14.4 From one component to multiple components

One approach to extend the algorithm given above to multiple components v_j is to run instances of that algorithm for each v_j in parallel. The issue with such an approach is that we might end up converging to the same vector, i.e. $\boldsymbol{v}_j = v$ for all j. In this case we obviously do not recover the inverse $\boldsymbol{A}^{-1}$.

To prevent this, one possibility is to compute the component in series, and ensure that they are orthogonal to each other (i.e decorrelated) through a Gram-Schmidt process. The idea of such a procedure is as follows. Assuming that we have computed components w_1 to w_{p-1}, the p^{th} components is then computed through the one components algorithm above combined with the following two decorrelation steps (which are applied after every Newton step (320))

$$\boldsymbol{w}_p \leftarrow \boldsymbol{w}_p - \sum_{j=1}^{p-1} \boldsymbol{w}_p^T \boldsymbol{w}_j \boldsymbol{w}_j \tag{321}$$

$$\boldsymbol{w}_p \leftarrow \frac{\boldsymbol{w}_p}{\|\boldsymbol{w}_p\|} \tag{322}$$

Finally, note that ICA does not always work for non Gaussian sources. Moreover, a main difficulty in ICA lies in the fact that we usually do not have information on the statistical nature of the sources. In particular, natural signals, such as images, often exhibit some correlation. An extreme example of such correlation is shown in Fig. 4. When signals are correlated, ICA will not be able to recover the original sources.

15 Manifold Learning

In the previous sections, we have considered some of the popular linear dimensionality reduction models (a.k.a linear latent variable models). In both ICA and PCA, the mapping between the latent space (i.e. the space of the latent variables) and the space in which the prototypes are living, can be encoded by a linear map. In this chapter, we will extend the discussion to non linear models. In particular, we will discuss methods that focus on learning the structure underlying the data even when this structure is different from the usual hyperplane.

In some datasets, although the relation between the prototypes might be lost when mapping those prototypes onto a linear subspace (for example through PCA or ICA), those prototypes might still be distributed according to a relatively simple (non linear) structure embedded in $\mathbb{R}^n$. We call such structure a manifold.

Definition 20 (Manifold [20])**.** *A manifold $\mathcal{M}$ is a metric space (that is a space together with a metric d on the space) such that for $\boldsymbol{x} \in \mathcal{M}$, there exists some neighborhood U of $\boldsymbol{x}$ and some integer $n \geq 0$ such that U is homeomorphic (there exists a continuous bijective mapping between the two objects) to $\mathbb{R}^n$.*

A few popular manifold benchmarks are highlighted in Fig. 6.

To better understand the interest of non linear dimensionality reduction compared to the linear latent variable models, consider Fig. 5. In this figure, we compare some of the non linear

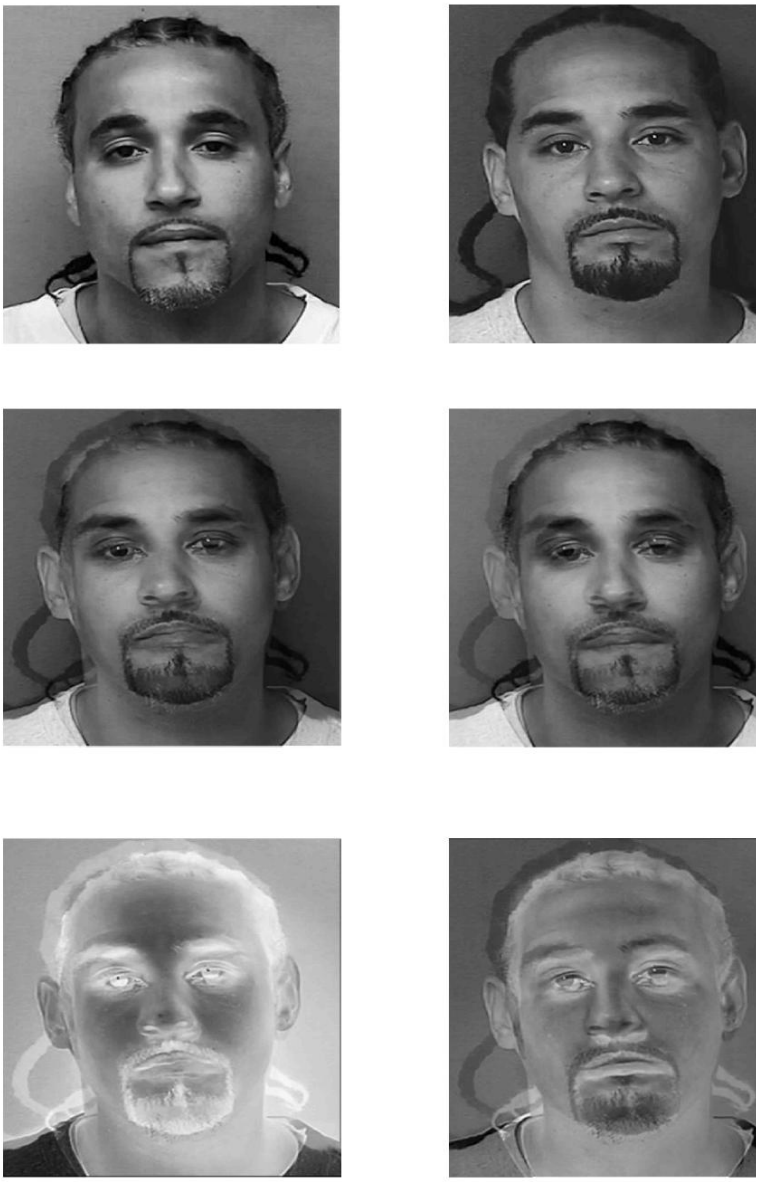

Figure 4: Example of correlated images ($\rho = 0.4$) for which ICA is unable to recover the sources. From top to bottom: Original (source) images, superposition obtained by mixing the images through the mixing coefficients $A_{11} = 1, A_{12} = 2, A_{21} = 2, A_{22} = 1.5$ and reconstruction through FastICA. Richard A. Jones (right) endured 17 years of wrongful incarceration before being finally freed from prison when Ricky L. Amos (left) was finally recognized to be the unique responsible of a robbery commited outside a Walmart in Roeland Park, Kansas. Source: CNN

dimensionality reduction methods that we will discuss below against Principal Component Analysis. The first row in the figure corresponds to applying non linear manifold learning methods, as well as PCA to the original S-shaped manifold from scikit-learn. The second row, corresponds to applying the same methods to a modified version of this manifold, i.e. if x_1, x_2, x_2 are the coordinate of a point on the original S manifold, we obtain the modified manifold as $x_1 = x_1 - 1$, $x_2 = 1.5x_2$ and $x_3 = 0.5x_3$. The modifier manifold is thus "shorter and fatter". When applying a principal component analysis to this second manifold, the PCA plane will be horizontal and the projection onto the principal plane will superimpose several of the manifold parts. The result is a complete destruction of the manifold structure.

Non linear dimensionality reduction methods are thus especially useful when considering datasets that are not distributed on a plane, but exhibit some form of (lower dimensional) structure. In such situations, the non linear methods might be able to capture such structure, while typical linear models such as PCA will not.

Among the methods presented in this chapter, *Multidimensional scaling*, as well as *Sammon's mapping* (section 15.1) as well as *Isomap* (section 15.2) can be considered as distance preservation methods. By distance preservation, we mean that the methods reduce dimensionality of the data by using the preservation of pairwise distances as a the main criterion. If a lower dimensional representation can be built in a way that preserves pairwise distances. then the geometric structure (global shape or local neighborhood) of the underlying manifold is preserved. In the non linear case, it is usually very difficult to preserve distances (there are many different ways to define non linear manifold).

Locally linear embedding (section 15.3) and (Kohonen) *Self Organizing maps* (section 15.4) on the other hand, are considered as preserving topology of the data. By topology of a manifold, we mean the *neighborhood relationships between subregions of the manifold*. The idea of topology preserving approaches is that pairwise distances are often too constraining or gives too much information. I.e. distances depend not only on the manifold but also on the chosen embedding of the manifold. Additional arguments in favor of the use of topology in dimensionality reduction can be found in [12]. Comparative information between distances are often enough to characterize a manifold. Neighborhood relations are intrinsic to the manifold whereas distances are defined on the whole ambient space and are not particular to the manifold itself. Topology is however more difficult to understand than pairwise distances. *How can we characterize topology of the manifold?*. One way is to associate the points to the vertices of a graph and encode proximity of the points by means of edge weights. *Kohonen self organizing maps* rely on a predefined lattice (build as a preliminary step to the method) whether *Locally linear embedding* and *Laplacian eigenmaps* rely on an iterative update of the lattice structure (i.e the lattice is updated while the method is running).

It is common to compare non linear latent variable models on some known manifold benchmarks. Here we will use the *S curve*, the *severed sphere* and the *Swiss roll* as those models are readily available in `scikit-learn`. Those three models are shown in Fig. 6 below.

Before discussing each of the dimension reduction approaches listed above, it is also worth noting that one of these methods, *multidimensional scaling (MDS)* cannot handle non linear embedding at least in its classical metric formulation. However, it is usually considered as the antecendent of non linear distance preserving methods [12] whence our choice to discuss this method in this chapter.

15.1 Multidimensional scaling

The Term *Multidimensional scaling* [10, 11] refers to a family of methods which construct a lower dimensional representation of the dataset from information on interpoint distances. MDS has been widely used in the humanities such as psychology, economics or sociology in which it is particularly interesting because it can be used with a notion of similarity between points. Unlike Self Organizing maps or principal curves and surfaces, MDS does not require interpoint distances but can be applied from a similarity measure only. This was then used as a quantitative approach at separating concepts.

For a given dataset $\{\boldsymbol{x}_i\}_{i=1}^N \subseteq \mathbb{R}^p$, we let d_{ij} to denote the distance between any two points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ in the dataset,

$$d_{ij} = \|\boldsymbol{x}_i - \boldsymbol{x}_j\| \tag{323}$$

The classical multidimensional scaling approach preserves inner products instead of distances. I.e in classical multidimensional scaling, we start with similarities s_{ij}. If not given similarities explicitely, we use pairwise inner products. The *classical stress function* is thus defined as

$$E_C(\boldsymbol{z}_1, \dots, \boldsymbol{z}_N) = \sum_{i,j}(s_{i,j} - \langle \boldsymbol{z}_i - \overline{\boldsymbol{z}}, \boldsymbol{z}_j - \overline{\boldsymbol{z}}\rangle) \tag{324}$$

Where the similarity is thus often given by the centered inner products $s_{ij} \equiv \langle \boldsymbol{x}_i - \overline{\boldsymbol{x}}, \boldsymbol{x}_j - \overline{\boldsymbol{x}}\rangle$. It is naturally also possible to consider a multidimensional scaling formulation which would look for points $\boldsymbol{z}_1, \dots, \boldsymbol{z}_N \in \mathbb{R}^k$ that minimize the (metric) *stress function*

$$E_M(\boldsymbol{z}_1, \dots, \boldsymbol{z}_N) = \sum_{i \neq j}(d_{ij} - \|\boldsymbol{z}_i - \boldsymbol{z}_j\|)^2 \tag{325}$$

In this case, the minimization is known as *least squares* or *Kruskal-Shephard* scaling. Moreover, if the distances are Euclidean, they can be converted to centered inner product as follows. Let $d_{i,j}^2 = \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2$, then we have

$$d_{i,j}^2 = \|\boldsymbol{x}_i - \overline{\boldsymbol{x}}\|^2 + \|\boldsymbol{x}_j - \overline{\boldsymbol{x}}\|^2 - 2\langle \boldsymbol{x}_i - \overline{\boldsymbol{x}}, \boldsymbol{x}_j - \overline{\boldsymbol{x}}\rangle. \tag{326}$$

Now let $\boldsymbol{M} = \frac{1}{N}\mathbf{1}\mathbf{1}^*$ and $B_{ij} = -\frac{d_{ij}^2}{2}$. The matrix $\boldsymbol{B}$ can read as

$$\tilde{\boldsymbol{S}} = (\boldsymbol{I} - \boldsymbol{M})\boldsymbol{B}(\boldsymbol{I} - \boldsymbol{M}) \tag{327}$$

Where $(\tilde{\boldsymbol{S}})_{i,j} = \langle \boldsymbol{x}_i - \overline{\boldsymbol{x}}, \boldsymbol{x}_j - \overline{\boldsymbol{x}}\rangle$.

A variation of this scaling which minimizes a reweighted verison of (325) is known as *Sammon* mapping.

$$E_S(\boldsymbol{z}_1, \dots, \boldsymbol{z}_N) = \sum_{i \neq j} \frac{(d_{i,j} - \|\boldsymbol{z}_i - \boldsymbol{z}_j\|)^2}{d_{ij}} \tag{328}$$

The classical formulation (324) is particularly attractive not only because inner products are easy to compute but also because Classical MDS minimizes the same criterion as PCA which was discussed earlier. Juste like PCA, classical MDS relies on a simple generative model $\boldsymbol{z} = \boldsymbol{W}\boldsymbol{x}$ where $\boldsymbol{W} \in \mathbb{R}^{d \times N}$ and is such that $\boldsymbol{W}^T\boldsymbol{W} = \boldsymbol{I}$. For a given dataset $\{\boldsymbol{x}_i\}$, let $\boldsymbol{Z} = [\boldsymbol{z}_1, \dots, \boldsymbol{z}_d]$ and $\boldsymbol{X} = [\boldsymbol{x}_1, \dots, \boldsymbol{x}_N]$. The matrix encoding the inner products (a.k.a *Gram matrix*) then reads as

$$\begin{aligned} \boldsymbol{S} &= \boldsymbol{Z}^T\boldsymbol{Z} && (329)\\ &= (\boldsymbol{W}\boldsymbol{X})^T(\boldsymbol{W}\boldsymbol{X}) && (330)\\ &= \boldsymbol{X}^T\boldsymbol{X} && (331) \end{aligned}$$

Eqs. 329 to (331) show that the latent variables can be found in a very convenient way by simply computing the eigenvalue decomposition of the matrix $\boldsymbol{S}$ encoding the inner products.

$$\begin{aligned} \boldsymbol{S} &= \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^T && (332)\\ &= (\boldsymbol{\Lambda}^{1/2}\boldsymbol{U})^T(\boldsymbol{\Lambda}^{1/2}\boldsymbol{U}^T) && (333) \end{aligned}$$

From this eigenvalue decomposition, the embedding is computed by retaining the first p eigenvalues (Note that as we are in a N-dimensional space, considering a dataset of size $m > N$, at most N eigenvalues in $\boldsymbol{\Lambda}$ are non zero),

$$\hat{\boldsymbol{Z}} = \boldsymbol{I}_{p \times N}\boldsymbol{\Lambda}^{1/2}\boldsymbol{U}^T \tag{334}$$

In (334), the general idea is summarized by Algorithm 2 below (the algorithm appears in the version below in [12]).

if *available data consists of points* $\boldsymbol{x}_i$ **then**
    Stack them into a matrix $\boldsymbol{Y}$ then center them, compute the pairwise inner products $\boldsymbol{S} = \boldsymbol{Y}^T\boldsymbol{Y}$ and go to third step;
else if *available data consists of pairwise Euclidean distances* **then**
    Square the distances and build the matrix $\boldsymbol{B}$ (see discussion above);
    Perform double centering of $\boldsymbol{B}$ to get $\tilde{\boldsymbol{S}}$ following (327).;
Compute the eigenvalue decomposition $\tilde{\boldsymbol{S}} = \boldsymbol{U}\boldsymbol{\Lambda}\boldsymbol{U}^*$;
The p-dimensional representation is obtained by computing the product

$$\hat{\boldsymbol{Z}} = \boldsymbol{I}_{p\times N}\boldsymbol{\Lambda}^{1/2}\boldsymbol{U}^* \tag{335}$$

;

Algorithm 2: Multidimensional Scaling

15.2 Isometric feature mapping (ISOMAP)

Isometric feature mappings (ISOMAP) [23] is the simplest non linear dimensionality reduction approach that uses graph distances as an approximation to geodesic distances [12]. In fact, the original ISOMAP follows from replacing the Euclidean distance in Classical multidimensional scaling by a graph distance. The complete Isometric feature mapping algorithm has three steps [23]

- During the first step, the method determines which points are neighbors on the manifold based on pairwise distances between points in the original/input space. The two most common approaches are to consider the points lying in a given Euclidean ball around each point $\boldsymbol{x}_i$, or to simply consider the neighborhood to be the K nearest neighbors. The method then builds a graph $\mathcal{G}$ by connecting connecting neighbouring points with edges of weights $w_{ij} = d(i,j)$.
- The second step estimates the pairwise geodesic distances $d_{\mathcal{M}}(i,j)$ of the prototypes by computing the shortest path distances in the graph $\mathcal{G}$.
- Finally, the last step applies classical Multidimensional scaling to the matrix of graph distances $D_{i,j} = (d_{\mathcal{G}}(i,j))$

The lower dimensional representation generated through ISOMAP is shown on the sphere, S-curves and Swiss Roll in Fig. 8

15.3 Locally linear embedding

Locally linear embedding is a topology preserving method which is originally due to Roweis and Saul [19]. In LLE the idea is to define a lower dimensional embedding but keep the local affine structure of the data. We say that LLE is based on conformal mappings [12]. A *conformal* or *biholomorphic* map is mapping that preserves local angles. The first step in LLE is to determine which angles to keep when defining the lower dimensional representation of the data. For this, LLE first finds the K-nearest neighbors to each point $\boldsymbol{x}_i$ from the original dataset (other techniques are possible).

Locally linear embedding then relies on the assumption that the manifold underlying the data is locally (i.e at the level of the local K-neighborhood) linear (which seems a reasonable assumption when the dataset is large enough and not too noisy). Following this assumption, LLE then replaces each point $\boldsymbol{x}_i$ by a linear combination of its K neighbors. After computing the neighborhoods, the second step in LLE then computes the linear combination of the K eighbors that best approximates every given point $\boldsymbol{x}_i$ from the dataset,

$$\min_{w} \|\boldsymbol{x}_i - \sum_{k \in \mathcal{N}(\boldsymbol{x}_i)} w_{i,k} \boldsymbol{x}_k\|^2. \tag{336}$$

Here we let $\mathcal{N}(\boldsymbol{x}_i)$ to denote the k nearest neighbors of $\boldsymbol{x}_i$. The low dimensional representation is then given by

$$\min_{\boldsymbol{z}} \|\boldsymbol{z}_i - \sum_{k \in \mathcal{N}(\boldsymbol{z}_i)} w_{i,k} \boldsymbol{z}_k\|^2. \tag{337}$$

The low dimensional representation thus maintains the local neighborhoods. Eq. (337) is equivalent to solving the matrix equation

$$\mathrm{Tr}\left[\boldsymbol{Z} - \boldsymbol{W}\boldsymbol{Z})^T(\boldsymbol{Z} - \boldsymbol{W}\boldsymbol{Z})\right] = \mathrm{Tr}\left[\boldsymbol{Z}^T(\boldsymbol{I} - \boldsymbol{W})(\boldsymbol{I} - \boldsymbol{W})\boldsymbol{Z}\right] \tag{338}$$

for which the solution is given explicitely by the eigenvectors of the sparse symmetric PSD matrix $\boldsymbol{M} = (\boldsymbol{I} - \boldsymbol{W})^T(\boldsymbol{I} - \boldsymbol{W})$. The best embedding of dimension p is then given by the lowest p eigenvectors. In fact the first eigenvector is usually discarded by LLE because it corresponds to the trivial $\mathbf{1}$ eigenvector encoding the free translation mode. When discarded, this eigenvector enforces a zero mean embedding.

15.4 Self Organizing maps

The idea underlying Self Organizing map goes back to the work of von der Malsburg in 1973. The idea which was then developed as a way to better understand the visual cortex then remained under the radar until the 80's, when Kohonen came with a simplification of the ideas of von der Malsburg and provided an easy implementation. Because of their relative simplicity, SOMs are used in a variety of areas including time series prediction and data visualization.

Self organizing maps are sometimes build upon the idea of Principal component analysis by extending these ideas to non linear data.

In classical quantization, a smaller set of prototypes is used to approximate (i.e to fit) a larger set of points from the original space. However, the prototypes are free to move independently of each other. The idea of SOMs is similar except that in SOM all prototypes are connected to each other through a lattice (i.e the prototypes are living on a 2D lattice) also called *Constrained topological map*. Each time a prototype is updated and moved, its neighbors on the lattice move accordingly. By doing this, we ensure that the points in the dataset corresponding to points that are close on the lattice, will be close to each other as well. Self organizing maps can thus be intuitively explained as starting from a two dimensional rectangular grid of prototypes $\boldsymbol{z}_j \in \mathbb{R}^N$. The prototypes can be initialized for example as lying on the two dimensional principal component plane of the data (the prototypes are then often chosen to be equispaced on that plane). The SOM procedure then slowly bends the plane/lattice to match it as much as possible with the original data.

In the case of SOM, the lattice thus plays the role of the embedding space/manifold. SOM runs through the dataset multiple times (performing multiple *epochs*). Each epoch consists of the following two steps being applied for every points $\boldsymbol{x}_i$ in the considered subset $\mathcal{S}$ [12].

- For any given point $\boldsymbol{x}_i$ from the original dataset, the first step determines the index of the closest point on the lattice,

$$\gamma^* = \underset{\boldsymbol{\gamma}}{\mathrm{argmin}}\, d(\boldsymbol{x}_i, \boldsymbol{z}(\boldsymbol{\gamma})) \tag{339}$$

  Here d is the Euclidean distance.

- The second step then updates the prototypes $\boldsymbol{z}(\boldsymbol{\gamma})$ as

$$\boldsymbol{z}(\boldsymbol{\gamma}) \leftarrow \boldsymbol{z}(\boldsymbol{\gamma}) + \alpha \mathcal{N}(\boldsymbol{\gamma}, \boldsymbol{\zeta})(\boldsymbol{x}_i - \boldsymbol{z}(\boldsymbol{\gamma})) \tag{340}$$

The *learning rate* α obeys $0 \leq \alpha \leq 1$ and is often chosen to slowly decrease with the epochs (a typical choice is to decrease α from 1 to 0 over about a thousand iterations). $\mathcal{N}(\boldsymbol{\gamma}, \boldsymbol{\zeta})$ is called the *neighborhood function*. This function is defined by a (threshold) parameter $\mathcal{N} = \mathcal{N}_\theta$ and was defined in the original von der Malsburg paper [24] to be the *Bubble function*,

$$\mathcal{N}(\boldsymbol{\gamma}_1, \boldsymbol{\gamma}_2; \theta) = \begin{cases} 0 & \text{when } d_{\mathcal{G}}(\boldsymbol{\gamma}_1, \boldsymbol{\gamma}_2) > \theta \\ 1 & \text{when } d_{\mathcal{G}}(\boldsymbol{\gamma}_1, \boldsymbol{\gamma}_2) \leq \theta \end{cases} \tag{341}$$

It is also sometimes defined from the gaussian kernel, as

$$\mathcal{N}_\theta(\boldsymbol{\gamma}_1, \boldsymbol{\gamma}_2) = \exp\left(-\frac{d^2_{\mathcal{G}(\boldsymbol{\gamma}_1, \boldsymbol{\gamma}_2)}}{2\theta^2}\right) \tag{342}$$

The distance $d_{\mathcal{G}}$ is often chosen to be any distance defined on the lower dimensional space, i.e., $d_{\mathcal{G}}(\boldsymbol{\gamma}_1, \boldsymbol{\gamma}_2) = d(\gamma_1, \boldsymbol{\gamma}_2)$. The threshold parameter θ is also often chosen to decrease with the epochs from a starting value θ_0 to 1 (over about a thousand iterations).

At the end of the iterations, the mapping $f : \boldsymbol{x} \mapsto \boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{z}(\gamma_{1,\boldsymbol{x}}, \ldots, \gamma_{P,\boldsymbol{x}}) = \boldsymbol{z}(\boldsymbol{\gamma}_{\boldsymbol{x}})$ from any point of the original dataset to the points on the lattice is obtained by simply taking the nearest prototype on the lattice, i.e.

$$f(\boldsymbol{x}_i) = \boldsymbol{z}(\boldsymbol{\gamma}) \quad \text{with} \quad \boldsymbol{\gamma} = \underset{\boldsymbol{\gamma}}{\operatorname{argmin}}\, d(\boldsymbol{x}_i, \boldsymbol{z}(\boldsymbol{\gamma})). \tag{343}$$

The general idea is summarized in Algorithm 3 (the version appears in [12]).

1 Set $k = 0$ and define the original lattice by choosing an ininitial p-dimensional subspace (for example the PCA plane) and assigning indices to the prototypes on the lattice;
2 Set the values of α and λ for epoch k;
3 **for** *all points* $\boldsymbol{x}_i$ *in the dataset* **do**
4 | Compute the index $\boldsymbol{\gamma}$ of the closest point on the lattice as in (339); Then update the prototypes as in (340);
5 **end**
6 **if** *magnitude of the prototype update is small* **then**
7 | Stop
8 **end**
9 **else**
10 | Set $k \leftarrow k + 1$ and repeat steps 2 and 3
11 **end**

Algorithm 3: Self Organizing Maps

To be complete, although we only discuss the batch version in these notes, there exist both batch and online versions of the self organizing map algorithm.

16 Reinforcement learning

The main idea underlying Reinforcement Learning (RL) is that we want the agent to learn, without a teacher, but by interacting with the environment. Reinforcement learning is perhaps the approach which is the closest to how learning occurs in human beings.

Reinforcement learning relies on 5 key concepts [22]:

- The learning agent

- The environment
- The policy
- The reward
- The value

The agent interacts with its environment to achieve a goal. Therefore the agent must be able to sense the state of the environment and to take actions that affect that state. Through its repeated interactions with the environment, the agent progressively improves its performance.

The behavior of the agent is defined by the policy. The policy can be thought of as a mapping $\pi : E \to A$ between the description of the environment and the set of possible actions.

Reinforcement learning also associates to every action, an immediate *reward* which is a direct indication on whether an action taken by the agent was good or bad.

Although the goal of the RL agent is the maximization of the combined rewards, maximizing the immediate reward might not always be the optimal choice as it will constraint the agent to keep repeating the same action again and again, without any opportunity to "explore" new possibilities. As an illustration of this, imagine that you have just moved into a new neighborhood and you would like to find the best coffee place in that particular neighborhood. One approach would be to investigate a couple of places (lets say two) and then settle down in the place that offers the best coffee. You then continue going to that same place again and again. Drinking this coffee makes you feel good and we thus assign it a positive +1 reward. But because you never tried any other places besides the first two, you have no idea whether there exists or not a place that offers a better coffee. In particular, by further investigating, you might discover a third place, where the coffee is tasting even better than place two (i.e. corresponds to a higher, lets say +2 reward). This idea is at the core of reinforcement learning. To formalize it, in addition to the immediate reward, we consider the notion of value. The value is in some sense, a prediction of the total reward that an agent could get in the future by choosing a particular action. To go back to our coffee example, an action that has a low immediate rewards (such as taking the risk of leaving the first coffee place) might have a higher value because it is possibly followed by other actions that will have a higher reward (finding a third (much) better coffee place).

Together, the five elements above also highlight the difference between reinforcement learning on the one hand, and supervised learning or unsupervised learning on the other. In supervised learning, the knowledge is provided by an external supervisor sometimes called the "teacher" who exactly knows what the correct data should be. On that aspect, reinforcement learning is closer to practical situations, in which it is often impractical to obtain examples of desired behavior. The RL agent takes the action which he assumes to be the most beneficial, yet it has no idea what the optimal action is. In fact, the agent receives a feedback from the environment but this feedback does not give him an exact "yes or no" indication, such as the labels in supervised learning.

Reinforcement learning is also different from unsupervised learning, in the sense that it does not try to reveal structure from unlabeled data. Finding structure in the data does not provide well defined rewards such as those used to guide the agent in RL.

16.1 Multi-armed bandit

The k-bandit problem is one of the simplest reinforcement learning problem. In the k-Bandit problem, the agent has to make a choice between k possible actions, each associated to particular reward which is described by a stationary distribution depending on each particular action. The objective of the agent is then to maximize the total reward over a given period of time. The simplest possible decision model for the value is then to take the expected reward for each action, i.e.

$$v(a) = \mathbb{E}\{R|a\} \tag{344}$$

If we could compute this expectation, we would select at each step, the action which maximizes this value.

In practice, we can only estimate the average by keeping track of the reward we get each time we select a particular action. I.e. at each step, the agent (provided that it chooses action a) updates the empirical value for action a as

$$v_{\text{est}}(a) = Q_t(a) = \frac{\text{sum of rewards when } a \text{ is taken prior to } t}{\text{number of times } a \text{ was selected prior to } t} = \frac{\sum_{i=1}^{t-1} R_i \mathbb{1}_{A_i=a}}{\sum_{i=1}^{t-1} \mathbb{1}_{A_i=a}} \tag{345}$$

Here $\mathbb{1}_{A_i=a}$ is defined as

$$\mathbb{1}_{A_i=a} = \begin{cases} 1 & \text{if } A_i = a \\ 0 & \text{otherwise} \end{cases} \tag{346}$$

Given Q_t, the simplest action selection procedure is to take at each step the action which has the highest $Q_t(a)$. I.e.,

$$A_t = \underset{a}{\text{argmax}} Q_t(a) \tag{347}$$

Such a procedure is known as greedy action selection method. Following a greedy policy is guaranteed to accumulate rewards. However, as we saw in the coffee example, it might miss better opportunities by sticking to a limited subset of all possible actions. An alternative consists in following a greedy approach most of the time, yet, once in while (with probability ε), trying out a new action selected at random. This approach is known as ε-greedy action selection

We can now introduce the Bandit algorithm

Initialize, for action $a = 1$ to k;
$Q(a) \leftarrow 0$;
$n(a) \leftarrow 0$;
Repeat;
$$A \leftarrow \begin{cases} \underset{a}{\text{argmax}} Q_t(a) & \text{with probability } 1-\varepsilon \\ \text{a random action} & \text{with probability } \varepsilon \end{cases};$$
$n(a) \leftarrow n(a) + 1$;
$Q(a) \leftarrow Q(a) + \frac{1}{n(a)}[R(a) - Q(a)]$;

Algorithm 4: Bandit algorithm, see [22]

The last line in this algorithm is simply the update of the average. Assuming that the average reward for action a is $Q_t(a)$, and that the agent selects action a, we update this average as

$$\begin{aligned} Q_t(a) &= \frac{1}{n(a)} \sum_{i=1}^{n(a)} R_i(a) && (348) \\ &= \frac{1}{n(a)} \left(R_n(a) + \sum_{i=1}^{n(a)-1} R_i(a) \right) && (349) \\ &= \frac{1}{n(a)} \left(R_n(a) + (n(a)-1) \frac{1}{n(a)-1} \sum_{i=1}^{n(a)-1} R_i(a) \right) && (350) \\ &= \frac{1}{n(a)} \left(R_n(a) + (n(a)-1) Q_{t-1}(a) \right) && (351) \\ &= \frac{1}{n(a)} \left(R_n(a) + n(a) Q_{t-1}(a) - Q_{t-1}(a) \right) && (352) \\ &= Q_{t-1}(a) + \frac{1}{n(a)} \left[R_n(a) - Q_{t-1}(a) \right] && (353) \end{aligned}$$

16.2 Q-learning

A major breakthrough in reinforcement learning was the development of Q-learning. Q-learning is a particular instance of "*off policy Temporal difference learning*". In Q-learning, the agent learns the action-value function Q, which is a mapping between (state, action) pairs and corresponding estimated values. I.e during training, the agent learns to associated particular states of the environment and particular actions with a corresponding value estimate. At each step, one action is selected in an ε-greedy fashion (i.e either the agent chooses the action by relying on the Q-table, or its takes the action to be random)

The action value table is then updated as follows,

$$Q[S_t, A_t] \leftarrow Q[S_t, A_t] + \alpha \left[R_{t+1} + \gamma \max_a Q[S_{t+1}, a] - Q[S_t, A_t] \right] \tag{354}$$

The table is thus updated as before, except that we now consider an additional term which takes into account the possible future increase in the value. I.e. for a given possible action choice a, we look up all the possible subsequent actions in the table. If the action a change the environment to state S'_t that means looking at all possible actions the agent can take on the environment in that particular state, so all entries of the form $Q[S'_t, a]$. Then we take our potential future gain to be the maximal value. In Q-learning, we thus try to be one step ahead compared to the traditional bandit algorithm which only considers the current value estimate.

In general, we will not learn the action-value table Q, but we will replace it by a neural network which will approximate it.

References

[1] Andrew D Back and Andreas S Weigend. A first application of independent component analysis to extracting structure from stock returns. *International journal of neural systems*, 8(04):473–484, 1997.

[2] Christopher M Bishop et al. Pattern recognition and machine learning (information science and statistics). 2006.

[3] Leon Bottou and Yoshua Bengio. Convergence properties of the k-means algorithms. In *Advances in neural information processing systems*, pages 585–592, 1995.

[4] George Casella and Roger L Berger. *Statistical inference*, volume 2. Duxbury Pacific Grove, CA, 2002.

[5] Edward W Forgy. Cluster analysis of multivariate data: efficiency versus interpretability of classifications. *biometrics*, 21:768–769, 1965.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[7] Aapo Hyvärinen and Erkki Oja. Independent component analysis: algorithms and applications. *Neural networks*, 13(4-5):411–430, 2000.

[8] Željko Ivezić, Andrew J Connolly, Jacob T VanderPlas, and Alexander Gray. *Statistics, data mining, and machine learning in astronomy: a practical Python guide for the analysis of survey data*, volume 1. Princeton University Press, 2014.

[9] Leonard Kaufman and Peter J Rousseeuw. *Finding groups in data: an introduction to cluster analysis*, volume 344. John Wiley & Sons, 2009.

[10] Joseph B Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, 1964.

[11] Joseph B Kruskal. Nonmetric multidimensional scaling: a numerical method. *Psychometrika*, 29(2):115–129, 1964.

[12] John A Lee and Michel Verleysen. *Nonlinear dimensionality reduction*. Springer Science & Business Media, 2007.

[13] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[14] J MacQueen. On convergence of k-means and partitions with minimum average variance. In *Annals of Mathematical Statistics*, volume 36, page 1084. inst mathematical statistics IMS business office-suite 7, 3401 Investment blvd, Hayward, CA 94545, 1965.

[15] James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[16] Kevin P. Murphy. Machine learning, a probabilistic perspective, 2012.

[17] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[18] José M Pena, Jose Antonio Lozano, and Pedro Larranaga. An empirical comparison of four initialization methods for the k-means algorithm. *Pattern recognition letters*, 20(10):1027–1040, 1999.

[19] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[20] Michael D Spivak. *A comprehensive introduction to differential geometry*. Publish or perish, 1970.

[21] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[22] Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

[23] Joshua B Tenenbaum, Vin De Silva, and John C Langford. A global geometric framework for nonlinear dimensionality reduction. *science*, 290(5500):2319–2323, 2000.

[24] Chr Von der Malsburg. Self-organization of orientation sensitive cells in the striate cortex. *Kybernetik*, 14(2):85–100, 1973.

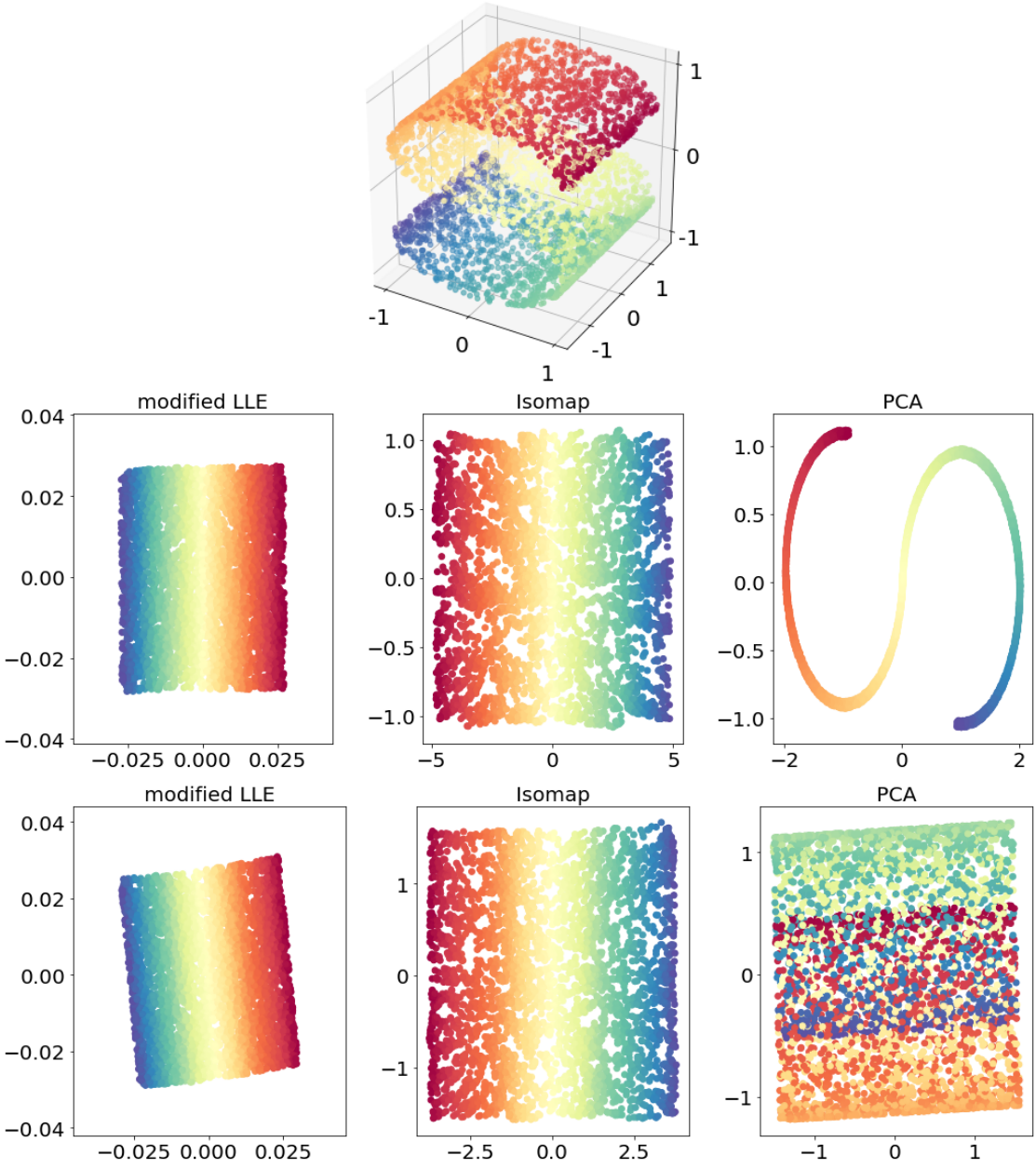

Figure 5: Various two dimensional representations of the S-shaped manifold from scikit-learn. The first row corresponds to two dimensional representations of the original manifold. In the second row, this maniold was modified following [8] to further highlight the weakness of PCA.

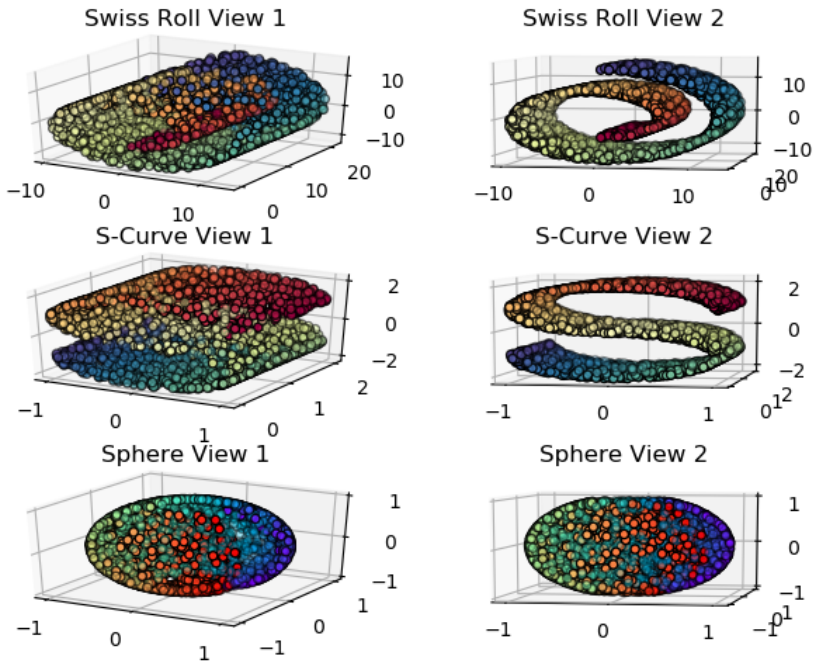

Figure 6: Usual benchmarks used for the comparison of non linear latent variable models. From the top to bottom: Swiss roll, S-manifold and sphere.

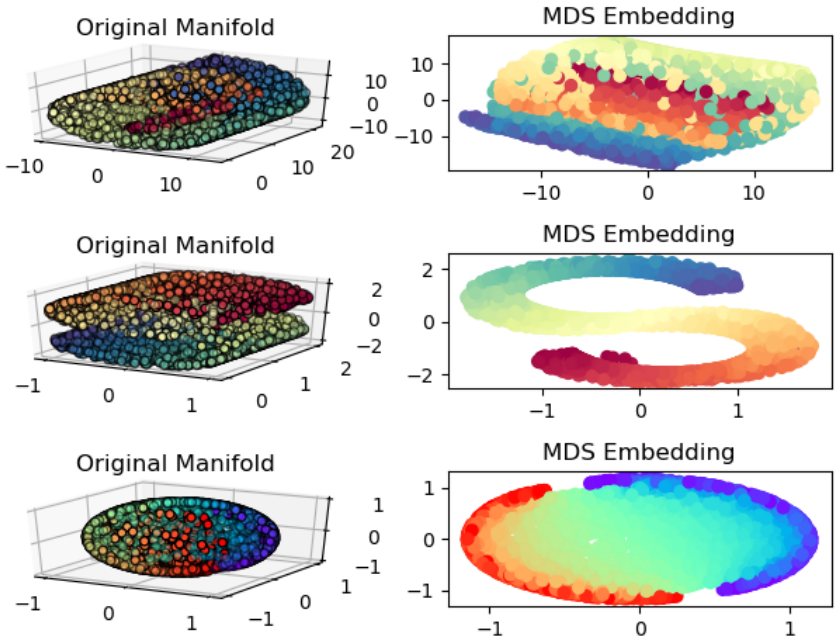

Figure 7: Two dimensional representation obtained by Multidimensional scaling on the *Swiss Roll*, the *S-curve* and the *Sphere* manifolds.

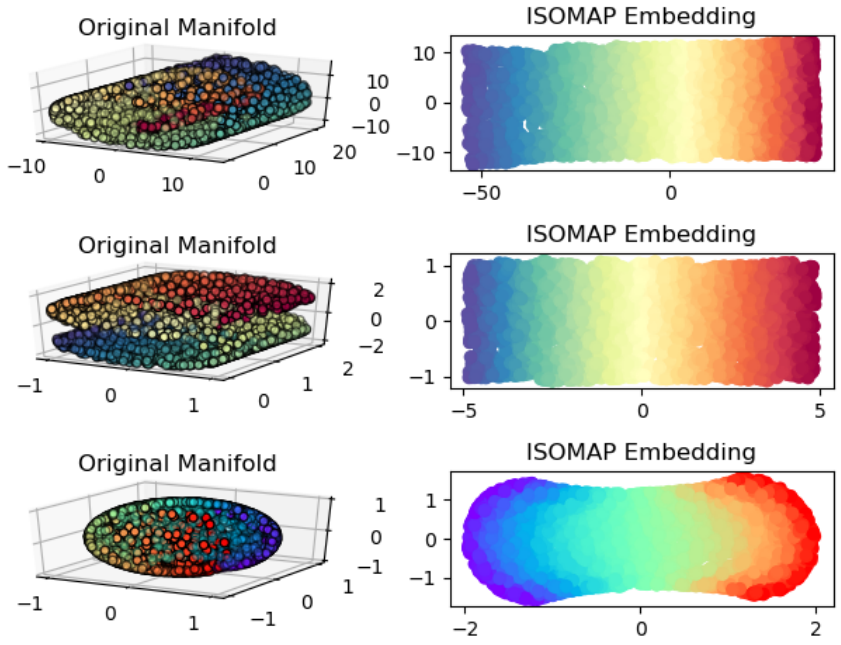

Figure 8: Two dimensional representation obtained by ISOMAP on the *Swiss Roll*, the *S-curve* and the *Sphere* manifolds.

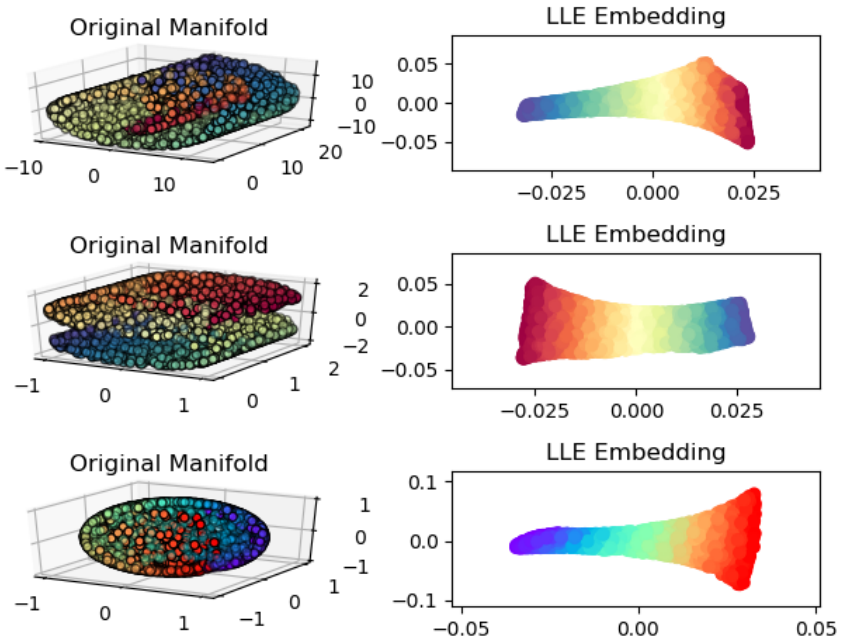

Figure 9: Two dimensional representation obtained by locally linear embedding on the *Swiss Roll*, the *S-curve* and the *Sphere* manifolds.