Unsupervised learning - lecture notes

M. Dalmonte and V. Vitale and...

January 2021

These notes are related to a course given in Jan/Feb, 2021 by Laio and Rodriguez on unsupervised learning.

Contents

1 First lecture: introduction

1.1 Why unsupervised learning? Motivations from a computational physics perspective

Rare events, a known challenge in computational physics, from phase transition, to the understanding of molecular processes, are very hard to access. Thanks to impressive computational advances, the atomistic simulation community is now able to simulate complex systems and generate complicated trajectories corresponding to very involved processes. For example it is nowadays possible to simulate molecules dynamics collecting over thousands of configurations. The challenge is now transferred to analyzing such huge amount of data. This requires the development of new methods to perform data analysis.

1.2 Warming-up: an example of supervised learning

We start by going over one example of supervised learning as a warm up.
Let us consider a set of many images of cats and dogs. Each image is black and white, and contains $N \times N$ pixels (1Mpixel) that are characterized by their intensity, expressed as a real number. Thus a single image corresponds to N^2 real numbers.

Supervised learning means training a model that, given an image $x \in \mathbb{R}^{N^2}$ as input, will return as an output a single vector $y \in \mathbb{R}$ that represents the degree of doggish-ness of the image x. To classify the image as a "dog", one then defines a threshold which allows to associate a bit from to the output. Formally, this reads as:

$$x \in \mathbb{R}^D \to y \in \mathbb{R} \to (0, 1) \tag{1}$$

Performing such procedure is equivalent to performing a dimensional reduction from 10^{N^2} to 1. The power of this procedure is remarkable if one observes that a standard pictures contain at least 10^6 pixels (1000×1000).

1.3 From supervised to unsupervised learning

Performing this dimensional reduction, did we throw away something? The obvious answer is yes.
We can enlarge the question, and ask to distinguish not only dog/cat, but also pictures the angle of the picture (front/side) or any kind of feature one could think about.
In this regard, given a set of images, one could be tempted to say that there are infinite independent classification tasks that one can carry on.
In practice, even if one could carry on an infinite number of classification task, this last statement is wrong. The rationale behind it is that many of them will not be independent.

1.3.1 Dimensional reduction and local intrinsic dimension

Q1. - Dimensional reduction can be summerized with the following question: starting from the vector $x \in \mathbb{R}^D$, are these data contained in a manifold $\mathcal{M}$ of lower dimension $d < D$?

Let us consider a data set as in Fig. 1. It is evident that there are regions where points are at least locally correlated: that is given a point of the dataset, not always it happens that its neighbours are distributed in arbitrary directions. The number of linearly independent directions around a point in which one can find other points is called *local intrinsic dimension* d of the manifold $\mathcal{M}$. This quantity can be region-dependent as in Fig. 1. We have a $d = 2$ region highlighted in blue and a $d = 1$ one in red. While there is no guarantee that such lower dimensions are meaningful, there is evidence that many data sets are contained in manifolds of intrinsic dimension $d \ll D$.
The tasks of unsupervised learning can be summerized as follows:

T1. to find explicit an parametrization of the manifold $\mathcal{M}$, that is a set of coordinates y that parametrizes $\mathcal{M}$. This task is often very demanding.

T2. finding the intrinsic dimension of

$$\mathcal{M}$$

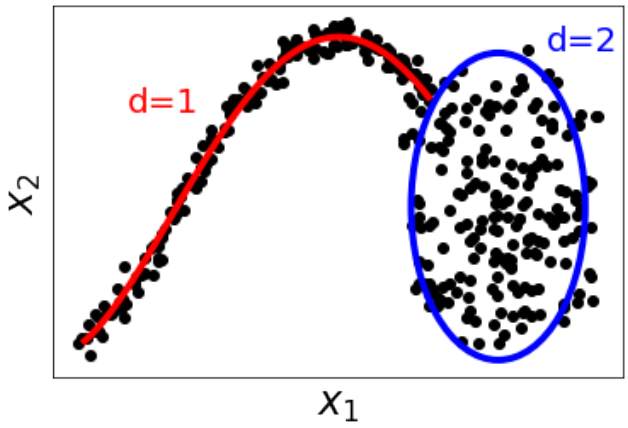

Figure 1: Two-dimensional ($D = 2$) data set: $x^i = (x_1^i, x_2^i)$. The intrinsic local dimension d depends on the region in the plane. In red it is highlighted a region where $d = 1$, in blue where $d = 2$.

. In particular answering to the following broad question:

- is the intrinsic dimension constant over $\mathcal{M}$?
- what is the topology of the manifold?

Let us consider an example: for images of cats and dogs, typical datasets have dimensions of order 10. There are 10 independent classification tasks that one can carry on (not rigorous statement due to curvature [only rigorous for hyperplanes], but gives a good order of magnitude). This implies that, when doing supervised learning, moving from D down to 1, is not a drastic improvement: we are ultimately throwing away only 9 dimensions, not $10^6 - 1$ as we could have initially guessed.

1.3.2 Homogeneous and inhomogeneous datasets

Q2. - An important question regarding data sets is the following: are the data points uniformly distributed on $\mathcal{M}$? Formally, given the data $x \in \mathcal{M}$, and a probability density $\rho(x)$ such that $\rho(x) = 0$ if $x \notin \mathcal{M}$, is $\rho(x)$ constant on $\mathcal{M}$? Probability densities can be inhomogeneous for several reasons: presence of clusters or secondary probability peaks, very often corresponding to additional information hidden in the dataset . In general, $\rho(x)$ is not featureless. Then, two additional tasks can be added to the list for unsupervised learning has then an additional task

T3. Estimating $\rho(x)$.

T4. Finding the position of the different peaks of $\rho(x)$ which in many relevant cases correspond to different categories.

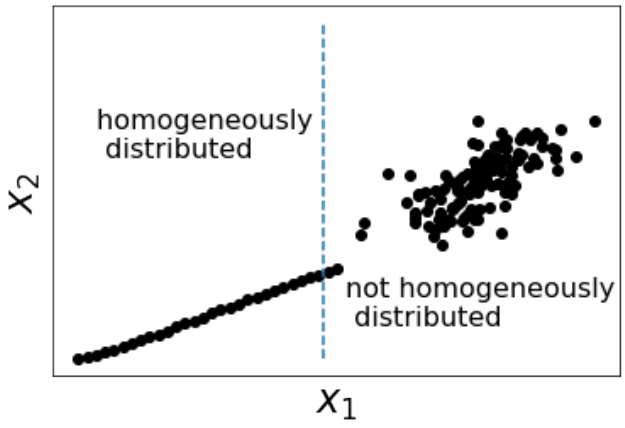

Figure 2: Two-dimensional data set $x^i = (x_1^i, x_2^i)$. In the left quadrant the data are homogeneously distributed ($\rho(x) = \text{const.}$ for $x \in \mathcal{M}$), in the right one they are not.

In general, in unsupervised learning, all of these features have to emerge from the analysis, and not being given as an input.

The re
st of the lecture notes will be about addressing these four tasks.

To fix the language, it is sometimes useful to have composite maps, where for instance our dataset is separated into regions, each of them having a given chart. There may be intersections between these submanifolds: at intersections, the intrinsic dimension will increase (typically, will combine the two dimensions of the manifolds). This gives a simple proxy to identify boundaries of a manifold and interesections - and thus its topology.

1.3.3 Examples of dimensional reduction in physics and overall approach

In natural sciences, dimensional reduction has always been performed according to domain knowledge, for example studying a magnet one identifies the magnetization as order parameter. Theories are often developed in this way: they work at given scales, and provide predictions. The assumptions at the basis of dimensional reduction can then be checked a posteriori. Examples include:

- Thermodynamics: the physics of a manyparticle systems is well described by few macroscopic variables.
- Systems undergoing chemical reactions: chemical reactions on their own are described by very few variables (and labels), at least at the phenomenological level, even if the systems at the microscopic level are very high dimensional.

The difference of this theories with respect to data science approach is that the latter tries to derive these parameters, or at least some of their properties, without any initial assumption.

1.3.4 Plan for next lectures

Starting from next lecture, we will discuss methods to determine charts.

1. We will start from $\mathcal{M}$ being hyperplanes [no curvature, no topology]. This will be done via principal component analysis (PCA);
2. The next question will deal with data points including curvatures, but on manifold that is topologically equivalent to a hyperplane. Methods will include ISOMAP, kernel PCA;
3. the third level will involve $\mathcal{M}$that are not topologically equivalent to hyperplanes - that is, complex topological objects. This is the frontier of current data analysis - the main difficulty being that one is never able to find a "single" y (due to points at intersections between surfaces, for instance).

Miscellanea. - Is there a classification of dimensions (more important to less important)? This is actually given by their locality (related to the eigenvalues of their covariance matrix, for manifolds that are topologicall simple).

2 Second lecture: Principal Component Analysis (PCA)

This lecture is dedicated to Principal Component Analysis (PCA) which is a cornerstone of manifold learning methods. In fact, in most of the cases the other methods just recast the problem so to be able to perform PCA afterwards. PCA was first introduced in 1901 by Paerson in a different fashion. In general, manifold learning approaches pass from a given representation x (D-dimensional) to another one y (d-dimensional), looking for $d \ll D$ variables that describe as faithfully as possible the data set: this is called *dimensional reduction*. Thus they differ in the manner they: 1) Implemenet the variable transformation, 2) define the *faithfulness*. We will first introduce the notation for this section, then discuss PCA in details.

2.1 Notations and assumptions

We will denote with $x^i \in \mathcal{R}^D$ a D-component vector belonging to the data set, i.e. a vector of pixels values corresponding to a given photo of a cat. Here x^i_k is a given component of the vector x^i, $k = 1, \dots, D$ and $i = 1, \dots, N$; N is the

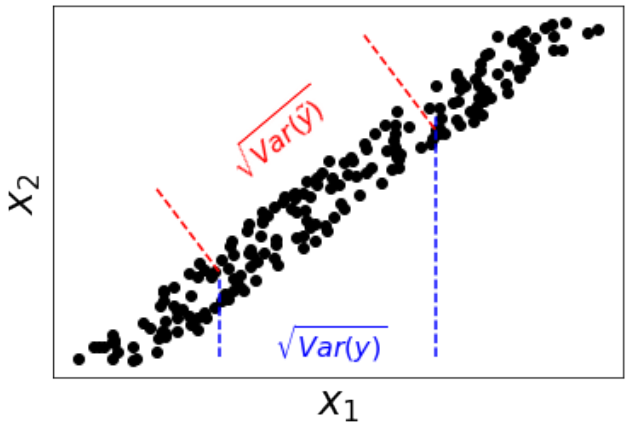

Figure 3: Example of a $D = 2$ dimensional dataset. (Temporary Figure)

number of data points or samples. The vectors x^i are considered *centered*:

$$x^i \text{ centered} \rightarrow \sum_i x^i_k = 0 \quad \forall\, k. \tag{2}$$

Such an assumptions is trivial since for not centered samples it is possible to subtract the average over the sample to each component k of x^i.
We will denote with $y^i \in \mathcal{R}^d$ a d-component vector belonging to the lower dimensional manifold. Here, given y^i_k, one has $k = 1, \ldots, d \ll D$ and $i = 1, \ldots, N$.

2.2 Introduction to PCA

As the task of manifold learning approaches is *dimensional reduction*, they differ in the ansätz made to realize it. PCA approach exploits the following:

1. y^i must be linear functions of x^i, namely

$$y^i_k = \sum_{l=1}^{D} A_{kl} x^i_l. \tag{3}$$

2. The parameters A_{kl} are obtained maximizing the variance in the dataset $\{y^i\}_{i=1,\ldots,d}$.

Let us discuss an example where $D = 2$ and one wants to perform a PCA to reduce the dimension of the dataset to $d = 1$. An example of the samples is depicted in Fig.3. According to PCA one has to consider y^i a linear combination of x^i. The choice $y^i = x^i_1$ would be possible. However condition 2. suggests that the variance of $\{y^i\}_{i=1,\ldots,d}$ must be maximised and it is evident that $\tilde{y}^i =$

$ax_1 + bx_2$ as depicted in red in Fig.3 satisfies this constrain.
Accordin to PCA $\tilde{y}^i$ is better than y^i: how can this be generalised?
The variable y^i is obtained diagonalizing the covariance matrix of the data x^i.
The covariance matrix is defined as

$$\text{Cov}(\{x^i\}) = \frac{1}{N}\sum_i x_l^i x_m^i \equiv C_{mn} \tag{4}$$

Note that the average squared of the samples is not subtracted because we are assuming that the dataset is centered. The covariance matrix of the points in Fig.3 can be considered something like:

$$C = \begin{pmatrix} 1 & 0.9 \\ 0.9 & 1.09 \end{pmatrix} \tag{5}$$

and easly diagonalized as

$$\begin{cases} \lambda_1 = 1.89, & A_1 = (0.7, 0.76), \\ \lambda_2 = 0.12, & A_2 = (-0.71, 0.7). \end{cases} \tag{6}$$

Here λ_1, λ_2 are the eigenvalues of C_{mn} and A_1, A_2 are the eigenvectors. Evidently $\lambda_2 \ll \lambda_1$. Since the purpose of PCA is selecting the highest variance we may neglect λ_2 and define

$$y^i = A_{11}x_1^i + A_{22}x_2^i = 0.7x_1^i + 0.76x_2^i. \tag{7}$$

We are selecting a line which goes through the data points and all the y^i are essentially the orthogonal projection of x^i on this line. What happens if no eigenvalues is negligible? The advanced methods which improve PCA are devoted to tackle this problem and we will discuss about them extensively in the following lectures.
The quality of the PCA approximation is measured via the following ratio

$$\text{PCA } \textit{fidelity} = \frac{\text{Tr}(\text{Cov}(y^i))}{\text{Tr}(\text{Cov}(x^i))}. \tag{8}$$

In this case it would be $\frac{\lambda_1}{\lambda_1+\lambda_2} = 0.96$. The threshold which defines a good approximation is completely arbitrary, in the following it will be 0.95.

Method summary - We can reduce the protocol to three steps:

1. Compute the covariance matrix $C_{mn} = \frac{1}{N}\sum_i x_l^i x_m^i$
2. Diagonalize C_{mn}.
3. Keep d eigenvectors so that the fidelity satisfies a given threshold.

2.3 Formal discussion of PCA analysis

In this section we justify the choice of diagonalizing the covariance matrix as a way to reduce the dimensionality of the manifold. Let us recap the assumptions of PCA:

1. $y_k^i = \sum_{l=1}^{D} A_{kl} x_l^i$.
2. $\sum_l A_{kl} A_{hl} = \delta_{kh}$
3. A_{kl} : $\mathrm{Tr}(\mathrm{Cov}(y^i))$ is maximum.

We introduced a second condition on the paramters of the linear combination, which is that they owe to be orthonormal, since an orthonormal basis is always best suited for linear algebra. Let us compute explicitly the point 3. and impose the constraint via Lagrange multipliers.

$$\mathrm{Tr}(\mathrm{Cov}(y^i)) = \frac{1}{N}\sum_l y_l^i y_l^i = \frac{1}{N}\sum_{lmn} A_{lm} A_{ln} x_m^i x_n^i = \sum_{lmn} A_{lm} A_{ln} C_{mn} \tag{9}$$

We only impose the constraint of normalization as orthogonality will come out for free. Let us defined

$$\mathcal{L} \equiv \mathrm{Tr}(\mathrm{Cov}(y^i)) - \sum_m \lambda_m (\sum_l A_{ml}^2 - 1). \tag{10}$$

$\mathcal{L}$ is a quadratic form where λ_m are the Lagrange multipliers that enforce the D constraints $\sum_l A_{ml} A_{ml} = 1$. The maximization consists in imposing the partial derivative of $\mathcal{L}$ with respect to all the A_{hk} equal to zero. Therefore

$$\frac{\partial \mathcal{L}}{\partial A_{hk}} = 0 \rightarrow \sum_l A_{hk} C_{kl} - \lambda_h A_{hk} = 0. \tag{11}$$

This is an eigenvalue equation for C_{lk}. The optimal parameters A_{hl} of the PCA ansätz are the coefficients of the eigenvectors of the covariance matrix. Since the covariance matrix is real and symmetric, its eigenvectors form an orthonormal basis, thus condition 2. is satisfied. Diagonalised C_{lk} one can keep d of the D eigenvectors and evaluate the fidelity:

$$\frac{\mathrm{Tr}(\mathrm{Cov}(y^i))}{\mathrm{Tr}(\mathrm{Cov}(x^i))} = \frac{\sum_{l=1}^{d} \lambda_l}{\sum_{l=1}^{D}} \tag{12}$$

Note that all the eigenvalues are non negative. Let us observe that when the data $x^i) i = 1, \ldots, N$ lay on a d-dimensional hyperplane, the dimensional reduction provided by PCA is *rigorous*. In fact the covariance matrix is a $D \times D$ matrix of rank d which means that $D - d$ eigenvalues are exactly zero. A simple example are data in $D = 2$ that lay on a straight line (which is hyperplane of dimension 1). In this case we can also say that y^i form a chart for x^i. This will pratically never happen. However a signature of a system in which a PCA may

be meaningfull is a gapless spectrum for the correlation matrix. In that case, plotting the eigenvalues in decreasing order allows to identify the gap and there the truncation can be perfomed. Of course the largest eigenvalues will be the ones carrying most of the information on the dataset.

The absence of a gap in the spectrum of the covariance matrix provides evidence that the manifold $\mathcal{M}$ does not lay on a hyperplane, hence there is no formal justification for the truncation. Indeed, operatively it may be still reasonable.

In the following lecture we will discuss what happens if the data lay on a curved or twisted manifold.

3 Third lecture

PCA is the appropriate approach to obtain a specific chart of a manifold $\mathcal{M}$if the latter is, at least approximately, an hyperplane. Typically, there is no reason that a generic data set is an hyperplane, since typically constraints are non-local functional of the variables.

For the next three lectures, we focus on cases where $\mathcal{M}$is topologically equivalent to a hyperplane, with a intrinsic dimension d that is small. If you apply PCA to these type of datasets, there is no way of finding approximate one-dimensional representations.

One possible way to handle such data sets is to focus on distances rather then on feature vectors. While in PCA you work with $\vec{X}_i \in \mathbb{R}^D$, one can work with distances, that can be thought as similarity measures between points in the data set:

$$\Delta^{ij} = ||\vec{X}_i - \vec{X}_j|| \tag{13}$$

with the cartesian distance being the simplest example. There are of course other possible choices, such as:

$$\Delta^{ij} = K(||\vec{X}_i - \vec{X}_j||) \tag{14}$$

where K is a non-linear function of its argument, or geodesic distances.

Methods based on distances are mostly inspired by an idea of Torgensen in 1951, that goes under the name of *multi-dimensional scaling*. This is at the basis of most of the methods of manifold learning that are currently employed. Let us formulate it rigorously.

Given a distance matrix Δ^{ij}, find a set of coordinates Y^i_ℓ such that, if one computes the cartesian distance between these coordinates

$$\delta^{ij} = ||Y^i_\ell - Y^j_\ell|| \tag{15}$$

would find that

$$\delta^{ij} \simeq \Delta^{ij} \quad \forall (i,j). \tag{16}$$

Hence the cartesian distance in this new space is a good approximation of the original distance.
It is important to determine what the similarity in Eq. 16 means. The similarity, in this case, is quantified by minimizing the *stress*. The latter object can be defined in different ways and we write here several alternatives:

- **Classical multi-dimensional scaling:** in this case, the stress is defined as:
$$S = \sum_{i,j}((\Delta^{ij})^2 - (\delta^{ij})^2)^2 \tag{17}$$
so this implies that finding this has an explicit solution under some specific conditions. The existence of such explicit solution is due to the fact that I am considering differences of power 2 (for linear difference, it won't be the case);

- **Metric multi-dimensional scaling:**
$$S = \frac{\sum_{i,j}((\Delta^{ij}) - (\delta^{ij}))^2}{\sum_{i,j}\delta^{ij}} \tag{18}$$
in this case, due to the square root entering in the cartesian distance, this stress is not a linear function of my coordinates Y. So one would have to perform optimization (gradient descent), but cannot be obtained by just diagonalizing a matrix, so there is not explicit solution;

- **Non-metric multi-dimensional scaling** this time the stress is:
$$S = \frac{\sum_{i,j}(g(\Delta^{ij}) - f(\delta^{ij}))^2}{\sum_{i,j}f(\delta^{ij})^2} \tag{19}$$
where g, f are given functions. One of the most important of these methods - introduced by Parrinello and Ceriotti - is Sketchmap (see the link here). The reason behind this method is that one wants to weight less distances between points that are very far from each other (since matching them is probably not super relevant), while instead optimizing as well as possible distances between neighbors. The main difficulty is that the final problem is cast as a highly non-linear optimization problem, including many metaparameters (free parameters). The optimization will likely result in glassy dynamics. Typically, the functions f, g are kernel functions that is, they amplify differences at short distances, one example being $f(r) = e^{-\frac{r^2}{2\sigma^2}}$.

3.1 Classical multi-dimensional scaling

Here we focus on classical multi-dimensional scaling (CMDS), that is, when the stress is defined as in Eq. (17). Let us set two conditions, always under the assumption that the data set X is centered:

1. $\Delta^{ij} = ||X^i - X^j||$,
2. $Y^i_\ell = \sum_m B_{\ell,m} X^i_m$ and the transformation coefficents form an orthonormal basis:
$$\sum_k B_{\ell k} B_{mk} = \delta_{\ell,m}. \tag{20}$$

Under these two hypothesis, one can show that CMDS has an explicit solution, that is equivalent to PCA. Essentially, one finds exactly the same chart since:

$$B_{\ell m} = \sqrt{\lambda_\ell} A_{\ell m} \tag{21}$$

where A is the covariance matrix. This result while negative in principle, because using distances does not improve the dimensional scaling, can also be exploited proactively. In particular, in cases where the first hypothesis is violated, one does not obtain PCA.

The aim in the following is to understand the relation between CMDS and PCA.

Let us first list a few results which will be useful in the following:

1. if (1) and (2) above holds, the stress results in
$$S = \sum_{ij} (G^{ij} - (\sum_\ell Y^i_\ell Y^j_\ell)) \tag{22}$$
where $G = \sum_{\ell=1}^D X^i_\ell X^j_\ell$ is the grand-matrix of the data, and the second term is the Grahm matrix in the reduced space;
2. for a given feature vector X^i_ℓ (that is actually $N \times D$ dimensional), I can construct two square matrices, the covariance matrix:
$$C_{\ell m} = \sum_{i=1}^N X^i_\ell X^i_m \tag{23}$$
with corresponding eigenvectors λ_ℓ, and and the Grahm matrix:
$$G^{ij} = \sum_{\ell=1}^D X^i_\ell X^j_\ell \tag{24}$$
then, the eigenvectors of G satisfy:
$$G^{ij} \hat{Y}^j_\ell = \mu_l \hat{Y}^i_\ell \tag{25}$$
and are thus the principal components
$$\hat{Y}^i_\ell = \sum_m A_{\ell m} X^i_m \tag{26}$$
with $\mu_\ell = \lambda_\ell \forall \ell \leq D$, and 0 otherwise. This implies that to find the principal components, I can either diagonalize the Grahm matrix or the covariance matrix.

We can now use the properties above to rewrite the stress, first of all expressing (remembering that G is symmetric):

$$G^{ij} = \sum_{\ell=1}^{D} \lambda_\ell \hat{Y}_\ell^i \hat{Y}_\ell^j \tag{27}$$

and adding it to the stress:

$$S = \sum_{ij} [G^{ij} - \sum_{\ell} Y_\ell^i Y_\ell^j]^2 = \tag{28}$$

$$= \sum_{ij} \left(\sum_{\ell} \lambda_{\ell=1}^{D} \hat{Y}_\ell^i \hat{Y}_\ell^j - \sum_{\ell=1}^{d} Y_\ell^i Y_\ell^j \right)^2 \tag{29}$$

so that:

$$Y_\ell^i = \sqrt{\lambda_\ell} = \hat{Y}_\ell^i \iff B_{\ell m} = \sqrt{\lambda_\ell} A_{\ell m} \tag{30}$$

Let us consider the data lying on a hyperplane of dimension D. Here we have $\lambda_\ell = 0 \; \forall \ell > D$. So in this case, the stress is identically zero and CMDS is exact.

Conclusions. - We have derived so far that:

- PCA can be obtained by diagonalizing the G matrix.
- the cartesian distance computed with coordinates:

$$Y_\ell^i = \sqrt{\lambda_\ell} \hat{Y}_\ell^i \tag{31}$$

  minimizes the stress as defined in CMDS.

3.2 CMDS beyond PCA

So far, CMDS looks like a very complicated approach which results in the same of doing PCA, but diagonalizing a larger matrix. What is the key trick that makes CMDS useful?
The trick is as follows: we diagonalize G^{ij}, but we do not estimate it directly from the original features X^i. In fact, we instead estimate it from Δ^{ij}, that can be different from δ^{ij}, that is we use only the distances, so we can violate hypothesis (1) above.
Let us define property (3)[1]:

$$\hat{G}^{ij} = -\frac{1}{2}(\Delta^{ij} + \frac{1}{N^2} \sum_{i,j} \Delta^{ij} - \frac{1}{N}(\sum_i \Delta^{ij} + \sum_j \Delta^{ij})) \tag{32}$$

[1]Note that in principle, higher order terms can also be included. However, the expression below is known to be exact in the case there exist a set of coordinates where Δ becomes a cartesian distance[Check this].

where the coordinates X^i do not appear anymore. The operation of putting the last two terms, that are symmetric, is sometimes referred to as double-centering. The sum of rows or columns vanishes, $\sum_i \hat{G}^{ij} = 0 = \sum_j \hat{G}^{ij}$. We thus can compute the Grahm matrix from distances only.

Let us derive property (3) analitically. We assume that Δ^{ij} is estimated as the square of a cartesian distance in an *unspecified* vector space, $\Delta^{ij} = ||z_i - z_j||^2$. We do not assume that we know the feature vectors X^i. This implies that Δ has properties of a metric. We can now use the following:

$$\Delta^{ij} = ||z_i||^2 + ||z_j||^2 - 2z_i \cdot z_j \tag{33}$$

and define our Grahm matrix as $G^{ij} = z_i \cdot z_j$, or:

$$G^{ij} = -\frac{1}{2}(\Delta^{ij} - ||z_i||^2 - ||z_j||^2) \tag{34}$$

and if I use:

$$\sum_i \Delta^{ij} = \sum_i ||z_i||^2 + N||z_j||^2 - 0 \tag{35}$$

since er assumed that the features are centered ($\sum_j z_j = 0$). Now performing an extra sum one obtains

$$\sum_{ij} \Delta^{ij} = 2N \sum_i ||z^i||^2 \tag{36}$$

and then can use a combination of the previous equations to obtain Eq. (32). The take-home message of this algebra is that, if we assume Δ satisfies the metrics axioms, then I can calculate the Grahm matrix using this simple derivation - that is called **double centering**.

3.2.1 Spectral problem of the Grahm matrix

Starting from the validity of Eq. 32, the first step is to solve the eigenproblem corresponding to the Grahm matrix:

$$\sum_j G^{ij} V_\ell^j = \lambda_\ell V_\ell^i \tag{37}$$

We write

$$V_\ell^i = \sum_m A_{\ell m} X_m^i \tag{38}$$

with covariance matrix A:

$$A = \sum_\ell C_{\ell m} A_{\ell n} = \lambda_m A_{mn} \tag{39}$$

where $A_{\ell m}$ is an eigenvector of the covariance matrix.
Let us represent the Grahm matrix in terms of the feature vectors:

$$G^{ij} = \sum_\ell X_\ell^i X_\ell^j \tag{40}$$

and write:

$$\sum_j G^{ij} V_\ell^j = \sum_{j,m} X_m^i X_m^j V_\ell^j \tag{41}$$

and looking for eigenvectors on the basis defined by the feature vectors, we use:

$$V_\ell^i = \sum_m B_{m\ell} X_\ell^i \tag{42}$$

and insert it in the previous equation, we get:

$$\sum_j G^{ij} V_\ell^j = \sum_{j,m,n} X_m^i X_m^j B_{n\ell} X_n^j \tag{43}$$

We can now use the definition of covariance matrix $C_{mn} = \sum_i X_m^i X_n^i$, and get:

$$\sum_j G^{ij} V_\ell^j = \sum_{mn} C_{mn} B_{n\ell} X_m^i. \tag{44}$$

Since $\sum_n C_{mn} B_{n\ell} = \lambda_\ell B_{m\ell}$, the previous equation can be written as

$$\sum_j G^{ij} V_\ell^j = \sum_m \lambda_\ell B_{m\ell} X_m^i = \lambda_\ell V_\ell^i \tag{45}$$

where the last is just obtained from the definition of V.
What we have proven here is that the eigenvectors of the Grahm matrix are related to the eigenvectors of the covariance matrix by a simple relation (the definition of V, Eq. (38)).

Protocol of non-trivial CMDS - We now formulate a protocol of practical use:

1. set a distance Δ^{ij} that shall be insensitive to the curvature of the embedding manifold.
2. estimate the Grahm matrix G^{ij} using the formula above.
3. diagonalize G, solving:
$$\sum_i G^{ij} \hat{Y}_\ell^j = \lambda_\ell \hat{Y}_\ell^j \tag{46}$$
4. inspect the spectrum λ_ℓ. If the spectrum shows a clear gap, choose d and find the chart on the manifold
$$Y_\ell^i = \sqrt{\lambda_\ell} \hat{Y}_\ell^i \tag{47}$$

A point to remark is that, for now, this procedure is abstract and we have not even discussed the first point, that is crucial.

4 Fourth lecture - ISOMAP

Today we introduce a non-trivial application of MDS. ISOMAP is the first significant attempt to solve the problem of curvature on the embedding manifold. It is strongly based on multi-dimensional scaling. We remind that the starting point for MDS is a set of distances Δ^{ij}, defined via a positive defined matrix satisfying the properties of a distance. Choosing the cartesian one $\Delta^{ij} = ||x^i - X^j||$ MDS becomes PCA. In general, if X^i are not available, one can estimate the Grahm matrix using Eq. 32, that only contains distances.

Introduced in 2010 by Tenenbaum, Langford and de Silva, The basic idea of ISOMAP is to perform MDS using as a distance Δ^{ij} an estimator of the geodesic distance on $\mathcal{M}$. The geodesic distance needs the length to be defined. Taken two points x_i, x_j, we call $\gamma(t), t \in [0,1]$ a path which connects x_i and x_j. One can define the length of the path as

$$L(\gamma) = \int_0^1 dt \sqrt{\sum_{\alpha,\beta} g_{\alpha\beta} \dot{\gamma}_\alpha \dot{\gamma}_\beta} \tag{48}$$

where $g_{\alpha\beta}$ is the metric tensor. Note that this is really needed only if one is not working in the embedding coordinates: in the ambient space $\mathbb{R}^D$, we can use the conventional norm:

$$L(\gamma) = \int_0^1 dt ||\dot{\gamma}(t)|| \tag{49}$$

which implies that we are summing the length of the infinitesimally small segments composing the curve.
The geodesic distance is defined as follows:

$$\Delta^{ij} = \min L(\gamma) \quad \gamma(0) = x_i, \gamma(1) = x_j \tag{50}$$

that is the shortest curve connecting the two chosen points, amongst all the possible ones. In practical applications, $\mathcal{M}$ is defined only implicitly by the property that we assume that all my data points belong to $\mathcal{M}$. i.e. one does not have a defined chart on the manifold. How can one even compute a geodesic? One needs to assume that, if x_i, x_j are close enough, then $\Delta^{ij} = ||x_i - x_j||$. This is equivalent to say that the curvature of my manifold is small enough that on the typical distance of nearby points, it can be approximated by a hyperplane.

Protocol to obtain the geodesic - To calculate the geodesic, one use the following procedure.

- Build a graph joining the data points that are 'close enough' (this requires fixing a cut-off distance). In this way we switch from the data set X^i to a sparse graph. With sparse one means that, on average, there are few connections per data point.

- Estimate the geodesic distance with the minimum distance on the graph, i.e. looking for the shortest path between tho points. This is now a problem in network theory, that is amenable to various algorithms.

4.1 Estimator of the geodesic distance: the Floyd algorithm

One of the algorithms that perform the claculation of the geodesic distance starting from a graph is the Floyd-Warshall algorithm. The latter proceeds iteratively through all the vertices of the graph finding the shorted path that connects them. It works as follows:

- initialize Δ in the ambient space:

$$\Delta^{ij} = ||x^i - x^j|| \quad \text{if } ||x^i - x^j|| < r_c \tag{51}$$

  and a large number otherwise. Here r_c is a threshold distance.

- perform the following loop for $i, j, k = 1, N$, where N is the number of vertices:
  - Calculate Δ^{ij}, Δ^{ik}, Δ^{jk}.
  - If $\Delta^{ij} > \Delta^{ik} + \Delta^{kj}$ set $\Delta^{ij} = \Delta^{ik} + \Delta^{kj}$.

  At each step, the loop is calculating the mdistance between point i and j using k as intermediate points and comparing this distance with the saved one Δ^{ij}. The final value of the loop returns the minimum length Δ^{ij} of all the paths connecting i, j on the graph.

The main assumption in this algorithm is that there should be no loops, so that the shortest path from i to j can pass from each node k at most once. The result of the algorithm is deterministic and exact except in the presence degeneracies. It determines the geodesic between all pairs of points.
Note that this algorithm requires to compute all pairwise distances, so it not hte most efficient, one needs the full matrix of geodesic distances. Alternative algorithms exist and will be discussed later.

4.2 Pipeline of ISOMAP

Let us discuss now what is the pipeline of ISOMAP:

- Estimate Δ^{ij} using, e.g., the Floyd algorithm on the graph connecting nearby data points.
- Compute G^{ij} by double centering (Eq. (32)).
- Solve the eigenproblem of G^{ij}.

- Define the chart on $\mathcal{M}$as:
$$\hat{Y}_\ell^i = \sqrt{\lambda}_\ell Y_\ell^i \tag{52}$$
as customary in MDS.

In terms of data, the sequence is:
$$X \in \mathbb{R}^D \to \Delta^{ij} \to G^{ij} \to \hat{Y} \in \mathbb{R}^d \tag{53}$$

Note that we are not reducing the number of data points, just the number of coordinates!

4.3 Potential problems in ISOMAP

There are 3 main issues with ISOMAP:

1. Computational cost due to Floyd algorithm, that scales as N^3. There are alternative algorithms scaling with $N^2 \ln(N)$. This operation is in general expensive;
2. Definition of r_c. If one has data points that are more dense in some regions with respect to others, taking r_c too large will give an underestimation of Δ^{ij}. Instead, taking it too small, may not properly identify minimum paths, and even get disconnected subgraphs. In general it is a tricky situation.
3. Topology. Typical failure is when simple topologies are present such as a circle.

5 Fifth lecture - kernel PCA

The tool we discuss today, kernel principal component analysis, is very flexible; at the same time, we will show that there are many unknowns along the process, that are both challenging and have potential for improvements. The method can be introduce in two manners that are very different: we will start from the defintion of the algorithm.

5.1 Kernel PCA: algorithm

The algorithm works as follows (there is a certain redundancy with the previously defined ones that we stress anyway):

- we start from a feature vector X^i ($N \times D$ matrix);
- we take a *suitable* symmetric function of $X^i X^j$; this symmetric function will be denoted by kernel, and will satisfy:
$$K(X^i, X^j) = K(X^j, X^i) \tag{54}$$
we emphasize that this is a rather low-level assumption;

- perform multi-dimensional scaling based on K:
  1. compute the Kernel matrix $K^{ij} = K(X^i, X^j)$
  2. double centering on K^{ij}, that is we compute G^{ij}:
  $$G^{ij} = -\frac{1}{2}(K^{ij} - \frac{1}{N}\sum_i K^{ij} - \frac{1}{N}\sum_j K^{ij} + \frac{1}{N^2}\sum_{i,j} K^{ij}) \qquad (55)$$
  which is a Grahm matrix;
  3. find eigenvalues and eigenvectors of G.
- now, we have the spectrum $\lambda_\ell \geq 0$ of the Grahm matrix; if the spectrum features a gap, that I know how to approximate it. However, if the spectrum has no gap, I can try to change the kernel! This exploits the huge variational freedom of the algorithm. Taking as Kernel just the standard Grahm matrix, I just recover PCA. What we are doing here is performing non-linear transformation of our data: each of them gives us hope to recover the correct chart of our manifold.

There are two routes to understand kPCA. They lead to different guidelines for choosing the kernels:

1. one can assume that the kernel is related - or coincides - with a suitable distance on the manifold;
2. "standard": the kernel allows extending the feature space from X^i to a larger space including many non-linear functions of X^i.

5.1.1 Route 1: kernel as a proxy of a good distance

The very typical case we have in mind is one where our data points look like in Fig. 4. At very small scale, the intrinsic dimension of the data is 2. However, it is clear that at larger scales, the relevant variable defining my manifold is only 1. The 'transverse' direction can be construed as noise. A good distance shall comply with this qualitative picture: *i)* small distances are all equally small, and *ii)* I do not care about the relative distance of far points, as long as in the reduced representation they remain far. Note that the second assumption is instead violated in ISOMAP - where one cares about long-distances.

A possible choice for a distance according to this is:

$$\Delta^{ij} = 1 - \exp\left[-\frac{||X^i - X^j||^2}{2\sigma^2}\right] \qquad (56)$$

that is almost constant at small distances (parabolic) (condition *i)*), and approaches a constant at large distances (condition *ii)*). The distance coincides with the cartesian distance on a scale determined by σ: at this scale, curvature effects to not play a major role. The choice of σ is of course a key point and shall be performed carefully. A meaningful way to define σ is to set it equal to the length scale where the curvature of the manifold does not play any role: the results low dimensional representation (if it exists) would map the points in a space in which the distances of order σ are correctly reproduced.

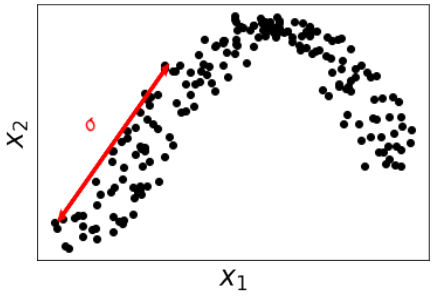

Figure 4: (Temporary Figure) Data with intrinsic dimension $d = 1$.

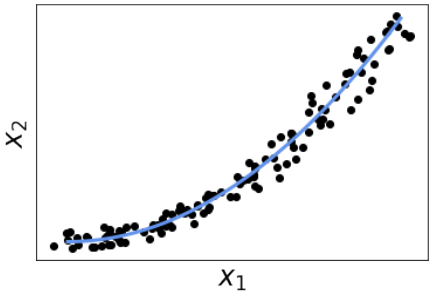

Figure 5: Data distributed on a parabola.

Gaussian kernel. - First, we rescale:

$$X^i_\ell \to \frac{X^i_\ell}{\sqrt{\sum_i (X^i_\ell)^2}} \tag{57}$$

and then define the Gaussian kernel:

$$K(X^i, X^j) = \exp\left[-\frac{||X^i - X^j||^2}{2\sigma^2}\right] \tag{58}$$

the resulting low-dimensional representation (if it exists) maps the points onto a space where the distances of order σ are correctly reproduced.

5.1.2 Route 2: the kernel allows to introduce non linear features

Let us assume to consider feature vectors that lay on a non linear manifold, in Fig. 5 we draw a parabola. PCA being linear is not satisfying as dimensional reduction method in this case. However one could think of extening the space introducing the coordinate $(x^i)^2$. In this manifold the data would lay on a straight line and PCA would be enough to dinstinguish that $d = 1$ and $y^i = (x^i)^2$ is the needed coordinate transformation. The main message is that we need to add the right feature to extend the manifold, however is there a proper way to add the extra non-linear feature? Doing it explicitly as we did in our example is not efficient of course. Indeed, it can be done implicitly by defining a suitable kernel. Let us discuss this by means of an example: we introduce

$$K^{ij} = \sum_l x^i_l x^j_l + (\sum_l x^i_l x^j_l)^2 = G^{ij} + (G^{ij})^2, \tag{59}$$

where the sqare is element-wise. Assuming the embedding space has $D = 2$ we can develope the expression and write:

$$K^{ij} = x^i_1 x^j_1 + x^i_2 x^j_2 + (x^i_1 x^j_1)^2 (x^i_2 x^j_2)^2 + 2x^i_1 x^j_2 x^j_1 x^i_2 \tag{60}$$

We can observe that K^{ij} is a Grahm matrix in an extended feature space we call ψ. Therefore

$$\psi = (\psi_1, \psi_2, \psi_3, \psi_4, \psi_5) = (x_1, x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2) \tag{61}$$

and

$$K^{ij} = \sum_{l=1}^{5} \psi_l^i \psi_l^j. \tag{62}$$

From this example it is evident we can enlarge the feature space defining a non-linear function of the Grahm matrix G^{ij}:

$$K^{ij} = \sum_{m=1}^{M} \alpha_m (G^{ij})^m \tag{63}$$

where the power is element-wise. The greater is m, the more new non-linear features we add. Choosing properly α_m one reconstructs the Gaussian Kernel. The power of the method is evident, it is possible to introduce a degree of non-linearity arbitrarly by definition of K^{ij}.

5.2 Final remarks

What are the properties of a useful and meaningful dimensional reduction?

1. Points nearby in $\{x^i\}$ must not be far away in $\{y^i\}$.
2. Points far away in $\{x^i\}$ must not be close in $\{y^i\}$.
3. d, dimension of the $\{y^i\}$ space, must be kept as small as possible. The only feature that provides evidence of the effectiveness of the dimensional reduction is the spectrum of G^{ij}.
4. Having interpretable $\{y^i\}$. in the sense that it is possible to reconstruct a function $y(x)$ that encodes the coordinate transformation.

Let us observe that in cases where the topology of the original manifold is not equivalent to a hyperplane, all dimensional reduction methods fail to retrieve the intrinsic dimension of $\mathcal{M}$. For example having data laying on a two-dimensional circle it is impossible to satisfy property 1). In fact, cutting the circle at a given point and stretching it on a line would lead to the points at the edges be distant L (length of the circle) instead of being nearby as in the original data set. Performing Kernel PCA to a manifold which is topologically equivalent to a circle would find $d = 2$ instead of $d = 1$. Evidently the number of relevant components in Kernel PCA is not equivalent to the intrinsic dimension of the data, while it provides a lower bound to it. The two are coinciding when $\mathcal{M}$ is topologically equivalent to a hyperplane.
Question: does the relative value between intrinsic and relevant components dimensons give information about the topology of the manifold?

6 Sixth lecture: the intrinsic dimension - I_d - and its estimators

We have already seen that there are cases where the intrinsic dimension of a data set can be estimated by just looking at the spectrum of the covariance matrix (PCA) or of the Grahm Matrix (ISOMAP, kPCA). However If one finds a gap after a certain number m of eigenvalues, this implies that $I_d \leq m$. It is equal if the embedding manifold is topologically equivalent to a hyperplane. At present, our methods only give an upper bound to I_d (under optimal conditions).

6.1 Historial introduction

The idea of intrinsic dimension is connected and mutuated from the language of dynamic systems. A dynamical system is typically described by a feature vector X^t (since we are dealing with time dependent problems), whose time evolution is described by:

$$X^{t+1} = f(X^t) \tag{64}$$

where f is a function, that is typically deterministic, but in general, can also depend on some noise parameter - in this case, it is defined as f_ξ. Note that Markov chain Monte Carlo and molecular dynamics can be written in this form. There are also more complicated forms of dynamical processes (for instance, including dependences on more time steps).

In the 70s, people started studying properties of dynamic systems, including the study of dynamic maps and attractors (see, e.g., in the field of master equations), and the onset of chaos (typically associated to high dimensional manifolds scanned by the dynamics). Among the tools developed at those times, a particularly relevant one are te **Poincareé maps**: since one is dealing with dynamics that is untractable in the full configuration space, one can study projections over a few variables (like 2 or 3) that can be visualized and analyzed. Sometimes, the data points on such projections lie on regions of space that have interesting features.

Characterizing attractors on such manifolds can be done in several ways. One important concept is the **fractal dimension** of an attractor. In particular, some attractors are neither lines nor 2D spaces: they do not fill densely a 2D space, but they are definitely not lines. Very often, such fractal dimension is interesting not only for small d. but also for very large d. Dynamic systems that are chaotic have a large fractal dimension for instance.

The intrinsic dimension is (almost) the same of the fractal dimension. The main difference is that the I_d we are looking at are often not fractal, but integer. It is mostly a small semantic difference.

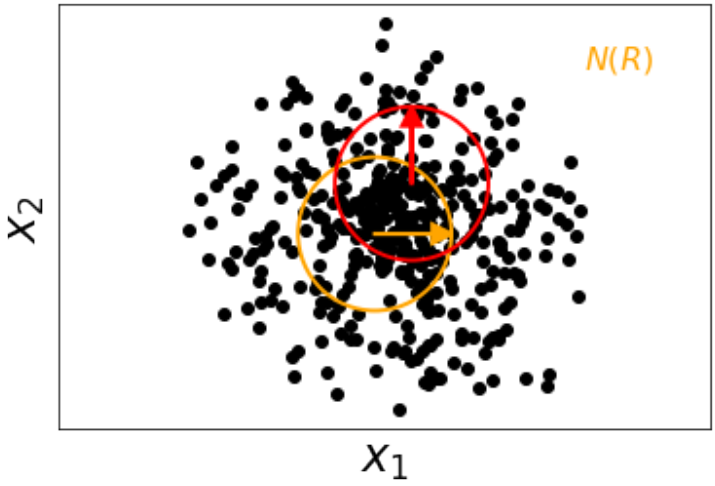

Figure 6: Box counting methods consists in defining a distance R for each point and counting the number of points that lay in the circle of radius R: N(R). In orange and red two circle for two different points in the data set.

6.2 Methods to estimate I_d

The most famous manner to estimate the fractal dimension goes under different names in literature. We refer to this as **correlation dimension**, that was introduced by Grassberger and Procaccia in the early 80s. A small variation of that is called **box counting**. There is then the **Haussdorf dimension** method, that is more rigorous. All of these algorithms give the same answer for a Riemanian manifold. Instead, if applied to fractals, one gets slightly different results. In the following we will describe a few algorithms to estimate the I_d.

6.2.1 Box counting

Let us consider a set of data points, and take one. We then define a distance R and count how many data points lie within this distance, $N(R)$ (Fig.6). This quantity scales as $A \times R^d$, where d is the dimension of the embedding manifold. One then computes $N(R)$ for several values of R, and extract d via linear fitting in log-scale. The quantity $d = \lim_{R\to 0} \frac{d\ln(N(R))}{d\ln(R)}$ extracted in this way is called box counting dimension.
There are a few hidden hypothesis at the basis of this functional definition:

1. the density of data points on $\mathcal{M}$is constant. The problem is that the prefactor A is a density-dependent function, and one is assuming that this is a constant. This can be seen by explicitly writing:

$$\langle N(R)\rangle = \rho\Omega_d R^d \tag{65}$$

where Ω_d is the volume of a unitary d-sphere in $\mathbb{R}^d$ ($\Omega_1 = 2\pi, \Omega_2 = \pi, ...$). The prefactor ρ is a problem since in any practical instance of an estimate, one cannot take the limit $R \to 0$due to the fact that for very small R one would not have any points within in sphere of radius R. If one takes all the distances measured from only one data point and then considers the limit $R \to 0$, the procedure is exact, but typically there are not enough data points. Instead, what one typically does is to consider all the data set. For every point, one measures how many points lie within R, and then perform an average. In fact, we are mixing points where densities are differen. Therefore one would get:

$$N(R) = \Omega_d(\rho_1 + \rho_2 + \rho_3 + ...)R^d \tag{66}$$

which implies that we are mixing variations of the density with the scaling of the number of data points with R. These local densities are in fact R dependent: if $\rho(x)$ is not constant, it is not correct to utilize the scaling $N(R) = \rho R^d$.

2. the manifold $\mathcal{M}$can be curved, in particular at large R. In that case, the distance that I measure in cartesian space is smaller than the one I shall consider (that only goes along the manifold). This problem has been solved by Carnevale and Granata (around 2013). They propose to utilize the geodesic instead of the cartesian distance.

The first problem is very important and one must get rid of the prefactor.

6.2.2 2NN estimator

In this case, the value of d is inferred from the probability distribution of the ratio between the distance of the second and the first neighbor of each point. We now describe the algorithm:

- take a point i, find the first neighbor and compute its distance, r_1^i;
- find the second neighbor and compute its distance, r_2^i;
- compute the ratio $\mu_i = r_2^i / r_i^1$, one can show that this ratio is pareto-distributed regardless of the value of ρ.

A pareto distrubition is defined for $\mu \in [1, \infty]$, and its probability density reads:

$$p(\mu) = \frac{d}{\mu^{d+1}} \tag{67}$$

In the context of box counting, one typically has that $N(R) = \Omega_d \rho R^d$, so one has a probability density that can be written as $N(R|d, \rho)$. Instead, the probability distribution above is insensitive to ρ - since local densities cancel, unless variations are very strong at short distances.

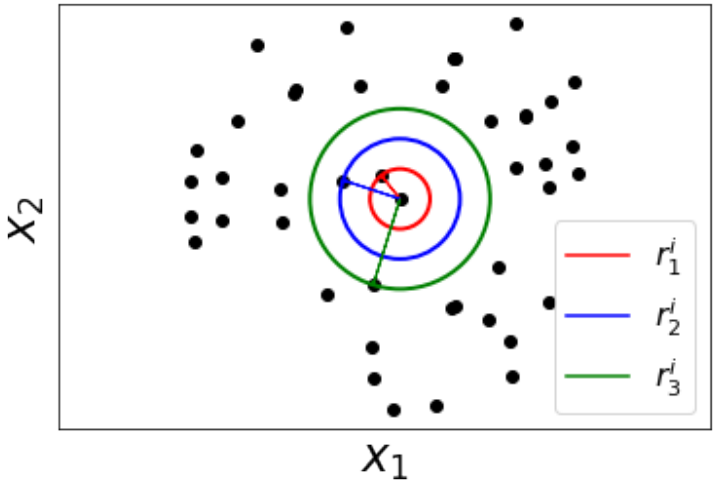

Figure 7: We denote with pedexes 1,2 and 3 the first three neighbours of a reference data-point r_0^i. With different colors we highlight the radius of the hyperspheres $\mathcal{S}_k^i$ centered at r_0^i on which r_k^i lay. The volume V_k^i is the hyperspherical shell between $\mathcal{S}_{k-1}^i$ and $\mathcal{S}_k^i$, $\mathcal{S}_0^i$ being the empty hypersphere.

Protocol of 2NN - Before discussing the last point in detail, we provide an estimator of d. The procedure is the following:

- for all data points, one measures μ^i
- one assumes that the μ^i are harvested independently from a Pareto(d):

$$P(\mu_1, \mu_2, ...\mu_N) = \prod_{i=1}^{N} \text{Pareto}[\mu_i] = \prod_{i=1}^{N} \frac{d}{\mu_i^{d+1}} \tag{68}$$

- one can apply either maximum likelyhood or Bayesian inference to extract I_d.

For the case of maximum likelyhood, what we do is that we maximize $\ln P$ with respect to d. Since the logarithm is monotonic, it is equivalent to maximize the probability. With a little algebra, one gets the quantity

$$\ln P = \sum_i \ln \frac{d}{\mu_i^{d+1}} = \sum_i (\ln(d) - (d+1)\ln \mu_i) \tag{69}$$

to be maximized. Since it is a convex function, we can just set its derivative to zero:

$$\frac{\partial \ln P}{\partial d} = 0 \longrightarrow \frac{N}{d} - \sum_i \ln \mu_i = 0 \tag{70}$$

that implies:

$$d = \frac{N}{\sum_{i=1}^{N} \ln \mu_i} \tag{71}$$

which is our estimator of I_d.

Justification of the Pareto-distribution assumption. - We now review the most important property that we mentioned above, that is, that μ_i is Pareto-distributed and depending on I_d.
Taken a first point i, we can draw a sphere that passes through the first neighbor (Fig. 7 as a reference). The volume of this hypersphere is:

$$V = \Omega_d r_1^d \tag{72}$$

This volume is distributed as follows:

$$V \sim \exp[\rho]. \tag{73}$$

The symbol $\sim$ just means that the values on the left are harvested from the distribution on the right. We can then derive the probability density of the volumes as

$$P(V) = \rho \exp[-\rho V] \tag{74}$$

since we are considering a set of points harvested from a constant probability density ρ. Note that it is possible to derive some of these properties directly from the Poisson distribution.
We now take into account also the second neighbor. We are generating two volumes, the second one being the hypersphere of the 2NN. In fact, one can construct a sequence of hyperspherical shells, each one with one data point in the internal border, and one in the external one. All this hypershells have volumes that are harvested from the same distribution! So one gets:

$$V_1, V_2, V_3 \sim \exp[\rho] \tag{75}$$

where explicitly one has:

$$V_2 = \Omega_d (r_2^d - r_1^d) \tag{76}$$

The Eq. (75) above can be proved. Now, given that both V_1 and V_2 are exponentially distributed, we want to find the probability distribution of the quantity:

$$\lambda = \frac{V_2}{V_1}. \tag{77}$$

One gets:

$$P(\lambda) = \int_0^\infty dV_1 \rho e^{-\rho V_1} \int_0^\infty dV_2 \rho e^{-\rho V_2} \delta(\frac{V_2}{V_1} - \lambda), \tag{78}$$

that can be calculated as

$$P(\lambda) = \int_0^\infty dV_1 \rho^2 e^{-\rho V_1} e^{-\rho \lambda V_1} = \frac{1}{(1+\lambda)^2}. \tag{79}$$

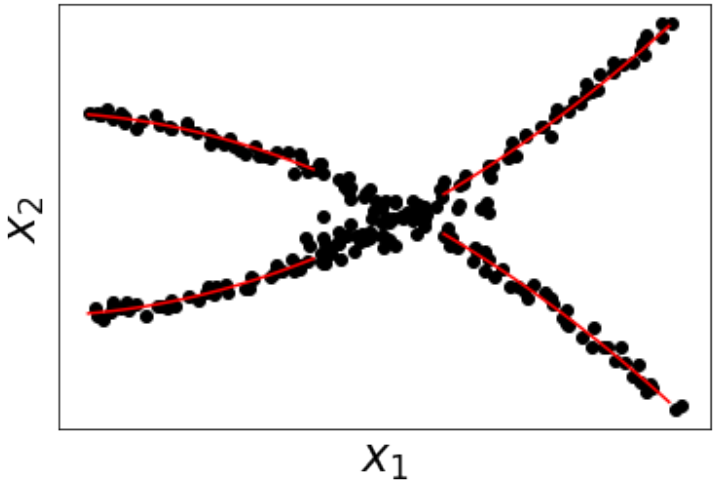

Figure 8: Example of a 2D dataset. In orange the lines along which $I_d \simeq 1$.

Here the dependence on ρ has completely disappeared, our method is thus density independent.
Calculated the distribution of λ, one can easily obtain the one for μ, since:

$$\lambda = \frac{V_2}{V_1} = \frac{\Omega_d(r_2^d - r_1^d)}{\Omega_d r_1^d} = \left(\frac{r_2}{r_1}\right)^d - 1 = \mu^d - 1 \tag{80}$$

and using Eq. (79) on:

$$p(\mu)d\mu = p(\lambda)d\lambda \tag{81}$$

one gets:

$$p(\mu) = \frac{d}{\mu^{d+1}} \tag{82}$$

which is what we wanted to prove.
In practical applications, we have overcome a major problem. Moreover, since the density can be very inhomogeneous, we want to use only the first two 2NN, as we really need to harvest data from the same distribution.

6.3 The I_d and scale dependence for real world applications

Let us start form an example which is exemplified in Fig.8. At small scale it is evident that $I_d = 2$. One can see it as the number of linearly independent direction where I can see data point). At very large scale, $I_d = 2$ again due to the fact that our manifold is topologically complex and has curvature. Instead, at intermediate scales, for the vast majority of the data points, $I_d \simeq 1$: the

variations along the direction orthogonal to the main coordinate are negligible. This is marked in particular using red lines.

Suppose that we can estimate I_d as a function of scale, i.e. distance between points:

- at at small scale, one would get $Id = 2$;
- at intermediate scale the curve would approaches one smoothly;
- at large scale $I_d = 2$ again.

If one wants to characterize a manifold, it is necessary not to have a scale dependence. However, in any realistic application, one always finds highly non-trivial dependence like in the example above.
The simplest manner to treat this scale dependence is the following, called *decimation*. We can associate the estimator of d we use to a given scale. For instance, for 2-NN, we can associate a scale corresponding to the average value of the first nearest neighbor distance:

$$s \simeq \langle r^i_{(1)} \rangle \tag{83}$$

that is supposed to be the dominant scale. This procedure returns a point (I_d, s). As a next step, we decimate the data set: for each X points, we keep 1. Then we get another point (I_d, s) using the same procedure. W can repeat this procedure recursively as $s = \langle r^i_{(1)} \rangle$ increases. This approach could give us insights on the dependence of I_d with respect to the scale.

Following to be clarified **Example on classical trajectories -** Let us take 1000 particles in 3D, I_d=2993 (7 constraints due to Hamilton dynamics). However, very likely, only very few minimal directions are important. This means that looking at the system at different scales whould give a different I_d. The point is that one needs to have an idea of the meaningful scale, and assume scale invariance/ (which implies a constant I_d over a scale of length). Then one could look for a plateux in I_d as a function of s and take that I_d as a good estimate. Multiple plateaus can also appear when the topology of the embedding manifolds is a foam. For instance, this happens for a partition function with more than one lengthscale. An open issue is to identify bifurcation points, which is not trivial in high dimension. Another question could be how to identify regions of different intrinsic dimension, however in this last case a solution has been obtain recently.

7 Seventh lecture: brief intro to neural networks

Neural networks are becoming key tools to perform unsupervised learning. The starting point of our discussion will be critical problem that is present in kPCA, which is the best method we have introduced so far to perform dimensional reduction. For kPCA is very challenging to find a good kernel function. A good kernel is surely related to the most appropriate distance measure, that

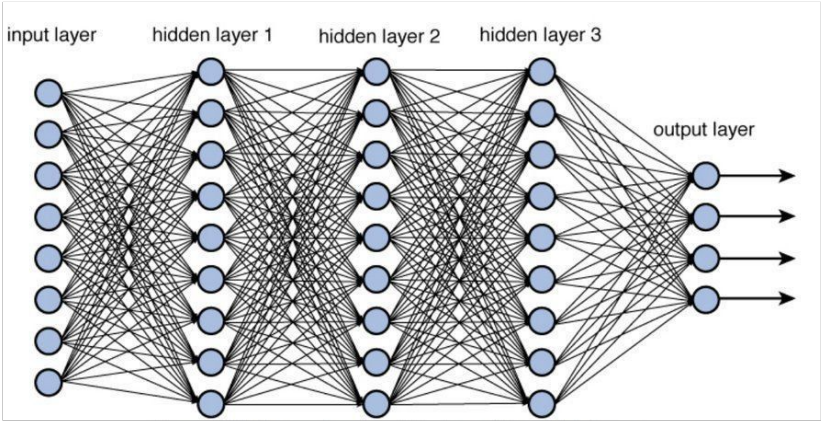

Figure 9: Sketch of a deep neural network. Each layer can have arbitrary dimension.

can capture all the relevant information at a given scale, which might change within the data set. KPCA is based on the expectation that, with an explicit non-linear transformation, one is able to perform a direct dimensional reduction, from X to Y.
The question is: why would neural networks perform better?
Instead of attempting this transformation directly, neural networks can do it in intermediate steps:

$$X \to X' \to X'' \to ... \to Y \tag{84}$$

where each of these steps is a non-linear transformation and a dimensional reduction. Neural networks become very useful since, being them deep (contain many transformations in sequence), can exploit the fact that a sequence of mild non-linearities can produce arbitrarily complex non-linearities.

7.1 Brief review of NN properties

Let us assume to work with a data set X^i_ℓ where $i = 1, ..N$ and $\ell = 1, ..D$ are data points and number of features, respectively. A layer of the neural network is basically a function that maps $\bar{X}^i \in \mathbb{R}^D$ into a vector $\bar{X}^{(1)i} \in \mathbb{R}^{D^{(1)}}$. We will denote with

$$\bar{X}^i \equiv \bar{X}^{(0)1}, D \equiv D^{(0)} \tag{85}$$

the leftmost part of the neural network. Explicitly, one can write:

$$\vec{X}^{(1)} = \vec{f}_\pi(\vec{X}^{(0)}) \tag{86}$$

where f is an explicit vector function, based on a set of parameters π.
One can take simple choices for the transformation. For example:

- a linear transformation such that:

$$\vec{X}_\ell^{(1)} = \sum_{m=1}^{D^{(0)}} \pi_{m\ell}^{(0)} X_m^{(0)}. \tag{87}$$

  Here if the object $\pi_{m\ell}^{(0)}$ is defined for $\ell = 1, ...D^{(1)}$ and $D^{(1)} < D^{(0)}$, one performs a dimensional reduction (equivalent to PCA in a single layer).

- non-linear activation function:

$$\vec{X}_\ell^{(1)} = g(\sum_{m=1}^{D^{(0)}} \pi_{m\ell}^{0} X_m^{(0)}) \tag{88}$$

  Where g can be chosen almost arbitrarly. A common choice is the switching function, interpolating between zero and one as an arctangent, in analogy with neurons in the brain or a RELU function(0 until some point, then linear). However there are a lot of activation functions that can be considered as sigmoids or hyperbolic tangents.

In a deep neural network, one concatenates many such functions, $f_{\pi^j}^j$. We will denote with L the number of layers (a sketch of the neural network is presented in Fig.9). This is not the most general neural network, it is of a particular type called **feed-forward neural network**, whose key property is that the value of the activation at the layer $\ell + 1$ only depends on the values of $X^{(\ell)}$. This is akin to a Markov process. The number of parameters is of course very large but there exist procedures to perform the learning efficiently.

7.2 An example: the simplest possible task

As an example let us consider binary classification which is not unsupervised, but is simple to introduce the problem of feed-forward neural networks. For binary classiifaction $D^{(L)} = 1$, so that $X^{(L)}$ is a single real number: we can use this, setting a threshold, to classify the input. For example, feeding to the network images of cats and dog, if $X^{(L)i} \geq 0$, the image could correspond to a dog otherwise to a cat. To achieve this result the network must be trained in order to recognize the features of dogs and cats. The training procedure is the following: we consider the set of data $\vec{X}^i$ and, for each data point, set a so called *ground classification* b which relates a number $\{0, 1\}$ to the picture (which can be related to a classification like dog/cat). The predictive performance of a network is quantified by the **loss function**:

$$\mathcal{L} = \sum_{i=1}^{N} \left(b^i - g(X^{(L)i} - \pi^{(L)}) \right)^2 \tag{89}$$

where π^L is a single number that makes our classification consistent. The loss function depends explicitly on all the parameters, since:

$$X^{(L)} = f_{\pi^{(L-1)}}^{(L-1)}(\vec{X}^{(L-1)}) = f_{\pi^{(L-1)}}^{(L-1)}(f_{\pi^{(L-2)}}^{(L-2)}(\vec{X}^{(L-2)})) = ... \tag{90}$$

This implies that my activation at layer L is an explicit function of the input, and maps $\mathbb{R}^D$ into $\mathbb{R}$:

$$X^L = F_{\pi^{(0)},\pi^{(1)},\ldots,\pi^{(L)}}(\vec{X}^0) \tag{91}$$

which can be seen as the fact that doing supervised learning is nothing but enforcing dimensional reduction (so knowing the final dimension is useful). In a good neural network, we would like to have:

$$F_{\vec{\pi}}(\vec{X}^{(0)i}) \sim b^i \tag{92}$$

so that given a picture as an input, the neural network returns exactly the ground classification. To do so we minimize the loss function via gradient descent. This means that one start from the guess of a vector of parameters and proceeds recursively using

$$\vec{\pi} = \vec{\pi} - \epsilon \nabla_{\vec{\pi}} \mathcal{L} \tag{93}$$

where ϵ is the learning rate. At each step the parameters change and the loss function is calculated. Then again a minimization is performed to verify it the parameters are optimal. In a feed-forward neural network, the gradient can be computed in a simple and closed for.! The reason is that my function f is a simple composite function and the chain rule can be used. For simple scalar function infact:

$$\frac{\partial F}{\partial \pi} = f' \frac{\partial g}{\partial \pi}; \tag{94}$$

The chain rule is often referred to back-propagation because one calculates the gradient starting from the last layer and going back ccomputing all the derivative. Usually the analytical expression of the derivatives is very simple and the back-propagation can be performed quite efficiently at each step of the minimization.

Comment on the depth of the neural networks - In principle, even with a single layer, it is possible to reproduce arbitrary functions within the setting we described above. However, it shall be noted that, when not deep enough, the neural network will not work in practical applications. This is especially true of feed-forward neural networks with simple activation functions.
For instance, considering the loss function above: suppose that there is a minimum at π_{best}. If the size of the parameter space is much smaller than the training set:

$$\text{size}(\pi) \ll N \tag{95}$$

then, the landscape is very glassy, just because of undertraining, so that there are plenty of local minima that satisfy $\nabla_\pi \mathcal{L} = 0$. On the other hand if one has a given number of parameters N, it is not efficient to fit small data sets $M \ll N$, as there are infinite ways of doing it. Originally, the maximum number of parameters that was assumed to be manageable was $N/10$, in order not to encounter any overfit. In the early 2000s, it was realized that one can try to train the data with architectures where, instead of using few intermediate layers, one has $L \simeq 10, 20$ hidden layers. They were using parameter spaces of order

$10^3/10^4$ per layer. This resulted in fitting data spaces of order 10^4 with 10^6 parameters, apparently violating typical understanding based on overfitting. In fact, what happens is that if

$$\text{size}(\pi) \gg N \tag{96}$$

then, the loss landscape shall not be glassy anymore. This problem is however not fully understood. The landscape remains characterized by the presence of local minima, as the function is non-linear, but there is a region in parameter space that is 'large', where the loss is 0! The dimension of this region is:

$$\dim(\mathcal{L} = 0) \simeq \text{size}(\pi) - N \tag{97}$$

This way, the problem of glassiness was overcome. However, how can one avoid overparametrization? There is presently no theory that address this. The two ingredients to avoid overfitting are:

- regularization: basically, the training of the neural network is done by minimizing the loss, with the addition of a regularization function:

  $$\mathcal{L}(\pi) + \text{Reg}[\pi] \tag{98}$$

  that could be somehow similar to langrange multiplier, like $\lambda \sum_\alpha (W_\alpha)^2$.

- stochasticity: while we have said that one typically uses gradient descent to minimize the neural network, it is also true that the loss can be rewritten as $\mathcal{L} = \sum_{i=1}^{N} \mathcal{L}_i$. One can then chose randomly M points among the N ones, and then compute the gradient only on those. This random set changes at every step. This is equivalent to add noise to the gradient descent - a noise that is very non-gaussian. This procedure is called **stochastic gradient descent**. Remarkably, this is able to find solutions that avoid overfitting. Moreover, it is also computationally more handy, as one is dealing with a much simpler optimization procedure (one computes only very few loss points, not all of them). In practice thus, SGD is used almost always just for computational reasons.

8 Eight lecture: autoencoders

An autoencoder is a deep neural network in which the size of the output layer is equal to the size of the input layer, in formulas:

$$D^{(0)} = D > D^{(1)} > ... > D^{(L/2)} < D^{(L/2+1)} < .. < D^{(L)} = D. \tag{99}$$

The intermediate layer is called bottleneck. A sketch of an autoencoder is shown in Fig.10. Ideally, one would like to have

$$D^{(L/2)} \geq d \tag{100}$$

where I_d is the intrinsic dimension of $\{X\}$. Which means that the dimension of the bottleneck is equal to the intrinsic dimension. If this is the purpose of the

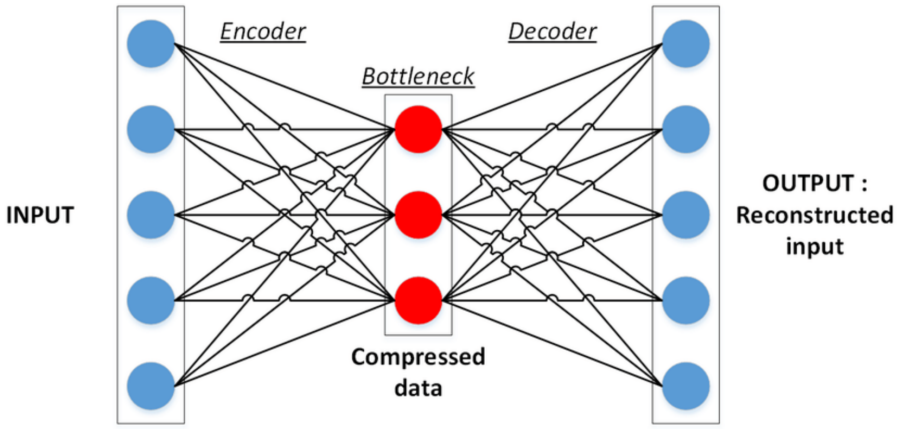

Figure 10: Sketch of an autoencoder with a single intermediate layer.

autoencoder, one has to define the form of the loss function with the specific goal of making the representation is the bottleneck as equivalent as possible to the Y variables (the chart on the lower dimensional but fully descriptive coordinates). We write this as follows:

$$\mathcal{L}(\pi) = \sum_{i=1}^{N} ||\vec{X}^{(L)i} - \vec{X}^{i}||^2. \tag{101}$$

We need to add the requirement that the input is approximately equal to the output:

$$X^{(0)} \sim X^{(L)}. \tag{102}$$

The first part of the network, from X to Y, is called encoder. The second part, from Y to $\tilde{X}$, is called decoder. The advantage of all of this is that the function $Y(X)$ is explicit. This scheme is in principle extremely powerful, because it combines deep neural networks to represent complicated non-linear functions, but at the same time has an explicit mapping to Y. However, one still faces the challenge of training the autoencoder.
It is important to stress that they will likely not work if you fix the size of the autoencoder to smaller than I_d!

8.1 A simple example: autoencoder with two layers and PCA

Let us consider a simple example with just 2 layers (Fig.10), with f^1, f^2 both linear:

$$f^1(X^0): \quad X_\ell^{(1)} = \sum_{m=1}^{D} \pi_{m\ell} X_m^{(0)} \tag{103}$$

$$f^2(X^1): \quad X^{(2)}_\ell = \sum_{m=1}^{d} \pi_{m\ell} X^{(1)}_m \tag{104}$$

Then, the loss can be shown to be quadratic function of the parameters, and one finds that there is a single value of π such that

$$\nabla_\pi \mathcal{L} = 0 \tag{105}$$

that corresponds to the PCA result.

8.2 Why do we need regularization? and how to overcome

Suppose that we know nothing about a model and wish to have a very compact representation, $D^{L/2} = 1$. If one allows a sufficiently complicated non-linear function to act on a single real number, one can make it work. However, this modelling cannot be meaningful for realistic dataset like images. In fact each input would be just associated to a different real number and the autoencoder would just work as a **memory-storage**, that is it would not have any predictive power for data outside of the training network. Memory storage is a specific example of overparametrization. In general, one needs a sufficiently large bottleneck which depends on the intrinsic dimension of the dataset. There are different types of autoencoders, we list three here:

1. denoising autoencoders;
2. sparse autoencoders;
3. variational autoencoders.

8.3 Denoising autoencoders (DA)

In DA, while training, instead of feeding into the network an image and asking it to recover exactly the same image, we slightly perturb it with noise, like a local deformation, that is small enough that the image is still recognizable. My loss function is then:

$$\mathcal{L} = \sum_i ||X^{(L)i}(\tilde{X}^i) - X^i||^2. \tag{106}$$

Then one minizes the difference between the target image on the right, and the noisy image on the left. All of this has to be done stochastically, i.e. the inputs must be changed independently one from the other. The deformation can also be changed at every step of the gradient descent:

$$W^{t+1} = W^t - \epsilon \nabla_W \sum_i ||X^{(L)i}(\tilde{X}^i) - X^i|| \tag{107}$$

this way, we are enforcing the network to avoid memorization so that it is robust with respect small deformations. One could, in principle, also use larger training sets with more images (each with different noise): the procedure can be considered roughly equivalent, but ocmputationally heavier.

8.3.1 Variational autoencoders (VA)

The functioning of VA is rooted in variational inference: essentially, one is interested in reproducing a given probability distribution. The main idea is the following: each data point X^i *d*oes not correspond to a single reduced-dimensionality feature vector Y^i. Instead, we assume that

$$Y^i \sim P(Y|X^i) \tag{108}$$

that is, that Y^i is harvested from a given probability distribution. The latter is a different probability distribution for each X^i.
Itimplies that dimensional reduction moves from being a deterministic operation, to requiring the characterization of a probability distributions. This aims to reproduce not only the image, but also its 'stability'. There are several ways to perform this, let us discuss one. We assume that each feature is harvested from a Gaussian distribution. Each component of the feature vector is harvested from a normal distribution:

$$Y^i_\ell \sim \mathcal{N}(\mu^i_\ell, \sigma^i_\ell) \tag{109}$$

So I am learning a different value of the average and standard deviation for each data set separately. In the bottleneck, we will have a collection of nodes, for each μ_ℓ and σ_ℓ. The parameters $(\sigma_\ell(X^i), \mu_\ell(X^i))$ are then connected together such that their autoput is $Y^i = \mu(X^i) + g\sigma(X^i)$, where g is fed by a gaussian number generator. In this way the output of the nodes $(\sigma_\ell(X^i), \mu_\ell(X^i))$ is noisy. This is equivalent to what is done with denoising autoencoder but it is more stable because it is built in the autoencoder itself.
To be precise we can say that the structure of a VA is such that the bottleneck is bipartite (contains both μ and σ). Each of these two subnodes is combined into a single node including gaussian inputs and is followed by the usual decoder. In this way stochasticity is implicit in the construction of the encoder. It is important to stress out that all of this works under the assumption of uncorrelated variables: if correlations are present, the distribution space is considerably more complex.

9 Ninth lecture: clustering

[[temporary version!!!]] VV:This chapter is not clear in several points. Formulas must be checked.

9.1 Introduction to clustering

While until now we have mostly dealt with more math-based methods, moving to clustering will require us to change gears, as the very definition of cluster is somehow vague.

A good intuition can be gather by observing real world data sets. They are usually difficult to encode because they present very different features and each data point may also differ from the others. A simple example to take into account is a set of stones. Let us consider some possible features like weight, size, color (wavelength) or rugosity. In generality one may distinguish *raw characteristics* and *learned characteristic*. Among raw characteristic we can enumerate, for example,

- number of features / dimension,
- number of samples / cardinality,
- types of features / real (weight, rugosity), cathegorical (shape), integers.

Among learned characteristics instead one could think about

- statistics,
- intrinsic dimension,
- probability density,
- clustering.

When dealing with a large amount of data one tries to put together what is similar, and apart what is not. This procedure can be called *clustering*. However, it is important to point out that clustering is not equivalent to classification: clustering generates groups, but does not label them. Labeling is something that requires an additional operation.

9.1.1 Feature selection

Feature selection is particularly important for performing clustering. Typically, we need to reduce the number of features due to the curse of dimensionality. This may become problematic in the case of large data sets. Or they can be multiple features that can lead to clustering and it is necessary to choose which one to consider or how many.
The answer to feature selection is actually very dependent on the problem we are interested in solving, and might require feedback from a preliminary analysis (that typically requires the whole process to be carried out). Also, remember that feature selection depends on *labeled data*, so it is supervised learning: in this respect, it is fundamentally different from dimensional reduction.
At the formal level, we need to correlated the features to the *ground truth* (our ultimate goal).

The first technique that is used to perform feature selection is called **variable ranking**. This consist in identify which variable better explains the various labels, and quantify it. This can be done via evaluating a linear correlation coefficient:

$$R(i) = \frac{\sum_{k=1}^{m}(x_{k,i} - \bar{x}_i)(y_k - \bar{y}_i)}{\sqrt{\sum_{k=1}^{n}(x_{k,i} - \bar{x}_i)^2 \sum_{k=1}^{m}(y_k - \bar{y}_i)^2}} \tag{110}$$

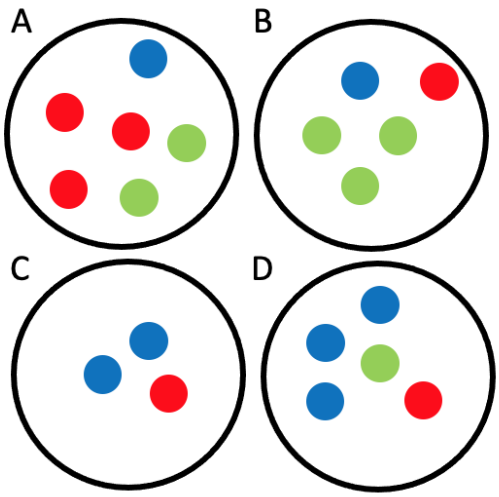

Figure 11: Three class problem (red, green, blue), discrete variable assuming 4 values (A,B,C,D).

It works for continuous variables and outputs, and relies on the goodness of the linear fit. // Another method is to utilize **single variable classifiers**. Oftentimes, those rely on the establishing a correspondence between pure variables and ground truth on few features. // Finally, another method relies on *mutual information between variables and targets*. It quantifies how much information about your answer you have in each variable. For continuous variables, the mutual information reads:

$$I(i) = \int_{x_i} \int_y p(x_i, y) \ln \frac{p(x_i, y)}{p(x_i)p(y)} dx_i dy \tag{111}$$

Similar definitions hold for the discrete space. This allows us to rank the various variables. Let us discuss a simple example. Imagine a three class problem with a discrete variable which can assume four different values (A,B,C,D). We denote the classes with the colors red, green, blue. For a graphical aid we can refer to Fig.11. In this discrete case the probability is estimated through frequency counts. The mutual information becomes a sum over all the possible combinations of colors and groups (A,B,C,D). Referring to Fig.11 one could evaluate the probabilities of the several combinations of colors and clusters, for instance $p(A) = 6/19$, $p(A, \text{red}) = 3/19$, $p(\text{red}) = 6/19$. Calculating explicitly Eq. 111 on gets $I \simeq 0.17$ thus obtaining the degree at which the features are related. Aposteriori, it is possible to assess whether discarding one of them because irrelevant. (to be checked)

Challenges in variable ranking - The first challenge of features selection is how to treat redundant variables. While typically you do not want them, they can still be of help in reducing the noise. Sometimes, a variable that is useless by itself, can be of great use when considered together with others! Variable ranking will miss this. Notice that redundancy can also be measured by correlations. // Coming back to our stone data, let us make an example. If we use single variable ranking, and we use only a single feature like weight or volume, we would miss any possible classification. However but if one includes both of them, then can evaluate the density, and classify the stones in this regard. This example illustrates the idea of **subset selection**. One can utilize a wrapper to assess the usefulness of a given subset of variables. Search in the space of variables is not easy (brute force is NP hard).[2] It is also not clear how to assess the performance of this prediction, and guess which machine learning one can use to perform this operation.

9.1.2 Similarities and distances

Why are we working with distances? The idea is that sometimes it is actually easier to define a distance (similarity) between two objects, rather than relying on a direct representation. Let us, for instance, consider a data-set consisting of arrays of nucleotides, here we list two of them:

1. AACDPGGGADP
2. CCDPGGADACG

We are used to methods based on points in a high-dimensional space, while here we are left with strings of letters each pointing to a different nucleotide chain. Remarkably one could define a similarity measure S_{ij} (or distance d_{ij}) among the data points. In the example above one could consider the number of nucleotides of item 1 which are equal and similarly ordered with respect to item 2: highlighting them in red AACDPGGGADP one could estimate $S_{12} = 7/11$ or $d_{12} = 4/11$.
Once distances are defined, problem dependently, it is easy to define points in space simply by geometry (later on, we shall discuss whether those distances are metrics or not).
Note that distances are never fully unsupervised.
Clustering just tries to separate data *naturally*, in such a way that similar elements lay within the same cluster, while dissimilar elements belong to different ones. In this context, similarities S_{ij} and distances d_{ij} are naturally defined. Both definitions depend strongly on the nature of the features. Still, the two quantities are not the same.
A metric distance shall satisfy:

1. symmetry: $d(x, y) = d(y, x)$

[2] Even evaluate conditional mutual information is NP hard in these cases. [Can one use the equivalent of negativities for density matrices. The equivalent of transposition could be re-labelling]

2. non-negativity $d(x, y) > 0$
3. identity of indiscernibles: $d(x, y) = 0 \Longleftrightarrow x = y$
4. triangle inequality (not necessarily satisfied by a similarity!) $d(x, z) + d(z, y) \geq d(x, y)$

Only the first three properties are present in case of similarities.

Let us list here some examples of distances. Among metric distances we have

- the Minkowski distance:

$$d_{ij} = (\sum_{\ell=1}^{d} |x_{i\ell} - e_{j\ell}|^p)^{1/p} \tag{112}$$

  that reduces to Euclidean (p=2), city-block (p=1), or supremal for large p.

- Mahalanobis distance:

$$d_{ij} = (X_i - X_j)^T C^{-1} (X_i - X_j) \tag{113}$$

  where C is the covariance matrix. This is applied in case in which features have really different nature and range.

Some examples of non-metric distances are:

1. Pearson correlation,
2. point symmetry distance (typically used if it is known that there is a particularly important point in the sample),
3. cosine-similarity (often applied if the actual number is not important, but if two points are in the same direction),

$$S_{ij} = \frac{x_i^T x_j}{||x_j||||x_i||}. \tag{114}$$

If one uses qualitative features, one can exploit different distances:

1. Jaccard similarity

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}, \tag{115}$$

2. Hamming distance, which is the one we used before in our nucleotide example

$$H(A, B) = |A \cup B| - |A \cap B|, \tag{116}$$

In general, working with metrics can considerably simplify clustering, and improve the performance of an algorithm. One very good working example is the geodesic distances.

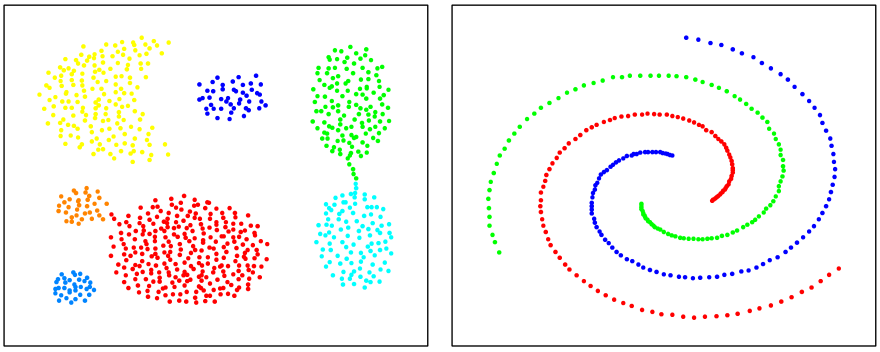

Figure 12: Images of data divided in clusters. For each image, clusters are highlighted with different colors

9.1.3 Clustering

First of all, what is a cluster? There is not a mathematical definition, likely because there are different clustering mechanisms that call for different definitions. In Fig. 12 we show some examples of data divided in clusters. In general, clustering is a data mining technique that can be used for several tasks, including search and selection in data bases, image recognition and classification. Clustering depends on the metric and the features that have been chosen.

The types of clustering are various and can be classified in terms of the output of a the clustering algorithm utilized: flat, fuzzy and hierarchical clustering.

- In flat clustering, we have hard partition of the data. Each element is associated to an integer
$$Cl(i) = \ell \tag{117}$$
that labels the cluster. The number of clusters is traditionally assigned as an input. There are typically no multi-level structures;

- Fuzzy clustering has soft element assignation (e.g., one data point can belong x% to some cluster, and (1-x)% to another):
$$\vec{Cl}(i) = (u_1, u_2, ...u_k) \tag{118}$$
where u_i is the degree of membership and its sum is normalized to 1. The number of clusters is traditionally assigned as an input. It may be difficult to transform this into a hard partition;

- hierarchical clustering: it produces a set of nested clusters organized as a hierarchical tree (dendrogram). There is no assumption on the number of clusters, and can be transformed in many hard-partitions. This clustering can correspond to taxonomies (that is, can be related to meaningful classifications). Note that this is not related, in general, to applying a series of flat clustering.

9.2 K-means algorithm

This algorithm was introduced in the '60s, and is still widely used. It attempts to minimize the intracluster distance while maximising intercluster distance. It is based on the context of **cluster centroid** (average position of the elements within a cluster), and can easily parallelized and linearized. The user must provide the number of clusters k as an input, thus it requires an apriori guess of the number of clusters.
One defines a so called *objective function* whose minimization leads to maximising the intercluster distance an minimizing the intracluster one. In K-means it is the following

$$O(z) = \sum_{\ell=1}^{m} \sum_{i=1}^{n} \delta_{z_{ij}} ||\vec{x}_i - \vec{c}_\ell|| \tag{119}$$

where the $\delta_{z_{ij}}$ checks the membership of a point with respect to a cluster, and the rest gives the distance from the centroid (whose position is $\vec{c}_\ell$). Minimizing the objective function is NP-hard. The algorithm we apply tries to locally minimize it. It requires in input the data points $\{\vec{x}_i\}_{i=1}^n$ and the number of clusters$k$ and gives in outputthe cluster membership assignements $\{z_i\}_{i=1}^n$ The procedure to obtain the output is the following.

1. Initialize $\vec{c}_l$ randomly, from the data set.
2. Repeat:
   - Fix $\{z_i\}_{i=1}^n$ by assigning $\{\vec{x}_i\}_{i=1}^n$ to the nearest centroid.
   - Recompute the centroides according to $\{z_i\}_{i=1}^n$:

$$\vec{c}_l = \frac{\sum_{i=1}^n \delta_{z_i,l} \vec{x}_i}{\sum_{i=1}^n \delta_{z_i,l}}$$

3. Stop when convergence is obtained

This procedure converges to the local minimum of the objective function.

9.2.1 K-means: weaknesses

Here we discuss the problems one could encounter using K-means algorithm.

1. The algorithm is sensitive to initialization, since it is a local optimization
2. The choice of k is an apriori input.
3. It is very sensitive to outliers.
4. It uses euclidean distances.
5. It only works for spherical clusters.

Inizialization - One can get stuck if the initial points get too close, or if the cluster space is weird. One can run several simulations with different initializations. Otherwise, one can use a different algorithm which improves K-means called K**-means++**. It works as followsL

- initialize the first cluster randomly, from the data set.
- repeat until all k centers are found:
  1. Compute $D_k = \min(x, c_k)$ for each x.
  2. Extract a new center c_k belonging to the data points according to probability $p(x) \approx D_k^2$.
- run k-means with the selected centers as initialization.

Choice of k - One can use the so called Scree-test. First compute the dissimilarity (the loss function above) for various k. This will decrease with k (unless one gets stuck in a metastable minimum). At some point, there will be a clear change in the improvement rate: the optimal k is the one where this improvement rate start to change.

Sensitivity to outliers - If an outlier is present, the centroid of the cluster can be very off where most of my points are. One can use k-medoids: instead of working with centroid, it works with the *most central element* (not the average of the position). Note that this can be used with all distances.

Limitation to spherical clusters - This is particularly clear if the clusters are very packed by deformed: those will be partitioned by k-means. This is due to the definition of distance utilized (euclidean) and can be overcome using kernel k-means.

10 Tenth lecture: theory and applications of clustering algorithms

We remind from the previous sections that working with distances (instead of features) is a way that is often very practical when dealing with realistic data sets, allowing them to be transformed from complex to tractable structures.

10.1 Kernel k-means

Originally, k-means only computes distances from the center of the clusters. Here we are interested in dealing with non-linear clusters, using the kernel trick. It proceeds in 3 steps:

1. we project the data set into a feature space by means of a non-linear mapping: $\vec{x}_i \to \vec{\phi}_i$;

2. distances in the feature space can be computed with using a kernel K:

$$||\vec{\phi}_i - \vec{\phi}_j||^2 = K^{ii} + K^{jj} - 2K^{ij} \tag{120}$$

3. the loss function and the centroids are analogous to the standard ones in k-means, just applied in feature space, for instance:

$$O(z) = \sum_{j=1}^{k} \sum_{i=1}^{n} \delta(z_i, \ell) ||\vec{\phi}_i - \vec{\nu}_\ell||^2 \tag{121}$$

where $\vec{\nu}_\ell$ is the position of the centroid ℓ.

While we cannot compute the position of the centroid explicitly, we can compute the value of the distances:

$$\begin{aligned} d_{ij}^2 &= (\vec{\phi}_i - \vec{\nu}_\ell) \cdot (\vec{\phi}_i - \vec{\nu}_\ell) = \vec{\phi}_i \cdot \vec{\phi}_i - 2\vec{\phi}_i \cdot \vec{\nu}_\ell + \vec{\nu}_\ell \cdot \vec{\nu}_\ell = \\ &= G^{ij} - 2\frac{\sum_{j=1}^{n} \delta(z_i, \ell) G^{ij}}{\sum_{j=1}^{n} \delta(z_i, \ell)} + \frac{\sum_{j=1}^{n} \sum_{k=1}^{n} \delta(z_j, \ell)\delta(z_k, \ell) G^{kj}}{(\sum_{j=1}^{n} \delta(z_j, \ell))^2}. \end{aligned} \tag{122}$$

Obtained the distances, the algorithm proceeds as follows:

1. compute the Gramm matrix,
2. randomly pick k centers,
3. iterate the calculation $z_i = \arg\min_\ell(d_{ij}^2)$ until convergence of zi.

10.2 Fuzzy c-means: a fuzzy clustering algorithm

The basic idea of this method is to consider a soft-assignation criteria within k-means:

$$\vec{C}l(i) = (u_1, u_2, \ldots, u_k) \tag{123}$$

The resulting objective function is obtained via substituting the delta function above with a degree of membership:

$$O(\mathbb{U}) = \sum_{\ell=1}^{k} \sum_{i=1}^{n} (u_{ij})^m ||\vec{x}_i - \vec{c}_\ell||^2 \tag{124}$$

where the matrix $\mathbb{U}$ is an $n \times k$ matrix with membership of the n-th element into the k-th cluster. Here m is the *fuzzyfication* parameter. Usually, one uses $m = 2$.
The definition of the center is like in k-means, weighted with membership:

$$\vec{c}_l = \frac{\sum_{i=1}^{n} (u_{il})^m \vec{x}_i}{\sum_{i=1}^{n} (u_{il})^m} \tag{125}$$

In this case, we cannot start with centers, but rather, we have a random initialization of $\mathbb{U}$. Once we have that, we can compute the centers. Then, we can update $\mathbb{U}$ as follows:

$$u_{ij} = \frac{1}{\sum_{\ell=1}^{k} \left[\frac{||\vec{x}_i - \vec{c}_\ell||}{||\vec{x}_j - \vec{c}_\ell||}\right]^{2/(m-1)}} \tag{126}$$

and repeat this until the change in $\mathbb{U}$ is below a given threshold. Convergence is typically element by element.
Unfortunately, fuzzy c-means has similar problems than k-means: initialization, number of clusters, outliers, spherical clusters.

10.3 Hierarchical methods

There are 2 types of hierarchical clustering whose result is equivalent:

- agglomerative: start with points as individual clusters, and at each point, merge the closest pair of clusters until only 1 or k clusters are left;
- divisive: start from a single cluster, and start making partitions until each cluster is left with a single point.

However, divisive clustering algorithms are very demanding, and not very easly applicable. We will thus focus on agglomerative cluster algorithms. Traditionally, one can use either a similarity or distance metric as a reference for partitioning.

Agglomerative cluster algorithm - The algorithm works as follows:

1. compute the distance matrix between input data points (each data point is considerated a separate cluster),
2. merge the two closest cluster and update the distance matrix between clusters;
3. repeat until a single cluster is left

All of this depends on the distance between clusters. It can be defined arbitrarly and is not as well defined as the one between points, as it is a distance between sets of points. There are several alternatives for how to estimate it[3].
The first hierarchical algorithm is based on **single-link distance**. This case consider the distance between two clusters as the minimum distance between any of their points. The challenge is that, at each joining step, one needs to perform several comparisons. This algorithm has some advantages: for instance, it can handle non-elliptical shapes. However, if in the presence of noisy data points, they can spoil the clusters, as it will generate chains. For similar reasons,

[3]Note that most times, these algorithms scale as N^3, that is not cheap.

it favors elongated clusters.
Another method is the **complete-link distance**. It takes as distance between the cluster the *maximum* between points within each cluster. Kind of the opposite with respect to single-link distances. It can generate nested clusters and is much less sensitive to noise. Clusters are typically more balanced. However, if the original clusters are not balanced, the algorithm enforces a fictitious balance.
These two methods emphasize the fact that hierarchical clustering typically returns cluster stuctures that are strongly dependent upon the choice of distance.
Another method is the **group average distance**. One just takes the average distance between group elements. This method is a bit more balanced than the previous ones and less susceptible to noise and outliers. However, it typically biases towards globular clusters. Yet another method is based on the **centroid distance**. where the distance is measured among centroids. This is not very different from the average distance procedure, with similar pros and cons.
Another method of choice is the **Ward's distance**:

$$D_w(C_i, C_j) = \sum_{x \in C_i} (x - r_i)^2 + \sum_{x \in C_j} (x - r_j)^2 - \sum_{x \in C_{ij}} (x - r_{ij})^2. \qquad (127)$$

Here C_i and C_j are two clusters and C_{ij} is the merging of the two; r_x is the centroid of cluster C_x. This is very close to k-means; it can be construed as its hierarchical analogue. Its properties, howeverm are similar to centroid distance methods.
In general, hierarchical cluster methods are very arbitrary in analysis real-world data. There is noy an immediate criterion to opt for one over another, and the choice have to be dictated from previous experience on similar problems.

11 Probability density

Till now we have considered two approaches for dealing with huge data set. The first focuses on performing a dimensional reduction from a D dimensional manifold to a d dimensional manifold ($D \ll d$), while the second maps the data points to a discrete set of clusters. In most cases a preliminary dimensional reduction approach is also needed to perform clustering, otherwise unfeasible due to the large amount of features of the data.

In both cases we do not have at our disposal a way of interpreting the new set of variables $\{y\}$ since the transformation exploited to derive them from $\{x\}$ is very involved. Instead of going through this path, one could think of studying the probability density of the embedding manifold M, which we already introduced as $\rho(x)$.

This would lead us to a result similar to clustering, however instead of describing the data points with a countable number of sets (clusters) we will identify *probability peaks* in the probability density distribution. The support of $\rho(x)$ is $M \subset \mathbb{R}^D$. Working with a smaller number of variables would be beneficial. Considering $\{y\}$ obtained after a dimensional reduction algorithm

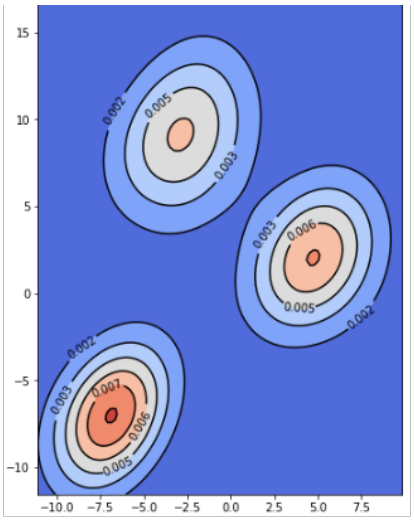

Figure 13: Example of probability density topography of a 2D dataset, after dimensional reduction.

(K-PCA, autoencoders), evidently $\rho(y)$ belongs to a subset in $\mathbb{R}^d$ and is easier to compute than $\rho(x)$. Then a necessary condition for evaluating the probability density is performing a *p*reliminary dimensional reduction. We will focus, from now on, on $\rho(y)$.
The way to describe $\rho(y)$ is through its topography, an example in Fig.13. We locate the probability peaks and the saddle points between the peaks; using them it is possible to fully characterize the probability landscape. Assuming to have n local maxima in $\rho(y)$, hence $n(n-1)/2$ possible saddle points, we construct a $n \times n$ symmetric matrix T_{ij} where on the diagonal we have the value of the probability density in the maxima and off diagonal the height of the saddle points between the peaks. Since between two peaks there could be several saddle points, we choose the one with the highest value in probability density, which likely would be the one which is less distant from both the peaks. According to this we have:

1. a huge dimensional reduction, beyond the preliminary one,
2. the possibility of interpreting the variables $\{y\}$ connecting the peaks and the data features,
3. the chance of introducing a further approximation, which would need a course on its own, that is in terms of *maximum probability path between two peaks* (minimum free energy for physicists).

11.1 Density estimation

In this section we will describe a few methods to estimate the probability density acting on the data. We will always assume to work on the variables $\{y\}$, hence after the preliminary dimensional reduction has been performed.

11.1.1 Histogram method

Let us work in one dimension to describe the *Histogram* method. Assuming $y \in [0,1]$, we divide the interval in n_{bins} bins. We denote with Δ the width of the bin and with y_i its center. The probability of observing a data point in bin i is evidently $P_i = \frac{n_i}{N} = \frac{\#\text{ of data points in bin } i}{\#\text{of data points}}$. Formally

$$\begin{aligned} P_i &= \int_{y_i-\frac{\Delta}{2}}^{y_i+\frac{\Delta}{2}} dy\rho(y) = \\ &= \int_{y_i-\frac{\Delta}{2}}^{y_i+\frac{\Delta}{2}} dy \left(\rho(y_i) + (y-y_i)\frac{d\rho}{dy}(y) + \frac{1}{2}(y-y_i)^2\frac{d^2\rho}{dy^2}(y) + \dots\right) = \\ &= \rho(y_i)\Delta + \frac{1}{24}\frac{d^2\rho}{dy^2}(y_i)\Delta^3 + O(\Delta^5). \end{aligned} \tag{128}$$

Therefore we can write an expression for the density $\rho(y)$:

$$\begin{aligned} \rho(y_i) &= \frac{P_i}{\Delta} - \frac{1}{24}\frac{d^2\rho}{dy^2}(y_i)\Delta^2 \\ &= \frac{n_i}{N\Delta} - \frac{1}{24}\frac{d^2\rho}{dy^2}(y_i)\Delta^2 \end{aligned} \tag{129}$$

The second term in the right-hand side is called *bias*, which is the systematic error we commit evaluating $\rho(y)$ using $P_i = n_i/N$ as probability estimator. Apart form the bias it is necessary to consider also the statistical error of the estimator which can be computed from its variance. The number of points in a bin i is distributed according to a binary distribution:

$$Bin(P_i, N) = \binom{N}{n+i} P_i^{n_i}(1-P_i)^{N-n_i}, \tag{130}$$

the square root of its variance returns the estimation of the statistical error:

$$\sqrt{E\left(\left(\frac{n_i}{N} - P_i\right)^2\right)} = \sqrt{\frac{P_i}{N}} = \sqrt{\frac{n_i}{N^2}}. \tag{131}$$

Finally the relative error on P_i is $\epsilon = \frac{1}{\sqrt{n_i}}$. We can individuate two regimes:

- Large $\Delta \to$ small ϵ, large bias.
- Small $\Delta \to$ large ϵ, small bias.

The optimal working point must consider the trade-off between the two limits. This method we described is not useful for $d \geq 3$ since the number of bins to be used scales with the dimensionality of the dataset, thus the number of points N needed.

11.1.2 Kernel methods

Formally reshaping the histogram estimator of the probability we can write:

$$P_i = \frac{n_i}{N} = \frac{1}{N} \sum_{j=1}^{N} \chi_{[y_i - \frac{\Delta}{2}, y_i + \frac{\Delta}{2}]}(y_j), \tag{132}$$

where $\chi_{[a,b]}(y)$ is 1 is $y \in [a, b]$, zero otherwise. This allows to extend $\rho(y_i)$ to $\rho(y)$, from the data-points to the data-manifold.
Let us introduce a kernel $K(y, \tilde{y})$:

1. $K(y, \tilde{y}) \geq 0$, $\forall y, \tilde{y}$,
2. $K(y, \tilde{y}) = K(\tilde{y}, y)$,
3. $\int dy K(y, \tilde{y}) = 1$, $\forall \tilde{y}$.

This allows us to define an estimator of the probability density which we call $\rho_{\Delta,N}(y)$:

$$\rho_{\Delta,N}(y) = \frac{1}{N} \sum_{j=1}^{N} K(y, y_j). \tag{133}$$

Considering $K(y, y_j) = \frac{1}{\Delta} \chi_{[y - \frac{\Delta}{2}, y + \frac{\Delta}{2}]}(y_j)$ this approach reduces to the Histogram method. Rigorously speaking $\rho_{\Delta,N}(y)$ is not an estimator of $\rho(y)$. In fact

$$\lim_{N \to \infty} \rho_{\Delta,N}(y) = \int dy' \rho(y') K(y, y'), \tag{134}$$

and $\rho_{\Delta,N}(y)$ happens to be an unbiased estimator of the convolution of the probability density with the kernel. Their difference is what we called bias. One could prove, also in this case, that the bias scales as $\Delta^2 \frac{d^2\rho}{dy^2}$ and that the realtive error depends on the number of points within a distance Δ from the center of the kernel. What it is gained using this approach is that the probability density estimator is defined as an analitic function in each point of the reduced manifold. An example of kernel is the Guassian Kernel:

$$K(y^i, y^j) = \frac{\exp\left[-\frac{(y^i - y^j)^2}{2\Delta^2}\right]}{\sqrt{2\pi\Delta^2}}. \tag{135}$$

When the data are not uniformly distributed in the manifold the Gaussian kernels fails to estimate the density distribution properly because the same Δ is not efficient in each point of the space.

11.2 K-NN estimator

It would be smart, when the probability density is not uniform in $\{y\}$, to adapt Δ so that the density in the region where the estimation of $\rho(y)$ is performed can be considered uniform, approximately. The simplest way to consider an adaptive Δ is the K-NN estimator. It works in the same manner as the 2-NN method already described:

1. Choose an integer k, external parameter,
2. Find the k-th nearest neighbour to point i,
3. Calculate the distance between the k-th nearest neighbour and i: $r^i_{(k)}$,
4. Evaluate $\rho(y_i)$ as $\rho(y_i) = \frac{k}{\Omega_d \left(r^i_{(k)}\right)^d}$.

With this procedure the number of points considered to evaluate $\rho(y)$ within each region is always the same despite $\rho(y)$ being uniform or not on the whole manifold. The computation can be done calculating the distances directly on the manifold M, since the first assumption in each dimensional reduction method is that the distances between near points should not change:

$$||x_i - x_j|| = ||y_i - y_j||, \quad \text{if they are close enough.} \tag{136}$$

However d must be estimated in advance because using D one would find $\rho(y_i) = \frac{k}{\Omega_D \left(r^i_{(k)}\right)^D}$ which approaches a very small number due to the huge dimension of the space ($D \sim 1000$). The quantity $\rho(y_i) = \frac{k}{\Omega_d \left(r^i_{(k)}\right)^d}$ is instead well defined ($d \sim 10$).
The reason why $K-NN$ estimates the density as we showed is easly explained in the same terms of the 2-NN method. Let us consider a point in the data manifold and its neighbours, one could refer to Fig.7. We defined v_l the hyperspherical shell between the $l-th$ and $(l-1)-th$ neighbour. The v_l's are statistically independent and their probability distribution is an exponential $P(v_l|\rho) = \rho \exp\{-\rho v_l\}$, assuming ρ constant in the neighborhood. Thus

$$P(\{v_1, \dots, v_k\}|\rho) = \prod_{l=1}^{k} P(v_l|\rho) = \rho^k \exp\left\{-\rho \sum_{l=1}^{k} v_l\right\} = \rho^k \exp\{-\rho V k\}, \tag{137}$$

where $V_k = \Omega_d \left(r^i_{(k)}\right)^d$. Using Bayes' theorem (under the appropriate conditions?) one derives

$$P(\rho|\{v_1, \dots, v_k\}) = C\rho^k \exp\{-\rho V_k\}, \tag{138}$$

with C normalization constant. Using maximum likelihood one infers the best value of ρ maximizing the probability $P(\rho|\{v_1, \dots, v_k\})$. With straightforward algebra we find

$$\frac{\partial \mathcal{L}}{\partial \rho} = \frac{\partial \log P(\rho|\{v_1, \dots, v_k\})}{\partial \rho} \to \rho = \frac{k}{V_k}, \tag{139}$$

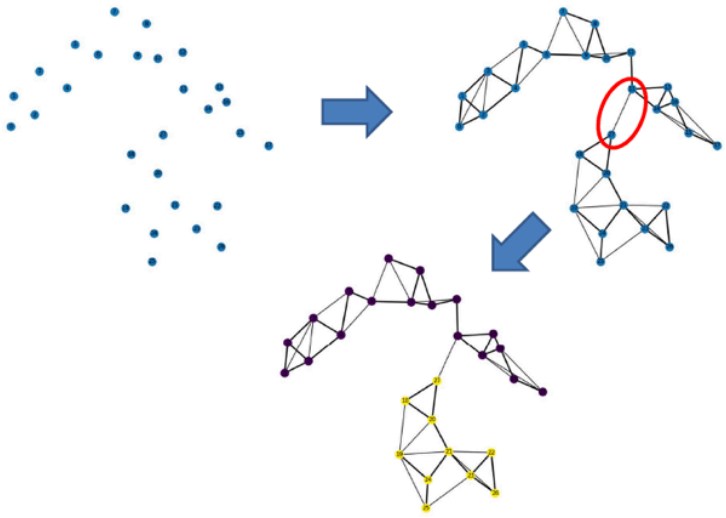

Figure 14: Sketch spectral clustering. Associating a graph to a set of points, then recognizing different clusters looking for cuts with small weight (using in Eq.140).

which is the K-NN estimator.

12 Lecture 12: Modern clustering algorithms

So far, we have mostly characterized data samples and feature selection and ranking in great detail. The methods we have reviewed have several limitations: k-means and fuzzy c-means are not well suited to deal with arbitrary shaped objects, and hierarchical needs a proper definition of intercluster distance. Moreover, there is generally sensitivity to noise.
The first example of modern methods is kernel k-means. There are of course hundreds of clustering methods, each of them with advantages and disadvantages. [[see reviews]]

12.1 Spectral clustering

We now discuss a different paradigm with respect to the previous ones that is dubbed spectral clustering. The basic idea is to pass through graph theory, instead of working in the full space, and then identifying links that can be cut (Fig.14 as a reference)
When trying to associate a graph to a data set, one typically employs undirected weighted graphs. These are characterized by a set of vertices $G(V, E)$, and links $E\{((i, j), S_{ij})\}$, with similarities $S_{ij} \geq 0$. These are also called *similarity graphs*. The graph however can be obtained in different manners (fig. 15):

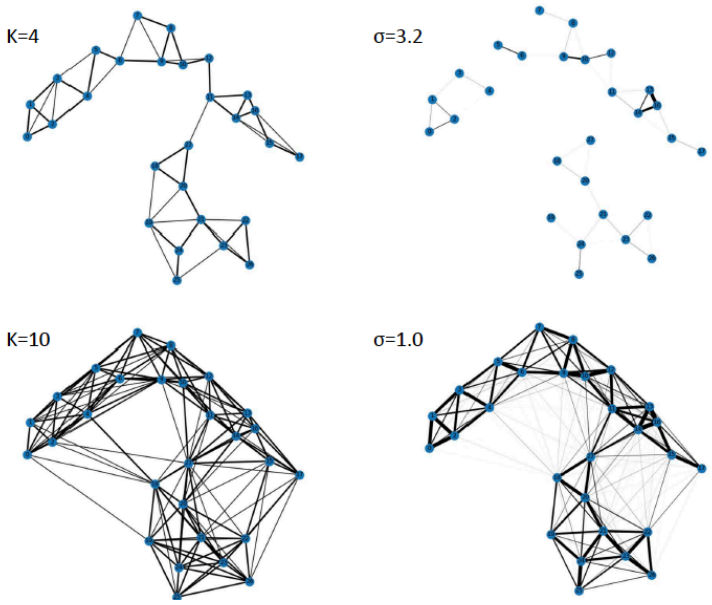

Figure 15: Sketch of different graphs. Left column: k-NN graphs for k=4,10. Right column: fully connected graph for $\sigma = 1.0, 3.2$.

1. ϵ-ball graph, where we join vertices within ϵ
2. k-NN graph, where we join k nearest neighbours,
3. fully connected graph with $S_{ij} = \exp\left[-d_{ij}^2/2\sigma^2\right]$. Graphically one associate a thicker edge to points whose similarity S_{ij} is larger and no edge if $S_{ij} = 0$.

The main challenge of the algorithm is to recognize which is the link that separates two different clusters, i.e. how to cut the graph. The cut can be expressed as:

$$\mathrm{Cut}(C, C') = \sum_{i \in C} \sum_{j \in C'} S_{ij} \tag{140}$$

This is essentially counting the similarities of the links that are cut. This formula holds for cutting into 2 parts. For many, one has

$$\mathrm{Cut}(C_1, C_2, ...C_k) = \frac{1}{2} \sum_{\ell} \mathrm{Cut}(C_\ell, \bar{C}_\ell) \tag{141}$$

where we have defined $\bar{C}_\ell$ as the complement of C_ℓ (that is, where we have cut all links not cut in C_ℓ). This way, we are cutting edges of low weight. Minimizing the cut function would give the information we are interested in, i.e. where to separate the clusters of the grap. However, in real-world applications, outliers,

isolated or border points, are often recognized as separate clusters because their similarity with most of the other points is very small. A way to resolve this problem, before delving into the algorithm itself, is to weighten the cut function properly.
First of all, we take into consideration the weight of each cluster by counting the number of edges $|C_\ell|$. We can then utilizing the ratio cut criterion:

$$\text{RatioCut}(C_1, ...C_k) = \sum_{\ell=1}^{k} \frac{\text{cut}(C_\ell, \bar{C}_\ell)}{|C_\ell|} \tag{142}$$

Alternatively, we can use the volume of a cluster. For that, we need to define the degree of a given vertex:

$$d_i = \sum_j S_{ij} \tag{143}$$

and the volume then reads:

$$\text{Vol}(C_\ell) = \sum_i d_i \tag{144}$$

so that the volume normalized cut reads:

$$\text{NCut}(C_1, ...C_k) = \sum_{\ell=1}^{k} \frac{\text{cut}(C_\ell, \bar{C}_\ell)}{\text{Vol}(C_\ell|)} \tag{145}$$

Given the ratio-cut function, one can try to minimize it. However it is NP hard problem, hence we will discuss smart ways to deal with it in the following.

12.1.1 Spectral clustering

This can be addressed via spectral clustering. We need first to introduce a few ingredients:

1. the degree $\mathbb{D}$-matrix, that satisfies:

$$D_{ii} = d_i, \quad D_{ij} = 0 \tag{146}$$

2. the laplacian matrix $\mathbb{L} = \mathbb{D} - \mathbb{S}$, where the second is the weight matrix whose elements are the similarities. This matrix is still symmetric, positive semidefinite, and the smallest eigenvalue is 0.

One can then find that the laplacian satisfies:

$$\begin{aligned} v^T \mathbb{L} v = v^T \mathbb{D} v - v^T \mathbb{S} v = \\ = \sum_{j=1}^{n} d_j v_j^2 - \sum_{i,j} v_i v_j S_{ij} = \frac{1}{2} \sum_{i,j} S_{ij} (v_i - v_j)^2 \end{aligned} \tag{147}$$

Let us try to use this laplacian to minimize the ratio cut for $k = 2$. We define a vector f whose components satisfy $f_i = \sqrt{\frac{|\bar{C}|}{|C|}}$ if $v_i \in C$, and $f_i = -\sqrt{\frac{|\bar{C}|}{|C|}}$

otherwise (that is, $v_i \in \bar{C}$).
Now, we can get:

$$\begin{aligned} f^T \mathbb{L} f &= \frac{1}{2} \sum_{i,j} S_{ij} (f_i - f_j)^2 = \\ &= \frac{1}{2} \sum_{i \in C, j \in C} S_{ij} \left(\sqrt{\frac{|\bar{C}|}{|C|}} + \sqrt{\frac{|\bar{C}|}{|C|}} \right)^2 + \\ &+ \frac{1}{2} \sum_{i \in \bar{C}, j \in C} S_{ij} \left(-\sqrt{\frac{|\bar{C}|}{|C|}} - \sqrt{\frac{|\bar{C}|}{|C|}} \right)^2 \\ &= \mathrm{nRatioCut}(C, \bar{C}) \end{aligned} \tag{148}$$

we have now expressed our loss function as a matrix/vector product. We need to minimize $f^T \mathbb{L} f$. Along minimization, we need to satisfy the fact that f is orthogonal to the unitary vector, and satisfies $||f||^2 = n$. This is still an NP-hard problem one is dealing with, but we can use the tools we introduced to reformulate it in a different way.
First let us generalize the problem to arbitrary value of k. Instead of working with a vector, we will be working with a matrix:

$$H_{ij} = \frac{1}{\sqrt{|C_i|}} \text{ if } i \in C_j \text{ else } 0 \tag{149}$$

with $i = 1, ..., n; j = 1, ..., k$, so the matrix is rectangular. Then:

$$H^T H = 1 \tag{150}$$

by definition of the matrix. If we then define h_i as the column vector,

$$h_i = H_{,i} \tag{151}$$

Then, we have:

$$h_i^T \mathbb{L} h_i = \frac{\mathrm{Cut}(C_i, \bar{C}_i)}{|C_i|} = (\mathbb{H}^T \mathbb{L} \mathbb{H})_{ii} \tag{152}$$

so that one gets:

$$\mathrm{RatioCut}(C_1, ...C_k) = \mathrm{Tr} \mathbb{H}^T \mathbb{L} \mathbb{H} \tag{153}$$

under the condition that $\mathbb{H}^T \mathbb{H} = 1$. The challenge is that $\mathbb{H}$ is discrete valued, so stays NP-hard. If we instead solve it for continuous values, it is tractable with lagrange multipliers. One then obtains k eigenvectors, and then applies k-means using those as coordinates - but is not guaranteed that this solution is similar to the unrelaxed one. This procedure is called relaxation, and is very widely used.
We now summarize it:

1. construct the similarity graph,

2. compute the laplacian,

3. compute the first k eigenvectors,

4. use k-means,

Unfortunately, this technique does not consider intracluster similarities. This can be achieved by using NCut instead of RatioCut. In this case, the indicator metric is defined as:

$$\mathbb{H}_{ii} = \frac{1}{\sqrt{\text{Vol}(C_j)}} \quad \text{if} \quad i \in C_j \tag{154}$$

and 0 otherwise. The traces do not change, but what changes is the condition on H, that is:

$$\mathbb{H}^T \mathbb{D} \mathbb{H} = 1 \tag{155}$$

In order to do this, one can define an auxiliary matrix $\mathbb{T} = D^{1/2}\mathbb{H}$, and then minimize the following:

$$\min(\mathbb{T} D^{1/2} \mathbb{H} D^{1/2} \mathbb{H} T) \tag{156}$$

which is like replacing the laplacian in a symmetrized way, $D^{1/2}\mathbb{H}D^{1/2}$.
There is a close relation between spectral clustering and k-means. In fact, they optimize the same wave functions under specific conditions. Moreover, playing a different trasformation to the laplacian, it is possible to map the problem to a random walk matrix, that can be analyzed via Markov state model analysis.

Problems - The main issues of spectral clustering are:

- how to construct the similarity matrix? this is arbitrary of course, as it depends on method and parameters and the algorithm is quite sensitive to this;
- deciding the number of clusters is unclear if we do not have a gap,
- computationally expensive,
- noise sensitivity: single outlier points can spoil the graph.

12.2 Expectation-maximization clustering methods

This method enters in the framework of *model based algorithms*. First, one considers the data as a set of realization of an underlying probability function $p(x)$. Then, one assumes the functional form of $p(x)$, and then estimate its parameters via maximum likelihood.
The Expectation-maximization algorithm is based on optimizing the parameters of this last step which cannot be done analytically. It is an iterative procedure to compute the maximum likelihood estimate even in the presence of missing or incomplete data.

We first have to define a mixture of k-Gaussians

$$p(x) = \sum_{k-1}^{K} \pi_k F(x|\theta_k), \quad \sum_k \pi_k = 1, \tag{157}$$

with mean and variance $\theta_k = (\mu_k, \sigma_k)$. Then $P(X)$ or its logarithm must be minimized to evaluate the weights π_k and then the parameters θ_k. Here we write $P(X)$:

$$P(X) = \prod_{i=1}^{n} \sum_{k-1}^{K} \pi_k F(x|\theta_k). \tag{158}$$

At each step, every point x_i is probabilistically assigned to the k-th Gaussian, with probability:

$$w_{ik} = \frac{\pi_k F(x_i, \theta_k)}{\sum_j \pi_j F(x_i, \theta_j)} \tag{159}$$

so that every Gaussian as an effective number of points associated to it, $N_k = \sum_{i=1}^{n} w_{ik}$. One can then compute the mean and the variance of each Gaussian:

$$\begin{cases} \mu_k = \frac{1}{N_k} \sum_i w_{ik} x_i, \\ \sigma_k = \frac{1}{N_k} \sum_i w_{ik} (x_i - \mu_k) x_i (x_i - \mu_k)^T. \end{cases} \tag{160}$$

k-means with a Gaussian kernel is very similar to this, apart from the fact that the process here is probabilistic. The present method has similar issues: if the model is not realistic, the method is compromised. Moreover, there is no guarantee to arrive at a global minimum. The main advantage is that the k can be inferred with bayesian model selection (this is called Dirichlet process). Still, the process can be very expensive at the computational level.

Expectation-maximization algorithm Here we write explicitly the steps of the algorithm:

1. initialize means μ_k, variances σ_k and coefficients π_k, and evaluate the initial value of $P(X)$;
2. Evaluate the weigths $w_{ik} = \frac{\pi_k F(x_i, \theta_k)}{\sum_j \pi_j F(x_i, \theta_j)}$;
3. maximize $P(X)$ (or its logarithm) and evaluate means μ_k, variances σ_k and coefficients π_k;
4. continue till convergence of $P(X)$.

13 Thirteenth lecture: theory and applications of clustering algorithms

Untill now we have overviewed some *classical* methods

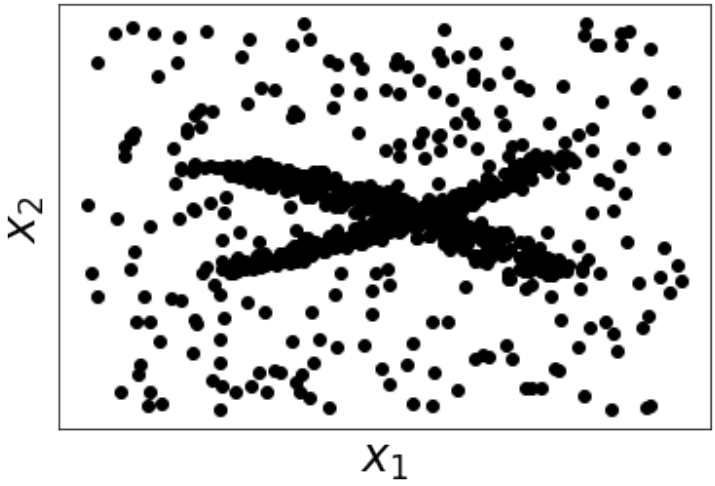

Figure 16: Example of data set in which Expectation-maximimation clustering method encounters difficulties.

- k-means clustering (hard partitions)
- c-means clustering (fuzzy clustering)
- hierchical clustering methods

and then covered:

- kernel k-means;
- spectral clustering;

which are more advanced and solve some problems related to the first ones. Basically, in the last two cases, one gets the data and transform them (using a kernel or creating a graph) and then performs K-means in the new space.
The last algorithm we considered is the Expectation-maximization alghoritm (EM) that consider data as realization of a given probability, assume a functional form for the probability itself and then evaluate its parameters using maximum likelihood.
In this section we want to discuss about the problems of EM. Imagine you are working with something like in Fig. 16. One would need two gaussians to capture the cluster in the middle which has a cross shape. Instead, the algorithm will try to build two clusters one for each branch of the cross, and will get confused in assigning points in the center of the cross where the two overlaps. This illustrates 3 issues:

1. choice of the model: an initial model where one uses crosses would solve this;

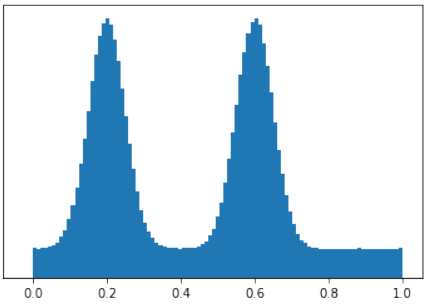

Figure 17: Example of probability density with two well defined peaks.

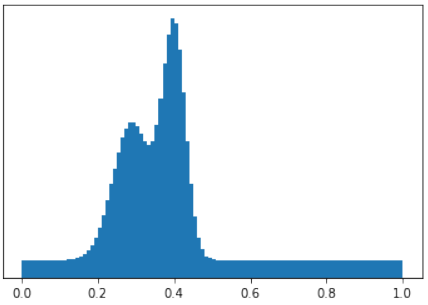

Figure 18: Example of probability density where peaks almost overlap.

2. choice of k; here, fixing $k = 2$ is fundamental

3. assignation of the points to the clusters.

Ultimately, the last two problems actually depend on the first one in the example above. However, for generic data, one does not know. In order to overcome this problem one has to introduce *non-parametric* density based clustering methods.

13.1 Non-parametric density based clustering

EM is a density based parametric clustering method, because it relies on the estimation of the parameters of the probability distribution which is fed as an hypothesis. Non-parametric methods allow to overcome some of its limitations. To discuss these algorithms, let us compare two definitions of cluster:

1. regions of high density divided by regions of low density (noise). If I have a region of high density, it is typically easy to identify (Fig.17), but for instance (see Fig.18), when the introduction of a threshold is needed, the definition of clusters will strongly depend on that.

2. peaks (modes) of the probability density distribution. The issues here arise if one has flat regions of high density.

We will now review two methods that rely on these definitions.

13.1.1 DBSCAN - density-based spatial clustering of applications with noise

DBSCAN locates regions of high density which are separated by regions of low density, according to the first definition aboce. We calculate the density as the number of points within a given region. Then, a point is defined as **core point** if it has more than a specified number of points $N_{\min}$ within a given radius ε. A **border point** has less points than $N_{\min}$, but has a core point in its neighborhood ε. A **noise point** is not core and has no core point within the

radius.
Then, any pairs of core points close enough (within a distance ε) are put in the same cluster. Also border points which are distant not more than ε to core points are attached to cluster. And the rest are discarded. Following this, one can introduce the concept of **ϵ-neighborhood**. This is just the set of points which lie in a radius ϵ from a given object. The **core objects** are the ϵ-neighborhood of an object that contains at least $N_{\min}$ points.
The reachability is also an important concept we have to introduce. An object q is **directly density-reachable** from p if it is within the ϵ-neighborhood of p, and p is a core object. An object q is **density-reachable** from p with respect to a distance ε if there is a chain of objects $p_1, p_2, \ldots, p_n$ such that

1. $p_1 = p$,
2. $p_n = q$,
3. p_{i+1} is directly density-reachable from p_i.

The last must hold given the definition of directly density-reachable, at fixed $N_{\min}$ and ε. An object p is instead **density-connected** to another q if there is an object o so that both p and q are density reachable. For instance, two border points can be density connected, but are not density reachable. In DBSCAN, we define a cluster C as a set of objects D that, with respect to ε and $N_{\min}$, satisfy:

1. *Maximality* - for each p, q if $p \in C$ and q is density-reachable from p then also $q \in C$.
2. *Connectivity* - for all $p, q \in C$ p is density-connected to q.

Note that with these definitions cluster may contain core objects and as well as border objects. Instead, a Noise object is the set of objects that are not directly density-reachable from at least one core object.

The algorithm works as follows, once ϵ and$N_{\min}$ are fixedL

1. select a point p,
2. retrieve all points that are density reachable from p,
3. if p is core, a cluster is formed,
4. if p is border, DBSCAN visits the next point in the database,
5. continue until all points have been processed.

The result is independent of the order of the processing. All of this is like defining a threshold in the density profile, fixed by the initial parameters. Note that this can have problems if the clusters have different dimensions, because in that case the density is very hard to compare.

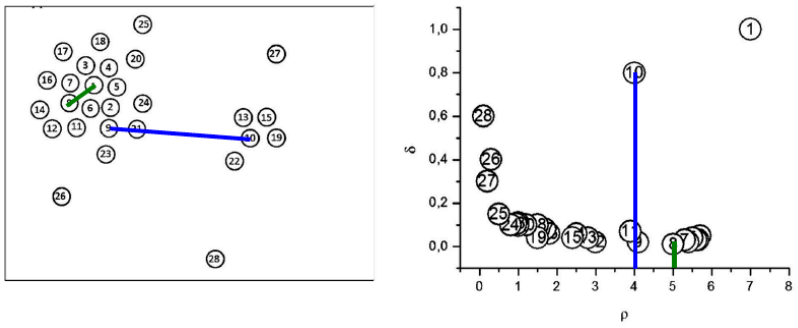

Figure 19: Left: plot of the data set. Right: minimum distance δ from points with higher density, for each point, as a function of the their density.

13.2 Fast search and find of density peaks

This method is developed following the second definition of clusters. It is thus based on the fact that each cluster corresponds to a peak in the density of points. A similar idea was already present in a method called "mean-shift", that had already issues related to coordinate definitions: the present method is solely relying on distances.
The main assumption here is that clusters correspond to peaks in the 'mother' probability distribution.
A key concept to introduce is δ. Imaging that you have two clusters in 2D space.

- One can progressively compute the density of the various points by counting the number of neighbors within a radius, that is fixed.
- for each point, one can compute the distance from all the points with higher density. Then, one takes the minimum value. This is called δ.
- this quantity allows us to define a cluster center (see Fig.19). Clust centers have large density, and large δ, and are outliers in the distribution.
- Once cluster centers are defined, it is easy to assign each point to the same cluster of its nearest neighbor of high density.

The corresponding algorithm works as follows:

- given the distance matrix d_{ij}, compute the density,
- compute the δ_i, with cut-off d_c,
- plot the decision graph, and identify the outliers,
- this fixes the number of clusters.

the method is only sensitive to relative densities, so the cut-off parameter needed in the second step is not very problematic.

There are still some points that we are not easly defined: those points are called **halo points**. One can specify them by defining a **border**: a point i of the cluster A is a border point if it has, within distance d_c, a point j belonging to another cluster. This definition is strongly dependent on parameters. One can add in addition that j does not have any point of A in its neighbor to strengthen the constraint.

13.3 Beyond density peaks

Density peak clustering has some issues:

- density estimation: this is not easy, as defining the proper parameters is challenging
- decision graph: those are complex in high dimensional data. Since supervision is needed, there is a degree of arbitraryness;
- hierarchical data sets lead to extra problems in the identification of clusters.

following a bit confusing We can still try to carry the procedure several times and utilize likelihood ratio test. This allows one to compare the distributions we have obtained, and can also tell us if the two models define the same distribution, or a different one.
This requires us to define a threshold to identify the clusters. Note that the input is not the total number of clusters, but rather, the error that we allow. Then, we can compute the density and its error with a linear correction. This can be done starting from an ansatz, and then maximized numerically.
This procedure is unbiased and returns a reasonable error estimate.

13.3.1 Beyond density peaks: DPA

We now have a good density estimator: can we use it to improve the density peak search? To do so we introduce a method called DPA. Let us start from some basic definitions.
A point is a putative center of a cluster if its density is a maximum within k^*, that is, among its optimal k nearest neighbors.
Two clusters shall be merged if the density at the border is compatible with the density at the maximum, within a given threshold. Note that this error is only due to the new density estimator. With this method, we get both the number of clusters, and the "barrier" between those. We can chart the fully topography of our data, so to constuct in principle a hierarchical structure. There is a related method, called hierarchical DBSCAN, that does something similar.

14 Fourteenth lecture: correlated data - exploiting kinetic information

In the last two lectures, we explore two different classes of strategies for unsupervised learning dimensional reduction. These 2 strategies can only be applied to specific categories of data.

So far, our interest was in data sets described by X^i, implicitly assumed that these data points where harvested independently from a probability density $P(x)$. Instead, now we will focus on cases where X^t are labelled by a "time", and those data come with a **order**. This property is very common,examples include

- dynamic systems:
$$X^{t+1} = f(X^t) \tag{161}$$
where the function can be either deterministic or stochastic f_ξ;
- text: the meaning comes from a specific ordering of the words;
- genome: again, order dictates specific features.

The second and third examples have both ordering, and direction.
In the methods we discuss now, we will exploit time ordering to infer the structure of the data via unsupervised learning.
Before describing the approaches, it is worth stressing that this time-ordering typically implies correlation between the data:

$$E(O(x^t)O(x^{t+1})) \neq E(O(x^t))E(O(x^{t+1})) \tag{162}$$

so this means that those are not samples harvested independently from a probability distribution. Note that time-correlations are typically considered a confounder for all tasks we have seen so far, from computing the I_d to train autoencoders. These correlations induce unwanted, hard to control errors. We will now exploit such time correlations to perform dimensional reduction.

14.1 What does dimensional reduction mean in a dynamic system?

The first thing that it means is that the dynamics takes place in a manifold $\mathcal{M}$ of dimension $d \ll D$, i.e.

$$\mathbb{R}^D \to \mathbb{R}^d \tag{163}$$

as is the case also in the absence of memory.
The second thing is related to the presence of metastable states: the latter are attractors, and one typically has m of them. An attractor is a stable set of this dynamics: if I iterate the map, I get stuck there. For instance, it can satisfy:

$$x = f_\xi(x) \tag{164}$$

Those can be both stable and unstable attractors (minima and saddle points). Typically, if ξ is finite, there are no attractors (it is only true in practice; there are processes where this is not true).

The holy grail of dimensional reduction is to determine a dynamic (stochastic) process in the space of the attractors. If I have m attractors, the dynamics is fully determined by the probability p_β of ending up in the β-attractor. In particular, I will have:

$$p_\beta^{t+1} = g(p_{\vec{\beta}}^t) \tag{165}$$

where $\vec{\beta}$ is a collection of all probabilities of the various attractors. In many applications of enormous relevance, g is a linear function. This fact is remarkable. The corresponding rate equation/master equation is:

$$p_\beta^{t+1} = \sum_{\alpha=1}^{M} K_{\alpha\beta} p_\alpha^t \tag{166}$$

How this equation can be generated by a stochastic process is non-trivial. It is a very important dimensional reduction as it describes not only the 'statistics' of the system, but also its kinetics, and provides a full characterization of the problem.

14.2 Markov state modelling

Markov chain modelling is a technique that allows to perform dimensional reduction on systems described a Markov process (where to determining X^{t+1}, one needs to know only X^t). We exclude here processes including annihilation and creation of data (effects such as duplication, etc.): for instance, deep neural networks belong to this class, as they can be seen as stochastic processes, but their activations map into spaces between different dimensions (think about feed-forward NN).
The starting point of this modelling is that we interpret the Markov chain as a mapping between probability densities:

$$P(X^{t+1}) \leftarrow P(X^t) \tag{167}$$

] on the same space. This transformation takes place via conditional probabilities:

$$P(X) = \int dy P(Y) P(X|Y). \tag{168}$$

Since my stochastic process is mapping my state between two configurations within the same space, I can associate to the dynamical process $X^{t+1} = f_\xi(X^t)$ an equation in the space of probability densities:

$$P(X^{t+1}) = \int dX^t P(X^{t+1}|x^t) P(X^t) \tag{169}$$

where the conditional probability is defined by f_ξ. This is sometimes referred to as Green's function. In many cases, such as molecular dynamics and Monte Carlo, this function is known. In general, it is not: we will concentrate on the latter, and we will aim at estimating $P(X^{t+1}|x^t)$.
Since it is not possible to estimate a conditional probability in continuous space via unsupervised learning, in general, one has to perform a preliminary dimensional reduction to a discrete space, that one does via clustering. Given X^t, one can exploit methods such as k-means, and produce a meaningful basis to estimate the conditional probability $P(X^{t+1}|x^t)$. The partitioning shall be sufficiently large, that is, the corresponding discrete set of state shall be large: $L \gg 1$.
We now rewrite Eq. (169) in discrete space, since we want to map the probabilities:

$$P(x) \to P_\beta, \quad \beta = 1, ..., L \tag{170}$$

where be mindful that the β is defined not on attractors, but on microstates. We then denote $P(X|X') = \Pi_{\alpha\beta}$, and get:

$$p_\beta^{t+1} = \sum_{\alpha=1}^{L} \Pi_{\beta\alpha} p_\alpha^t \tag{171}$$

Again, we remind that $\Pi_{\alpha\beta}$ is the probability of observing the state β at time $t+1$, given that it was in state α at time t:

$$\Pi_{\alpha\beta} = P(\beta, t+1|\alpha, t) \tag{172}$$

Note that in principle, one can just discretize space and work directly on a lattice: however, this is typically very intractable, and clustering provides a smart and efficient way to discretize.
The transition between Eq. (169) and Eq. (171) is not exact at finite L. We will come back to this approximation later.
Let us discuss two properties of $\Pi_{\alpha\beta}$:

- $\Pi_{\alpha\beta} \geq 0$ for all α, β
- $\sum_\alpha \Pi_{\alpha\beta} = 1$.

Those two qualify $\Pi_{\alpha\beta}$ as a *stochastic matrix*. For our purposes, few properties are important:

1. if p_β^t is a probability, then also p_β^{t+1} is a probability. This follows from the fact that, if $\sum_\beta p_\beta^t = 1$, then

$$\sum_\alpha p_\alpha^{t+1} = \sum_{\beta,\alpha} \Pi_{\alpha\beta} p_\beta^t = \sum_\beta p_\beta^t = 1 \tag{173}$$

2. the matrix Π has at least an eigenvalue equal to 1. This follows from:

$$\sum_\alpha \Pi_{\alpha\beta} \mathbb{I}_\alpha = \mathbb{I}_\beta \tag{174}$$

where $\mathbb{I}_\beta$ is a row vector with elements 1. This implies that $\mathbb{I}$ is a left eigenvector of Π with left eigenvalue 1.

3. the left eigenvalues and the right ones of a square matrix are the same. This implies that $\lambda = 1$ is also a right eigenvalue, with eigenvector p^{eq}_α, satisfying:

$$\sum_\beta \Pi_{\alpha\beta} p^{eq}_\beta = p^{eq}_\alpha \tag{175}$$

that implies that this is a stationary solution of our evolution equation Eq. (171).

A few questions may pop up: is p^{eq} the only stationary solution of Eq. (171); is p^{eq} a stable solution, i.e. if I perturb my system, do I observe a dynamics that brings me back to p^{eq}? We will prove those with two theorems.

Gersh-Gorin theorem. - [[last part requires clarifications]] All the eigenvalues of a stochastic matrix satisfy $|\lambda| \leq 1$ (note that the matrix is not symmetric, so the eigenvalues can be complex). This theorem is easy to prove starting from the left eigenvalue equation:

$$\lambda^\gamma u^\gamma_\beta = \sum_\beta \Pi_{\beta\alpha} u^\gamma_\beta \tag{176}$$

with left eigenvalues u^γ. If one takes the modulus of both sides, one get:

$$|\lambda^\gamma u^\gamma_\beta| = |\sum_\beta \Pi_{\beta\alpha} u^\gamma_\beta| \tag{177}$$

that, using the triangular inequality, leads to:

$$|\lambda^\gamma||u^\gamma_\beta| \leq \sum_\beta \Pi_{\beta\alpha} |u^\gamma_\beta| \tag{178}$$

and then

$$|\lambda^\gamma||u^\gamma_\beta| \leq \max_\beta |u^\gamma_\beta| \sum_\beta \Pi_{\beta\alpha} = \max_\beta |u^\gamma_\beta| \tag{179}$$

that implies:

$$|\lambda^\gamma| \leq \frac{\max_\beta |u^\gamma_\beta|}{|u^\gamma_\alpha|} \tag{180}$$

This must be true for all α, thus also for α such that $|u^\gamma_\alpha| = \max_\beta |u^\gamma_\beta|$. We call this $\alpha = \hat{\beta}$. So that one has:

$$|\lambda^\gamma| \leq \frac{\max_\beta |u^\gamma_\beta|}{|u^\gamma_{\hat{\beta}}|} = 1 \tag{181}$$

this will ensure that p^{eq}, if unique, will be a stable solution.

Perron-Frobenius theorem. - This theorem allows to tell under which condition the eigenvalue $\lambda = 1$ is unique, and thus, if the right eigenvector p^{eq} is unique. This will happen if:

1. all entries are strictly positive, $\Pi_{\alpha\beta} > 0$;
2. irreducibility holds: if $\Pi_{\alpha\beta}$ has 2 eigenvalues equal to 1, there exists a permutation of column and row indeces to bring $\Pi_{\alpha\beta}$ in a block diagonal form.

15 Fifthteen lecture

Last lecture we have considered ordered data. We have seen that the problem can be cast in the form of a master equation, that is related to the eigenvalue equation for the matrix Π. Let us recally the right eigenvalue equation:

$$\sum_{\beta} \Pi_{\alpha\beta} v_{\beta}^{\gamma} = \lambda^{\gamma} v_{\alpha}^{\gamma} \tag{182}$$

where v^{γ} are nothing but the right eigenvalues, labelled by γ. Similarly, for the left eigenvalues:

$$\sum_{\beta} \Pi_{\beta\alpha} u_{\beta}^{\gamma} = \lambda^{\gamma} u_{\alpha}^{\gamma}. \tag{183}$$

For square matrices, the left and right eigenvalues are the same. Moreover, the eigenvalues are mutually orthogonal:

$$\sum_{\alpha} u_{\alpha}^{\gamma} v_{\alpha}^{\gamma'} = \delta^{\gamma,\gamma'}. \tag{184}$$

We already demonstrated that:

1. $|\lambda^{\gamma}| \leq 1 \quad \forall \gamma$
2. at least one eigenvalue, that we call λ^0, is equal to 1. The left eigenvector associated to λ^0 is a vector $u^0 = (1, 1, 1, 1, \dots)$. The right eigenvector we denote as $v_{\alpha}^0 = p^{eq}$, and satisfies the property Eq.(175).
3. We will also assume that λ^0 is unique. It is not true in general but there are 2 sufficient conditions that make it possible
   - $\Pi_{\alpha\beta} > 0$, i.e. no zero entries in the matrix,
   - Π cannot be brought in a block diagonal form by relabeling the indeces.

   These two properties are strongly related to ergodicity.

We are now going to search for a solution of our master equation Eq. (171) once a specific initial condition of the type:

$$p_\alpha(0) = \tilde{p}_\alpha \tag{185}$$

is set. We look for a solution of this equation in the basis of the right eigenvectors:

$$p_\beta(t) = \sum_\gamma c^\gamma(t) v_\beta^\gamma \tag{186}$$

and write:

$$\begin{aligned} \sum_\gamma c^\gamma(t+\tau) v_\alpha^\gamma = \sum_{\alpha\beta} c^\gamma(t) v_\beta^\gamma = \\ = \sum_\gamma c^\gamma(t) \lambda^\gamma v_\alpha^\gamma \end{aligned} \tag{187}$$

from which we obtain:

$$\sum_{\alpha,\gamma} u_\alpha^{\gamma'} c^\gamma(t+\tau) v^\gamma)\alpha = \sum_{\gamma\alpha} u_\alpha^{\gamma'} c^\gamma(t) \lambda^\gamma v_\alpha^\gamma. \tag{188}$$

Here we use mutual orthonormality and sum over γ to get:

$$C^\gamma(t+\tau) = \lambda^\gamma c^\gamma(t). \tag{189}$$

We can now iterate this equation:

$$C^\gamma(t) = C^\gamma(t-\tau)\lambda^\gamma = C^\gamma(t-2\tau)(\lambda^\gamma)^2 = \cdots = C^\gamma(0)(\lambda^\gamma)^{\frac{t}{\tau}} \tag{190}$$

This tells us that the probability at time t is given by:

$$p_\beta(t) = \sum_\gamma \left[c^\gamma(0)(\lambda^\gamma)^{\frac{t}{\tau}} \right] v_\beta^\gamma \tag{191}$$

where $C^\gamma(0)$ are given by imposing:

$$p_\alpha(0) = \tilde{p}_\alpha = \sum_\gamma c^\gamma(0) v_\alpha^\gamma. \tag{192}$$

Therefore we have

$$\sum_\alpha u_\alpha^{\gamma'} \tilde{p}_\alpha = \sum_{\alpha,\gamma} u_\alpha^{\gamma'} c^\gamma(0) v_\alpha^\gamma \rightarrow c^\gamma(0) = \sum_\alpha u_\alpha^\gamma \tilde{p}_\alpha \tag{193}$$

that allows for an explicit form for $p_\alpha(t)$ as a function of time and of the initial condition. In particular, since:

$$\lambda^0 = 1, u_\alpha^0 = 1, v_\alpha^0 = p_\alpha^{eq} \tag{194}$$

one gets:

$$c^0(0) = \sum_\alpha u_\alpha^0 \tilde{p}_\alpha = \sum_\alpha \tilde{p}_\alpha = 1 \quad \forall \tilde{p}_\alpha \tag{195}$$

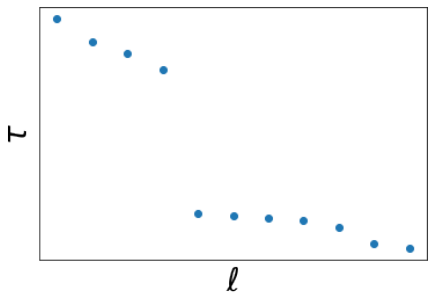

Figure 20: Plot of relaxations times as a function of ℓ.

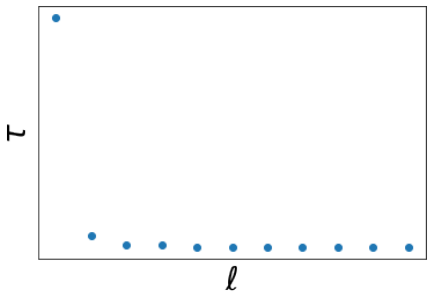

Figure 21: Plot of relaxations times: single relevant timescale.

We then rewrite Eq. (191):

$$p_\beta(t) = C^0(0)(\lambda^0)^{\frac{t}{\tau}} v_\beta^0 + \cdots = p_\beta^{eq} + \sum_\gamma (\lambda^\gamma)^{t/\tau} C^\gamma(0) v_\beta^\gamma \tag{196}$$

This implies that the second term will just lead to contributions that are vanishing in the long time limit, that is:

$$\lim_{t\to\infty} p_\beta(t) = p_\beta^{eq} \quad \forall \tilde{p}_\beta. \tag{197}$$

Evidently p_β^{eq}, with our assumptions, is only stationary solution. Note that this will be violated in case the 0-eigenvalue is not degenerate.

15.1 Dimensional reduction in time-ordered data sets

How can we use this machinery to do dimensional reduction? The key element to establish this is to give a physical interpretation to the eigenvalues $\lambda = |\lambda| e^{i\phi}$. We have in general:

$$(\lambda^\gamma)^{t/\tau} = |\lambda^\gamma|^{t/\tau} e^{i\phi^\gamma \frac{t}{\tau}} \tag{198}$$

The second term is an oscillation; the first term can be rewritten as:

$$|\lambda^\gamma|^{t/\tau} = \exp\left\{-\frac{t}{\tau^\gamma}\right\}, \quad \tau^\gamma = -\frac{\tau}{\ln|\lambda^\gamma|} \tag{199}$$

and τ^γ is called the implicit time scale γ of the dynamic process (an inverse decay rate).
For the sake of simplicity, let us assume we can neglect the phases (including them will just make notations more cumbersome). We can then write:

$$p_\beta(t) = p_\beta^{eq} + v_\beta^1 e^{-t/\tau^1} c^1 + v_\beta^2 e^{-t/\tau^2} c^2 + \ldots \tag{200}$$

Typically, we plot the relaxation times as a function of ℓ (see Fig.20). One can

see that, in case there is a clear separation of timescales (Fig. 21), e.g. $\tau^1 \ll \tau^2$, it is possible to efficiently approximate the dynamics of the probabilities, such as:

$$p_\beta(t) = p_\beta^{eq} + v_\beta^1 e^{-t/\tau^1} c^1 + v_\beta^2 e^{-t/\tau^2} c^2 + O(e^{-\frac{t}{\tau^3}}) \tag{201}$$

If a gap is present in the implicit time scales, one is allowed to truncate the expansion in a manner that keeps errors under control.
Let us consider for instance a case with $\tau^1 = 1.3, \tau^2 = 0.3$ (Fig. 21). I can approximate my dynamics with a single timescale (this typically happens in systems with first order phase transitions), that is:

$$p_\beta(t) \simeq p_\beta^{eq} + v_\beta^1 e^{-t/\tau^1} c^1 + \dots \tag{202}$$

very similar to PCA: we are projecting onto an orthonormal basis. The properties of the second vector are:

$$\sum_\alpha v_\alpha^1 u_\alpha^0 = 0 \to \sum_\alpha v_\alpha^1 = 0 \tag{203}$$

since u^0 is made only of 1s. In general, the right eigenvectors of a stochastic matrix always sum to 0 except for the equilibrium probability distribution.

The relaxation to equilibrium described by Eq. (203), due to the different signs in v_α^1, has contributions that approach p^{eq} from both above and below. One can then define macrostates (or Markov states): sets of microstates ($p_\beta(0) = p_\beta^{eq} + c v_\beta^1$) that relax to equilibrium form the same direction, that is, the sign structure of v_β^1 is the same (there exist L of those). They can be defined as:

$$A = \{\alpha; v_\alpha^1 < 0\} \tag{204}$$

so it is a set of indices that have the same sign.
All of these macrostates share a very important physical property: they relax to equilibrium to the same direction *whatever* $\tilde{p}_\alpha$ is.
What happens if the relevant timescales are d? Then, the microstates are defined as:

$$A_1 = \{\alpha : v_\alpha^1 < 0, v_\alpha^2 < 0, \dots, v_\alpha^d < 0\} \tag{205}$$

$$A_2 = \{\alpha : v_\alpha^1 < 0, v_\alpha^2 > 0, \dots, \alpha^d < 0\} \tag{206}$$

so that we have a large set of microstates. Those are very important as they allow to describe the dynamics of the system as transitions between these microstates.
Back to the original language, my ordered data set X^t can be described mesoscopically by assuming the system performs a set of jumps between the Markov states with time scales τ_ℓ, which means that we have obtained a very significant dimensional reduction.

Protocol- Our procedure went over the following steps:

1. start from raw data, $X^t \in \mathbb{R}^D$
2. microstates (of order 10000)
3. if a gap in τ_ℓ is present, d relaxation times (order d^2 Markov states), $Y^t \in \mathbb{R}^d$ are the right eigenvectors of Π. Those cannot be written explicitly;
4. still, one can write $y^\gamma_\beta = v^\gamma_\beta$, that play the role of the principal components here.

From the evolution of the microstates we might derive the evolution of the full probability density.

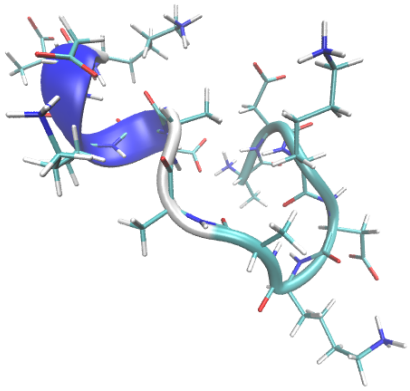

Figure 22: A single data points: Snapshot of a configuration of the peptide along the MD trajectory.

16 Exercises

16.1 PCA

For a given data set associated with a response variable:

1. Divide the data set into two subsets, a test data set (the first 1000 data points) and a learning data set (the rest of the data set).

2. Use a linear model to fit the response variable using all the coordinates in the learning data set. Then evaluate the performance of the model on the test data set.

3. Now fit (always fitting in the learning set and evaluating the performance in the test data set) with an increasing number of coordinates, starting just using only the first coordinate and going ahead until you have included all the coordinates but the last. At which point the quality of the fit is similar to the one using all the coordinates?

4. Perform a PCA analysis on the learning data set. Plot the spectrum of the eigenvalues. Transform both the learning and the test data sets.

5. Repeat points 2 and 3 using the transformed coordinates.

16.1.1 Data set description.

Molecular dynamics simulation of a peptide: In short, this is a peptide (small piece of protein) that is simulated at the atomic level by following Newton's laws. Our data set will consist of the coordinates of some atoms (13 alpha-carbon atoms that allow the description of the important properties of a configuration of the peptide) at a given time. It is in the file "zcoordinates.dat" in which each column corresponds to one of the coordinates of the atoms (so we have 3 times 13 columns) and each row to a different time (11001 rows). Therefore, we have 11001 data points embedded in a 39d space.

We will employ the radius of gyration (a measure of the compactness of the peptide) as response function. The file "rgyr.dat" will then consist of a single column with 11001 rows, each of them corresponding to the radius of gyration associated with the data points.

16.1.2 Programming and computational details.

- The data is given as text files with real numbers.
- You are allowed to use libraries for all the matrix algebra.
- The parameters for the linear model can be easily obtained as:

$$\overrightarrow{\beta} = \left(\mathbb{X}^T \bullet \mathbb{X}\right)^{-1} \bullet \mathbb{X}^T \bullet \overrightarrow{y}$$

  Where $\mathbb{X}$ is the matrix of learning data (with an extra entry set to one for all the data points in order to deal with the constant value), $\overrightarrow{y}$ is the vector of the response function and $\overrightarrow{\beta}$ is the vector with the parameters.

- The performance of the fit is evaluated as the sum of the squared difference between the predicted and the real values. $p = \left\| \mathbb{T} \bullet \overrightarrow{\beta} - \overrightarrow{y} \right\|^2$

16.2 Exercise k-PCA.

In this exercise we are going to focus on visualization. We are going to use the same data set as in the previous exercise.

1. Make a scatter plot of the two first coordinates and color the points according with the value of rgyr.
2. Do the same, but using the two first components of the standard PCA.
3. Perform a kernel-PCA analysis of the data using gaussian kernel with $\sigma^2 = [200, 100, 50, 20, 5, 2]$. Plot both the spectrum of eigenvalues and the scatter of the first two components (always coloring them according with rgyr) for each value of the parameter. What is the optimal choice of σ? Comment the results.

4. (optional) Repeat point 3 using a polynomial kernel. Is the dimensional reduction obtained in this manner better or worse?

16.3 Exercise Intrinsic Dimension.

Again, in this exercise, we are going to use the same data set as in the previous exercise.

1. Compute the Intrinsic Dimension (I_d) using TWO-NN for the whole data set. Is this value compatible with the one expected analyzing the spectrum of PCA?

2. Undersample the dataset randomly picking X points and compute the I_d. For each different value of X, repeat the operation 10 times and compute the average and the standard deviation of the $\mathrm{I_d}$ as function of X. Use $X = [10, 20, 50, 100, 200, 500, 1000, 2000, 5000, 10000]$. Plot the results using logscale for the X.

3. (optional) Repeat points 1 and 2 using the box-counting method. Comment the differences.

References