John A. Lee Michel Verleysen

Nonlinear Dimensionality Reduction

Springer

John Lee
Molecular Imaging and Experimental Radiotherapy
Université catholique de Louvain
Avenue Hippocrate 54/69
B-1200 Bruxelles
Belgium
john.lee@uclouvain.be

Michel Verleysen
Machine Learning Group – DICE
Université catholique de Louvain
Place du Levant 3
B-1348 Louvain-la-Neuve
Belgium
michel.verleysen@uclouvain.be

Library of Congress Control Number: 2006939149

ISBN-13: 978-0-387-39350-6 e-ISBN-13: 978-0-387-39351-3

Printed on acid-free paper.

9 8 7 6 5 4 3 2 1

springer.com

To our families

Preface

Methods of dimensionality reduction are innovative and important tools in the fields of data analysis, data mining, and machine learning. They provide a way to understand and visualize the structure of complex data sets. Traditional methods like principal component analysis and classical metric multidimensional scaling suffer from being based on linear models. Until recently, very few methods were able to reduce the data dimensionality in a nonlinear way. However, since the late 1990s, many new methods have been developed and nonlinear dimensionality reduction, also called manifold learning, has become a hot topic. New advances that account for this rapid growth are, for example, the use of graphs to represent the manifold topology, and the use of new metrics like the geodesic distance. In addition, new optimization schemes, based on kernel techniques and spectral decomposition, have led to spectral embedding, which encompasses many of the recently developed methods.

This book describes existing and advanced methods to reduce the dimensionality of numerical databases. For each method, the description starts from intuitive ideas, develops the necessary mathematical details, and ends by outlining the algorithmic implementation. Methods are compared with each other with the help of different illustrative examples.

The purpose of the book is to summarize clear facts and ideas about well-known methods as well as recent developments in the topic of nonlinear dimensionality reduction. With this goal in mind, methods are all described from a unifying point of view, in order to highlight their respective strengths and shortcomings.

The book is primarily intended for statisticians, computer scientists, and data analysts. It is also accessible to other practitioners having a basic background in statistics and/or computational learning, such as psychologists (in psychometry) and economists.

Louvain-la-Neuve, Belgium
October 2006

John A. Lee
Michel Verleysen

Contents

Notations

$\mathbb{N}$	The set of positive natural numbers: $\{0, 1, 2, 3, \ldots\}$
$\mathbb{R}$	The set of real numbers
y, x	Known or unknown random variables taking their values in $\mathbb{R}$
$\mathbf{A}$	A matrix
$a_{i,j}$	An entry of the matrix $\mathbf{A}$ (located at the crossing of the ith row and the jth column)
N	Number of points in the data set
M	Number of prototypes in the codebook $\mathbf{C}$
D	Dimensionality of the data space (which is usually $\mathbb{R}^D$)
P	Dimensionality of the latent space (which is usually $\mathbb{R}^P$) (or its estimation as the intrinsic dimension of the data)
$\mathbf{I}_D$	D-dimensional identity matrix
$\mathbf{I}_{P \times D}$	Rectangular matrix containing the first P rows of $\mathbf{I}_D$
$\mathbf{1}_N$	N-dimensional column vector containing ones everywhere
$\mathbf{y}$	Random vector in the known data space: $\mathbf{y} = [y_1, \ldots, y_d, \ldots, y_D]^T$
$\mathbf{x}$	Random vector in the unknown latent space: $\mathbf{x} = [x_1, \ldots, x_p, \ldots, x_P]^T$
$\mathbf{y}(i)$	The ith vector of the data set
$\mathbf{x}(i)$	(Unknown) latent vector that generated $\mathbf{y}(i)$
$\hat{\mathbf{x}}(i)$	The estimate of $\mathbf{x}(i)$
$\mathcal{Y}$	The data set $\mathcal{Y} = \{\ldots, \mathbf{y}(i), \ldots\}_{1 \leq i \leq N}$
$\mathcal{X}$	The (unknown) set of latent vectors that generated $\mathcal{Y}$
$\hat{\mathcal{X}}$	Estimation of $\mathcal{X}$
$\mathbf{Y}$	The data set in matrix notation: $\mathcal{Y} = [\ldots, \mathbf{y}(i), \ldots]_{1 \leq i \leq N}$
$\mathbf{X}$	The (unknown) ordered set of latent vectors that generated $\mathbf{Y}$
$\hat{\mathbf{X}}$	Estimation of $\mathbf{X}$

$\mathcal{M}$	A manifold (noted as a set)
$\mathbf{m}$	The functional notation of $\mathcal{M}$: $\mathbf{y} = \mathbf{m}(\mathbf{x})$
$E_x\{x\}$	The expectation of the random variable x
$\mu_x(x)$	The mean value of the random variable x (computed with its known values $x(i)$, $i = 1, \ldots, N$)
μ_i	The ith-order centered moment
μ'_i	The ith-order raw moment
$\mathbf{C}_{\mathbf{xy}}$	The covariance matrix between the random vectors $\mathbf{x}$ and $\mathbf{y}$
$\hat{\mathbf{C}}_{\mathbf{xy}}$	The estimate of the covariance matrix
$f(\mathbf{x})$, $\mathbf{f}(\mathbf{x})$	Uni- or multivariate function of the random vector $\mathbf{x}$
$\frac{\partial f(\mathbf{x})}{\partial x_p}$	Partial derivative of f with respect to x_p
$\nabla_{\mathbf{x}} f(\mathbf{x})$	Gradient vector of f with respect to $\mathbf{x}$
$\mathbf{H}_{\mathbf{x}} f(\mathbf{x})$	Hessian matrix of f with respect to $\mathbf{x}$
$\mathbf{J}_{\mathbf{x}} \mathbf{f}(\mathbf{x})$	Jacobian matrix of $\mathbf{f}$ with respect to $\mathbf{x}$
$\langle \mathbf{y}(i) \cdot \mathbf{y}(j) \rangle$	Scalar product between the two vectors $\mathbf{y}(i)$ and $\mathbf{y}(j)$
$d(\mathbf{y}(i), \mathbf{y}(j))$	Distance function between the two vectors $\mathbf{y}(i)$ and $\mathbf{y}(j)$ (often a spatial distance, like the Euclidean one) shortened as $d_{\mathbf{y}}(i, j)$ or $d_{\mathbf{y}}$ when the context is clear
$\delta(\mathbf{y}(i), \mathbf{y}(j))$	Geodesic or graph distance between $\mathbf{y}(i)$ and $\mathbf{y}(j)$
$\mathcal{C}$, $\mathcal{G}$	Codebook (noted as a set) in the data and latent spaces
$\mathbf{C}$, $\mathbf{G}$	Codebook (noted as a matrix) in the data and latent spaces
$\mathbf{c}(r)$, $\mathbf{g}(r)$	Coordinates of the rth prototypes in the codebook (respectively, in the data and latent spaces)

Acronyms

DR	Dimensionality reduction
LDR	Linear dimensionality reduction
NLDR	Nonlinear dimensionality reduction

ANN	Artificial neural networks
EVD	Eigenvalue decomposition
SVD	Singular value decomposition
SVM	Support vector machines
VQ	Vector quantization

CCA	Curvilinear component analysis	*NLDR method*
CDA	Curvilinear distance analysis	*NLDR method*
EM	Expectation-maximization	*optimization technique*
GTM	Generative topographic mapping	*NLDR method*
HLLE	Hessian LLE (see LLE)	*NLDR method*
KPCA	Kernel PCA (see PCA)	*NLDR method*
LE	Laplacian eigenmaps	*NLDR method*
LLE	Locally linear embedding	*NLDR method*
MDS	Multidimensional scaling	*LDR/NLDR method*
MLP	Multilayer perceptron	*ANN for function approx.*
MVU	Maximum variance unfolding (see SDE)	*NLDR method*
NLM	(Sammon's) nonlinear mapping	*NLDR method*
PCA	Principal component analysis	*LDR method*
RBFN	Radial basis function network	*ANN for function approx.*
SDE	Semidefinite embedding	*NLDR method*
SDP	Semidefinite programming	*optimization technique*
SNE	Stochastic neighbor embedding	*NLDR method*
SOM	(Kohonen's) self-organizing map	*NLDR method*
TRN	Topology-representing network	*ANN*

1

High-Dimensional Data

Overview. This chapter introduces the difficulties raised by the analysis of high-dimensional data and motivates the use of appropriate methods. Both practical and theoretical motivations are given. The former ones mainly translate the need to solve real-life problems, which naturally involve high-dimensional feature vectors. Image processing is a typical example. On the other hand, theoretical motivations relate to the study of high-dimensional spaces and distributions. Their properties prove unexpected and completely differ from what is usually observed in low-dimensional spaces. The empty space phenomenon and other strange behaviors are typical examples of the so-called curse of dimensionality. Similarily, the problem of data visualization is shortly dealt with. Regarding dimensionality reduction, the chapter gives two directions to explore: the relevance of variables and the dependencies that bind them. This chapter also introduces the theoretical concepts and definitions (topology, manifolds, etc.) that are typically used in the field of nonlinear dimensionality reduction. Next, a brief section presents two simple manifolds that will be used to illustrate how the different methods work. Finally, the chapter ends with an overview of the following chapters.

1.1 Practical motivations

By essence, the world is multidimensional. To persuade yourself, just look at human beings, bees, ants, neurons, or, in the field of technology, computer networks, sensor arrays, etc. In most cases, combining a large number of simple and existing units allows us to perform a great variety of complex tasks. This solution is cheaper than creating or designing a specific device and is also more robust: the loss or malfunction of a few units does not impair the whole system. This nice property can be explained by the fact that units are often partially redundant. Units that come to failure can be replaced with others that achieve the same or a similar task.

Redundancy means that parameters or features that could characterize the set of various units are not independent from each other. Consequently, the efficient management or understanding of all units requires taking the redundancy into account. The large set of parameters or features must be summarized into a smaller set, with no or less redundancy. This is the goal of *dimensionality reduction* (DR), which is one of the key tools for analyzing high-dimensional data.

1.1.1 Fields of application

The following paragraphs present some fields of technology or science where high-dimensional data are typically encountered.

Processing of sensor arrays

These terms encompass all applications using a set of several identical sensors. Arrays of antennas (e.g., in radiotelescopes) are the best example. But to this class also belong numerous biomedical applications, such as electrocardiagram or electroencephalograph acquisition, where several electrodes record time signals at different places on the chest or the scalp. The same configuration is found again in seismography and weather forecasting, for which several stations or satellites deliver data. The problem of geographic positioning using satellites (as in the GPS or Galileo system) may be cast within the same framework too.

Image processing

Let's consider a picture as the output of a digital camera; then its processing reduces to the processing of a sensor array, like the well-known photosensitive CCD or CMOS captors used in digital photography. However, image processing is often seen as a standalone domain, mainly because vision is a very specific task that holds a priviliged place in information science.

Multivariate data analysis

In contrast with sensor arrays or pixel arrays, multivariate data analysis rather focuses on the analysis of measures that are related to each other but come from different types of sensors. An obvious example is a car, wherein the gearbox connecting the engine to the wheels has to take into account information from rotation sensors (wheels and engine shaft), force sensors (brake and gas pedals), position sensors (gearbox stick, steering wheel), temperature sensors (to prevent engine overheating or to detect glaze), and so forth. Such a situation can also occur in psychosociology: a poll often gathers questions for which the answers are from different types (true/false, percentage, weight, age, etc.).

Data mining

At first sight, data mining seems to be very close to multivariate data analysis. However, the former has a broader scope of applications than the latter, which is a classical subdomain of statistics. Data mining can deal with more exotic data structures than arrays of numbers. For example, data mining encompasses text mining. The analysis of large sets of text documents aims, for instance, at detecting similarities between texts, like common vocabulary, same topic, etc. If these texts are Internet pages, hyperlinks can be encoded in graph structures and analyzed using tools like graph embedding. Cross references in databases can be analyzed in the same way.

1.1.2 The goals to be reached

Understanding large amounts of multidimensional data requires extracting information out of them. Otherwise, data are useless. For example, in electroencephalography, neurologists are interested in finding among numerous electrodes the signals coming from well-specified regions of the brain. When automatically processing images, computers should be able to detect and estimate the movement of objects in the scene. In a car with an automatic gearbox, the on-board computer must be able to select the most appropriate gear ratio according to data from the car sensors.

In all these examples, computers have to help the user to discover and extract information that lies hidden in the huge quantity of data. Information discovery amounts to detecting which variables are relevant and how variables interact with each other. Information extraction then consists of reformulating data, using less variables. Doing so may considerably simplify any further processing of data, whether it is manual, visual, or even automated. In other words, information discovery and extraction help to

- Understand and classify the existing data (by using a “data set” or “learning set”), i.e., assign a class, a color, a rank, or a number to each data sample.
- Infer and generalize to new data (by using a “test set” or “validation set”), i.e., get a continuous representation of the data, so that the unknown class, colour, rank, or number of new data items can be determined, too.

1.2 Theoretical motivations

From a theoretical point of view, all difficulties that occur when dealing with high-dimensional data are often referred to as the “curse of dimensionality”. When the data dimensionality grows, the good and well-known properties of the usual 2D or 3D Euclidean spaces make way for strange and annoying phenomena. The following two subsections highlight two of these phenomena.

1.2.1 How can we visualize high-dimensional spaces?

Visualization is a task that regards mainly two classes of data: *spatial* and *temporal*. In the latter case, the analysis may resort to the additional information given by the location in time.

Spatial data

Quite obviously, a high dimensionality makes the visualization of objects rather uneasy. Drawing one- or two-dimensional objects on a sheet of paper seems very straightforward, even for children. Things becomes harder when three-dimensional objects have to represented. The knowledge of perspective, and its correct mastering, are still recent discoveries (paintings before the Renaissance are not very different from Egyptian papyri!). Even with today's technology, a smooth, dynamic, and realistic representation of our three-dimensional world on a computer screen requires highly specialized chips. On the other hand, three-dimensional objects can also be sculptured or carved. To replace the chisel and the hammer, computer representations of 3D objects can be materialized in a polymer bath: on the surface a laser beam is solidifying the object, layer per layer.

But what happens when more than three dimensions must be taken into account? In this case, the computer screen and the sheet of paper, with only two dimensions, become very limited. Nevertheless, several techniques exist: they use colors or multiple linear projections. Unfortunately, all these techniques are not very intuitive and are often suited only for 4D objects. As an example, Fig. 1.1 shows the projection of a 4D cube that has been projected on a plane in a linear way; the color indicates the depth. Regardless of the projection method is, it is important to remark that the human eye attempts to understand high-dimensional objects in the same way as 3D objects: it seeks distances from one point to another, tries to distinguish what is far and what is close, and follows discontinuities like edges, corners, and so on. Obviously, objects are understood by identifying the relationships between their constituting parts.

Temporal data

When it is known that data are observed in the course of time, an additional piece of information is available. As a consequence, the above-mentioned geometrical representation is no longer unique. Instead of visualizing all dimensions simultaneously in the same coordinate system, one can draw the evolution of each variable as a function of time. For example, in Fig. 1.2, the same data set is displayed "spatially" in the first plot, and "temporally" in the second one: the time structure of data is revealed by the temporal representation only. In constrast with the spatial representation, the temporal representation easily generalizes to more than three dimensions. Nevertheless,

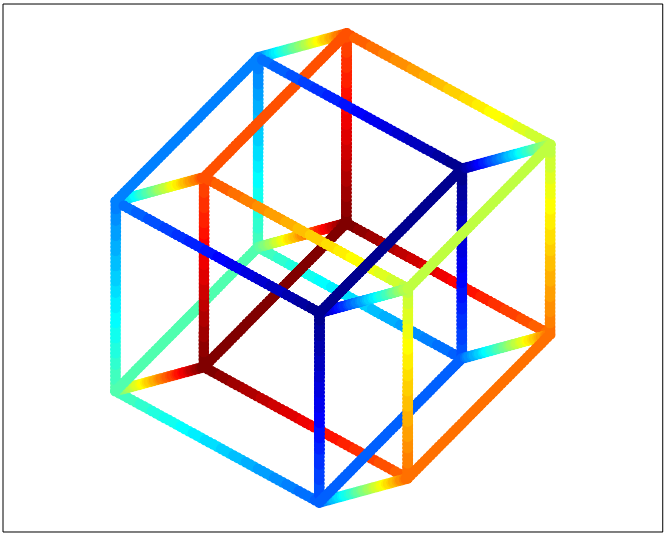


Fig. 1.1. Two-dimensional representation of a four-dimensional cube. In addition to perspective, the color indicates the depth in the fourth dimension.

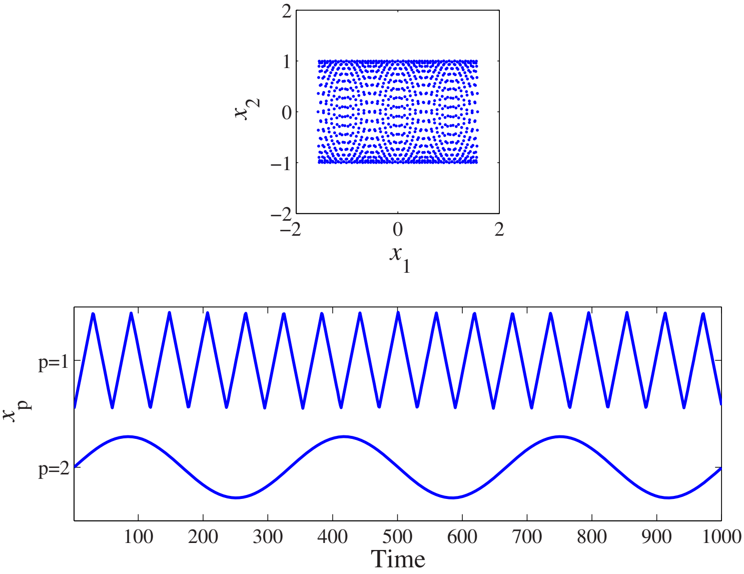


Fig. 1.2. Two plots of the same temporal data. In the first representation, data are displayed in a single coordinate system (spatial representation). In the second representation, each variable is plotted in its own coordinate system, with time as the abscissa (time representation).

when dimensionality increases, it becomes harder and harder to perceive the similarities and dissimilarities between the different variables: the eye is continually jumping from one variable to another, and finally gets lost! In such a case, a representation of data using a smaller set of variables is welcome, as for spatial data. This makes the user's perception easier, especially if each of these variables concentrates on a particular aspect of data. A compact representation that avoids redundancy while remaining trustworthy proves to be the most appealing.

1.2.2 Curse of dimensionality and empty space phenomenon

The colorful term "curse of dimensionality" was apparently first coined by Bellman [14] in connection with the difficulty of optimization by exhaustive enumeration on product spaces. Bellman underlines the fact that considering a Cartesian grid of spacing 1/10 on the unit cube in 10 dimensions, the number of points equals 10^{10}; for a 20-dimensional cube, the number of points further increases to 10^{20}. Accordingly, Bellman's interpretation is the following: if the goal consists of optimizing a function over a continuous domain of a few dozen variables by exhaustively searching a discrete search space defined by a crude discretization, one could easily be faced with the problem of making tens of trillions of evaluations of the function. In other words, the curse of dimensionality also refers to the fact that in the absence of simplifying assumptions, the number of data samples required to estimate a function of several variables to a given accuracy (i.e., to get a reasonably low-variance estimate) on a given domain grows exponentially with the number of dimensions. This fact, responsible for the curse of dimensionality, is often called the "empty space phenomenon" [170]. Because the amount of available data is generally restricted to a few observations, high-dimensional spaces are inherently sparse. More concretely, the curse of dimensionality and the empty space phenomenon give unexpected properties to high-dimensional spaces, as illustrated by the following subsections, which are largely inspired by Chapter 1 of [169].

Hypervolume of cubes and spheres

In a D-dimensional space, a sphere and the corresponding circumscripted cube (all edges equal the sphere diameter) lead to the following volume formulas:

$$V_{\text{sphere}}(r) = \frac{\pi^{D/2} r^D}{\Gamma(1 + D/2)} \quad , \tag{1.1}$$

$$V_{\text{cube}}(r) = (2r)^D \quad , \tag{1.2}$$

where r is the radius of the sphere. Surprisingly, the ratio $V_{\text{sphere}}/V_{\text{cube}}$ tends to zero when D increases:

$$\lim_{D\to\infty} \frac{V_{\text{sphere}}(r)}{V_{\text{cube}}(r)} = 0 \ . \tag{1.3}$$

Intuitively, this means that as dimensionality increases, a cube becomes more and more spiky, like a sea urchin: the spherical body gets smaller and smaller while the number of spikes increases, the latter occupying almost all the available volume. Now, assigning the value 1/2 to r, V_{cube} equals 1, leading to

$$\lim_{D\to\infty} V_{\text{sphere}}(r) = 0 \ . \tag{1.4}$$

This indicates that the volume of a sphere vanishes when dimensionality increases!

Hypervolume of a thin spherical shell

By virtue of Eq. (1.1), the relative hypervolume of a thin spherical shell is

$$\frac{V_{\text{sphere}}(r) - V_{\text{sphere}}(r(1-\epsilon))}{V_{\text{sphere}}(r)} = \frac{1^D - (1-\epsilon)^D}{1^D} \ , \tag{1.5}$$

where ϵ is the thickness of the shell ($\epsilon \ll 1$). When D increases, the ratio tends to 1, meaning that the shell contains almost all the volume [194].

Tail probability of isotropic Gaussian distributions

For any dimension D, the probability density function (pdf) of an isotropic Gaussian distribution (see Appendix B) is written as

$$f_{\mathbf{y}}(\mathbf{y}) = \frac{1}{\sqrt{(2\pi\sigma^2)^D}} \exp(-\frac{1}{2}\frac{\|\mathbf{y}-\mu_{\mathbf{y}}\|^2}{\sigma^2}) \ , \tag{1.6}$$

where $\mathbf{y}$ is a D-dimensional vector, $\mu_{\mathbf{y}}$ its D-dimensional mean, and σ^2 the isotropic (scalar) variance. Assuming the random vector $\mathbf{y}$ has zero mean and unit variance, the formula simplifies into

$$f_{\mathbf{y}}(\mathbf{y}) = K(r) = \frac{1}{\sqrt{(2\pi)^D}} \exp(-\frac{r^2}{2}) \ , \tag{1.7}$$

where $r = \|\mathbf{y}\|$ can be interpreted as a radius. Indeed, because the distribution is isotropic, the equiprobable contours are spherical. With the previous examples in mind, it can thus be expected that the distribution behaves strangely in high dimensions.

This is confirmed by computing $r_{0.95}$ defined as the radius of a hypersphere that contains 95% of the distribution [45]. The value of $r_{0.95}$ is such that

$$\frac{\int_0^{r_{0.95}} S_{\text{sphere}}(r)K(r)dr}{\int_0^{\infty} S_{\text{sphere}}(r)K(r)dr} = 0.95 \ , \tag{1.8}$$

where $S_{\text{sphere}}(r)$ is the surface of a D-dimensional hypersphere of radius r:

$$S_{\text{sphere}}(r) = \frac{2\pi^{D/2} r^{D-1}}{\Gamma(D/2)} \quad . \tag{1.9}$$

The radius $r_{0.95}$ grows as the dimensionality D increases, as illustrated in the following table:

D	1	2	3	4	5	6
$r_{0.95}$	1.96σ	2.45σ	2.80σ	3.08σ	3.33σ	3.54σ

This shows the weird behavior of a Gaussian distribution in high-dimensional spaces.

Concentration of norms and distances

Another problem encountered in high-dimensional spaces regards the weak discrimination power of a metric. As dimensionality grows, the contrast provided by usual metrics decreases, i.e., the distribution of norms in a given distribution of points tends to concentrate. This is known as the *concentration phenomenon* [20, 64].

For example, the Eulidean norm of vectors consisting of several variables that are i.i.d. (independent and identically distributed) behaves in a totally unexpected way. The explanation can be found in the following theorem (taken from [45], where the demonstration can be found as well):

Theorem 1.1. *Let* $\mathbf{y}$ *be a* D*-dimensional vector* $[y_1, \ldots, y_d, \ldots, y_D]^T$*; all components* y_d *of the vector are independent and identically distributed, with a finite eighth order moment. Then the mean* $\mu_{\|\mathbf{y}\|}$ *and the variance* $\sigma^2_{\|\mathbf{y}\|}$ *of the Euclidean norm (see Subsection 4.2.1) are*

$$\mu_{\|\mathbf{y}\|} = \sqrt{aD - b} + \mathcal{O}(D^{-1}) \tag{1.10}$$

$$\sigma^2_{\|\mathbf{y}\|} = b + \mathcal{O}(D^{-1/2}) \quad , \tag{1.11}$$

where a *and* b *are parameters depending only on the central moments of order 1, 2, 3, and 4 of the* x_i*:*

$$a = \mu^2 + \mu_2 \tag{1.12}$$

$$b = \frac{4\mu^2\mu_2 - \mu_2^2 + 4\mu\mu_3 + \mu_4}{4(\mu^2 + \mu_2)} \quad , \tag{1.13}$$

where $\mu = E\{x_d\}$ *is the common mean of all components* x_d *and* μ^k *their common central* k*-th order moment* ($\mu_k = E\{(x_d - \mu)^k\}$)*.*

In other words, the norm of random vectors grows proportionally to $\sqrt{D}$, as naturally expected, but the variance remains more or less constant for a sufficiently large D. This also means that the vector $\mathbf{y}$ seems to be normalized in high dimensions. More precisely, thanks to Chebychev's inequality, one has

$$P\left(\left|\|\mathbf{y}\| - \mu_{\|\mathbf{y}\|}\right| \geq \varepsilon\right) \leq \frac{\sigma^2_{\|\mathbf{y}\|}}{\varepsilon^2} \ , \tag{1.14}$$

i.e., the probability that the norm of $\mathbf{y}$ falls outside an interval of fixed width centered on $\mu_{\|\mathbf{y}\|}$ becomes approximately constant when D grows. As $\mu_{\|\mathbf{y}\|}$ also grows, the relative error made by taking $\mu_{\|\mathbf{y}\|}$ instead of $\|\mathbf{y}\|$ becomes negligible. Therefore, high-dimensional random i.i.d. vectors seem to be distributed close to the surface of a hypersphere of radius $\mu_{\|\mathbf{y}\|}$. This means not only that successive drawings of such random vectors yield almost the same norm, but also that the Euclidean distance between any two vectors is approximately constant. The Euclidean distance is indeed the Euclidean norm of the difference of two random vectors (see Subsection 4.2.1), and this difference is also a random vector.

In practice, the concentration phenomenon makes the nearest-neighbor search problem difficult to solve in high-dimensional spaces [20, 26]. Other results about the surprising behavior of norms and distances measured in high-dimensional spaces are given, for instance, in [1, 64] and references therein.

Diagonal of a hypercube

Considering the hypercube $[-1,+1]^D$, any segment from its center to one of its 2^D corners, i.e., a half-diagonal, can be written as $\mathbf{v} = [\pm 1, \ldots, \pm 1]^T$. The angle between a half-diagonal $\mathbf{v}$ and the dth coordinate axis

$$\mathbf{e}_d = [0, \ldots, 0, 1, 0, \ldots, 0]^T$$

is computed as

$$\cos\theta_D = \frac{\mathbf{v}^T \mathbf{e}_d}{\|\mathbf{v}\|\,\|\mathbf{e}_d\|} = \frac{\pm 1}{\sqrt{D}} \ . \tag{1.15}$$

When the dimensionality D grows, the cosine tends to zero, meaning that half-diagonals are nearly orthogonal to all coordinates axes [169]. Hence, the visualization of high-dimensional data by plotting a subset of two coordinates on a plane can be misleading. Indeed, a cluster of points lying near a diagonal line of the space will be surprisingly plotted near the origin, whereas a cluster lying near a coordinate axis is plotted as intuitively expected.

1.3 Some directions to be explored

In the presence of high-dimensional data, two possibilities exist to avoid or at least attenuate the effects of the above-mentioned phenomena. The first one focuses on the separation between relevant and irrelevant variables. The second one concentrates on the dependencies between the (relevant) variables.

1.3.1 Relevance of the variables

When analyzing multivariate data, not necessarily all variables are related to the underlying information the user wishes to catch. Irrelevant variables may be eliminated from the data set.

Most often, techniques to distinguish relevant variables from irrelevant ones are supervised: the “interest” of a variable is given by an “oracle” or “professor”. For example, in a system with many inputs and outputs, the relevance of an input can be measured by computing the correlations between known pairs of input/output. Input variables that are not correlated with the outputs may then be eliminated.

Techniques to determine whether variables are (ir)relevant are not further studied in this book, which focuses mainly on non-supervised methods. For the interested reader, some introductory references include [2, 96, 139].

1.3.2 Dependencies between the variables

Even when assuming that all variables are relevant, the dimensionality of the observed data may still be larger than necessary. For example, two variables may be highly correlated: knowing one of them brings information about the other. In that case, instead of arbitrarily removing one variable in the pair, another way to reduce the number of variables would be to find a new set of *transformed* variables. This is motivated by the facts that dependencies between variables may be very complex and that keeping one of them might not suffice to catch all the information content they both convey.

The new set should obviously contain a smaller number of variables but should also preserve the interesting characteristics of the initial set. In other words, one seeks a transformation of the variables with some well-defined properties. These properties must ensure that the transformation does not alter the information content conveyed by the initial data set, but only represents it in a different form. In the remainder of this book, linear as well as nonlinear transformations of observed variables will often be called *projections*, mainly because many transformations are designed for the preservation of characteristics that are geometrical or interpreted as such.

The type of projection must be chosen according to the model that underlies the data set. For example, if the given variables are assumed to be mixtures of a few unobserved ones, then a projection that inverts the mixing process is very useful. In other words, this projection tracks and eliminates dependencies between the observed variables. These dependencies often result from a lack of knowledge or other imperfections in the observation process: the interesting variables are not directly accessible and are thus measured in several different but largely redundant ways. The determination of a projection may also follow two different goals.

The first and simplest one aims to just detect and eliminate the dependencies. For this purpose, the projection is determined in order to reduce the number of variables. This task is traditionally known as *dimensionality reduction* and attempts to eliminate any redundancy in the initial variables. Principal component analysis (PCA) is most probably the best-known technique for dimensionality reduction.

The second and more complex goal of a projection is not only to reduce the dimensionality, but also to retrieve the so-called *latent variables*, i.e., those that are at the origin of the observed ones but cannot be measured directly. This task, in its most generic acceptation, is often called *latent variable separation*. Blind source separation (BSS), in signal processing, or Independent component analysis (ICA), in multivariate data analysis, are particular cases of latent variable separation.

As can be deduced, dimensionality reduction only focuses on the *number* of latent variables and attempts to give a low-dimensional representation of data according to this number. For this reason, dimensionality reduction does not care for the latent variables themselves: any *equivalent* representation will do. By comparison, latent variable separation is more difficult since it aims, beyond dimensionality reduction, at recovering the unknown latent variables as well as possible.

1.4 About topology, spaces, and manifolds

From a geometrical point of view, when two or more variables depend on each other, their joint distribution — or, more accurately, the *support* of their joint distribution — does not span the whole space. Actually, the dependence induces some structure in the distribution, in the form of a geometrical locus that can be seen as a kind of object in the space. The hypercube illustrated in Fig. 1.1 is an example of such a structure or object. And as mentioned above, dimensionality reduction aims at giving a new representation of these objects while preserving their structure.

In mathematics, topology studies the properties of objects that are preserved through deformations, twistings, and stretchings. Tearing is the only prohibited operation, thereby guaranteeing that the intrinsic “structure” or connectivity of objects is not altered. For example, a circle is topologically equivalent to an ellipse, and a sphere is equivalent to an ellipsoid.[1] However, subsequent chapters of this book will show that tearing still remains a very interesting operation when used carefully.

One of the central ideas of topology is that spatial objects like circles and spheres can be treated as objects in their own right: the knowledge of objects does not depend on how they are represented, or *embedded*, in space. For example, the statement, “If you remove a point from a circle, you get a (curved) line segment” holds just as well for a circle as for an ellipse, and even for

[1] Of course, this does *not* mean that soccer is equivalent to rugby!

tangled or knotted circles. In other words, topology is used to abstract the intrinsic connectivity of objects while ignoring their detailed form. If two objects have the same topological properties, they are said to be *homeomorphic*.

The “objects” of topology are formally defined as topological spaces. A *topological space* is a set for which a *topology* is specified [140]. For a set $\mathcal{Y}$, a topology T is defined as a collection of subsets of $\mathcal{Y}$ that obey the following properties:

- Trivially, $\emptyset \in T$ and $\mathcal{Y} \in T$.
- Whenever two sets are in T, then so is their intersection.
- Whenever two or more sets are in T, then so is their union.

This definition of a topology holds as well for a Cartesian space ($\mathbb{R}^D$) as for graphs. For example, the natural topology associated with $\mathbb{R}$, the set of real numbers, is the union of all open intervals.

From a more geometrical point of view, a topological space can also be defined using neighborhoods and Haussdorf’s axioms. The neighborhood of a point $\mathbf{y} \in \mathbb{R}^D$, also called a ϵ-neighborhood or infinitesimal open set, is often defined as the open ϵ-ball $B_\epsilon(\mathbf{y})$, i.e. the set of points inside a D-dimensional hollow sphere of radius $\epsilon > 0$ and centered on $\mathbf{y}$. A set containing an open neighborhood is also called a neighborhood. Then, a topological space is such that

- To each point $\mathbf{y}$ there corresponds at least one neighborhood $\mathcal{U}(\mathbf{y})$, and $\mathcal{U}(\mathbf{y})$ contains $\mathbf{y}$.
- If $\mathcal{U}(\mathbf{y})$ and $\mathcal{V}(\mathbf{y})$ are neighborhoods of the same point $\mathbf{y}$, then a neighborhood $\mathcal{W}(\mathbf{y})$ exists such that $\mathcal{W}(\mathbf{y}) \subset \mathcal{U}(\mathbf{y}) \cup \mathcal{V}(\mathbf{y})$.
- If $\mathbf{z} \in \mathcal{U}(\mathbf{y})$, then a neighborhood $\mathcal{V}(\mathbf{z})$ of $\mathbf{z}$ exists such that $\mathcal{V}(\mathbf{z}) \subset \mathcal{U}(\mathbf{y})$.
- For two distinct points, two disjoint neighborhoods of these points exist.

Within this framework, a (topological) *manifold* $\mathcal{M}$ is a topological space that is locally Euclidean, meaning that around every point of $\mathcal{M}$ is a neighborhood that is topologically the same as the open unit ball in $\mathbb{R}^D$. In general, any object that is nearly “flat” on small scales is a manifold. For example, the Earth is spherical but looks flat on the human scale.

As a topological space, a manifold can be compact or noncompact, connected or disconnected. Commonly, the unqualified term “manifold” means “manifold without boundary”. Open manifolds are noncompact manifolds without boundary, whereas closed manifolds are compact manifolds without boundary. If a manifold contains its own boundary, it is called, not surprisingly, a “manifold with boundary”. The closed unit ball $\bar{B}_1(\mathbf{0})$ in $\mathbb{R}^D$ is a manifold with boundary, and its boundary is the unit hollow sphere. By definition, every point on a manifold has a neighborhood together with a homeomorphism of that neighborhood with an open ball in $\mathbb{R}^D$.

An *embedding* is a representation of a topological object (a manifold, a graph, etc.) in a certain space, usually $\mathbb{R}^D$ for some D, in such a way that its topological properties are preserved. For example, the embedding of a manifold preserves open sets. More generally, a space $\mathcal{X}$ is embedded in another space $\mathcal{Y}$ when the properties of $\mathcal{Y}$ restricted to $\mathcal{X}$ are the same as the properties of $\mathcal{X}$.

A smooth manifold, also called an (infinitely) differentiable manifold, is a manifold together with its "functional structure" (e.g., parametric equations). Hence, a smooth manifold differs from a simple topological manifold, as defined above, because the notion of differentiability exists on it. Every smooth manifold is a topological manifold, but the reverse statement is not always true. Moreover, the availability of parametric equations allows us to relate the manifold to its latent variables, namely its parameters or degrees of freedom.

A smooth manifold $\mathcal{M}$ without boundary is said to be a *submanifold* of another smooth manifold $\mathcal{N}$ if $\mathcal{M} \subset \mathcal{N}$ and the identity map of $\mathcal{M}$ into $\mathcal{N}$ is an embedding. However, it is noteworthy that, while a submanifold $\mathcal{M}$ is just a subset of another manifold $\mathcal{N}$, $\mathcal{M}$ can have a dimension from a geometrical point of view, and the dimension of $\mathcal{M}$ may be lower than the dimension of $\mathcal{N}$. With this idea in mind, and according to [175], a *P-manifold* or *P-dimensional manifold* $\mathcal{M}$ is defined as a submanifold of $\mathcal{N} \subset \mathbb{R}^D$ if the following condition holds for all points $\mathbf{y} \in \mathcal{M}$: there exist two open sets $\mathcal{U}, \mathcal{V} \subset \mathcal{M}$, with $\mathbf{y} \in \mathcal{U}$, and a diffeomorphism $\mathbf{h} : \mathcal{U} \to \mathcal{V}, \mathbf{y} \mapsto \mathbf{x} = \mathbf{h}(\mathbf{y})$ such that

$$\mathbf{h}(\mathcal{U} \cap \mathcal{M}) = \mathcal{V} \cap (\mathbb{R}^P \times \{\mathbf{0}\}) = \{\mathbf{x} \in \mathcal{V} : x_{P+1} = \cdots = x_D = 0\} \ .$$

As can be seen, $\mathbf{x}$ can trivially be reduced to P-dimensional coordinates. If $\mathcal{N} = \mathbb{R}^D$ in the previous definition, then

- A point $\mathbf{y} \in \mathbb{R}^D$ is a manifold.
- A P-dimensional vector subspace (a P-dimensional hyperplane) is a P-manifold.
- The hollow D-dimensional hypersphere is a $(D-1)$-manifold.
- Any open subset is a D-manifold.

Whitney [202] showed in the 1930s that any P-manifold can be embedded in $\mathbb{R}^{2P+1}$, meaning that $2P+1$ dimensions *at most* are *necessary* to embed a P-manifold. For example, an open line segment is an (open) 1-manifold that can already be embedded in $\mathbb{R}^1$. On the other hand, a circle is a (compact) 1-manifold that can be embedded in $\mathbb{R}^2$ but not in $\mathbb{R}^1$. And a knotted circle, like a trefoil knot, reaches the bound of Whitney's theorem: it can be embedded only in $\mathbb{R}^D$, with $D \geq 2P+1 = 3$.

In the remainder of this book, the word *manifold* used alone typically designates a P-manifold embedded in $\mathbb{R}^D$. In the light of topology, dimensionality reduction amounts to re-embedding a manifold from a high-dimensional space to a lower-dimensional one. In practice, however, a manifold is nothing more than the underlying support of a data distribution, which is known only through a finite sample. This raises two problems. First, dimensionality reduction techniques must work with partial and limited data. Second, assuming the existence of an underlying manifold allows us to take into account the support of the data distribution but not its other properties, such as its density. This may be problematic for latent variable separation, for which a model of the data density is of prime importance.

Finally, the manifold model does not account for the noise that may corrupt data. In that case, data points no longer lie on the manifold: instead fly nearby. Hence, regarding terminology, it is correct to write that dimensionality reduction re-embeds a manifold, but, on the other hand, it can also be said that noisy data points are (nonlinearly) *projected* on the re-embedded manifold.

1.5 Two benchmark manifolds

In order to illustrate the advantages and drawbacks of the various methods of dimensionality reduction to be studied in Chapters 4 and 5, the manifolds shown in Fig. 1.3 will be used repeatedly as running examples. The first manifold, on the left in Fig. 1.3, is called the Swiss roll, according to the name of a Swiss-made cake: it is composed of a layer of airy pastry, which is spread with jam and then rolled up. The manifold shown in the figure represents the thin layer of jam in a slice of Swiss roll. The challenge of the Swiss roll consists of finding a two-dimensional embedding that “unrolls” it, in order to avoid superpositions of the successive turns of the spiral and to obtain a bijective mapping between the initial and final embeddings of the manifold. The Swiss roll is a noncompact, smooth, and connected manifold.

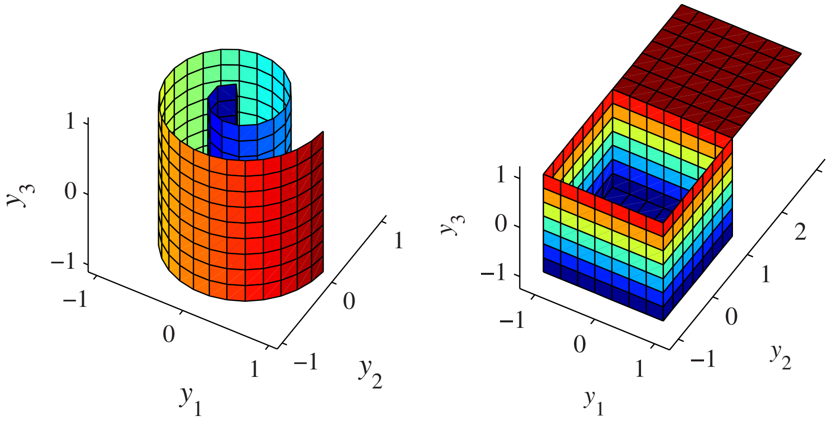


Fig. 1.3. Two benchmark manifold: the ‘Swiss roll’ and the ‘open box’.

The second two-manifold of Fig. 1.3 is naturally called the “open box”. As for the Swiss roll, the goal is to reduce the embedding dimensionality from three to two. As can be seen, the open box is connected but neither compact (in contrast with a cube or closed box) nor smooth (there are sharp edges and corners). Intuitively, it is not so obvious to guess what an embedding of the open box should look like. Would the lateral faces be stretched? Or torn? Or would the bottom face be shrunk? Actually, the open box helps to show the way each particular method behaves.

In practice, all DR methods work with a discrete representation of the manifold to be embedded. In other words, the methods are fed with a finite subset of points drawn from the manifold. In the case of the Swiss roll and open box manifolds, 350 and 316 points are selected, respectively, as shown in Fig. 1.4. The 350 and 316 available points are regularly spaced, in order to be as representative of the manifold as possible. Moreover, points are connected and displayed with different colors (indicating the height in the box or the radius in the Swiss roll). In the case of the box, points also have different shapes (small circles inside the faces, larger squares on the edges). All these features are intended to improve the readability once the manifold is mapped onto a plane, although the three-dimensional representation of Fig. 1.4 looks a bit overloaded.

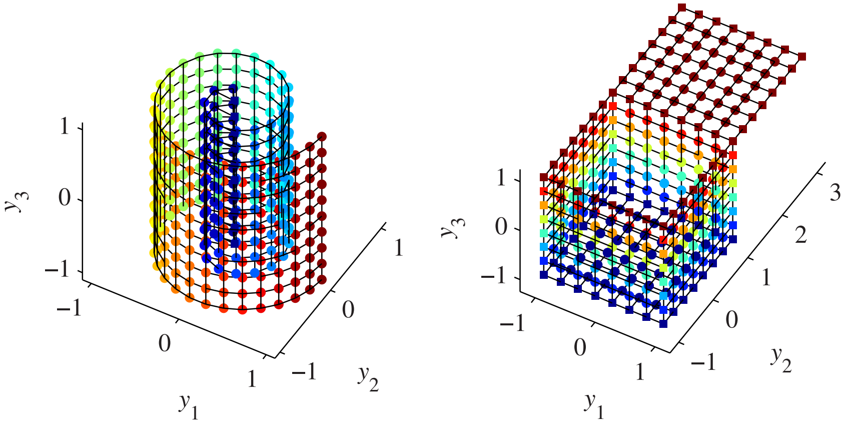


Fig. 1.4. A subset of points drawn from the “Swiss roll” and “open box” manifolds displayed in Fig. 1.3. These points are used as data sets for DR methods in order to assess their particular behavior. Corners and points on the edges of the box are shown with squares, whereas points inside the faces are shown as smaller circles. The color indicates the height of the points in the box or the radius in the Swiss roll. A lattice connects the points in order to highlight their neighborhood relationships.

1.6 Overview of the next chapters

This chapter has quickly reviewed some of the practical and theoretical reasons that raise interest toward methods of analyzing high-dimensional data. Next, Chapter 2 details the most common characteristics of such a method:

- Which functionalities are expected by the user?
- How is the underlying data model defined?
- Which criterion is to be optimized?

In order to illustrate the answers to these questions, Chapter 2 contains a description of principal component analysis (PCA), which is probably the most-known and used method of analyzing high-dimensional data. The chapter ends by listing several properties that allows us to categorize methods of nonlinear dimensionality reduction.

Because numerous DR methods do not integrate an estimator of the intrinsic dimensionality of the data, Chapter 3 describes some usual estimators of the intrinsic dimensionality. A good estimation of the intrinsic dimensionality spares a lot of time when the method takes it as an external hyperparameter. This chapter is necessary for completeness, but the reader familiar with the subject may easily skip it.

The next two chapters are dedicated to the study of two main families of DR techniques. Those techniques can be viewed as replacements, evolutions, or specializations of PCA. On one side, Chapter 4 details methods based on distance preservation. On the other side, Chapter 5 concentrates on the more elegant but more difficult principle of topology preservation. Each of these browses a wide range of classical and more recent methods, and describes them extensively. Next, Chapter 6 gives some examples and compares the results of the various methods.

Finally, Chapter 7 draws the conclusions. It summarizes the main points of the book and outlines a unifying view of the data flow for a typical method of analyzing high-dimensional data. Chapter 7 is followed by several appendices that deal with mathematical or technical details.

2

Characteristics of an Analysis Method

Overview. This chapter lists and describes the functionalities that the user usually expects from a method of analyzing high-dimensional data. Next, more mathematical details are given, such as the way data are modeled, the criterion to optimize, and the algorithm that implements them. Finally, the chapter ends by introducing a typical method, namely the principal component analysis (PCA). The method is briefly assessed by applying it to simple examples. Afterwards, and because of the limitations of linear methods like principal component analysis, this chapter promotes the use of methods that can reduce the dimensionality in a nonlinear way. In contrast with PCA, these more complex methods assume that data are generated according to a nonlinear model. The last section of this chapter attempts to list some important characteristics that allow us to classify the methods in various categories. These characteristics are, among others, the way data are modeled, the criterion to be optimized, the way it is optimized, and the kind of algorithm that implements the method.

2.1 Purpose

This chapter aims at gathering all features or properties that characterize a method of analyzing high-dimensional data. The first section lists some functionalities that the user usually expects. The next sections present more technical characterictics like the mathematical or statistical model that underlies the method, the type of algorithm that identifies the model parameters, and, last but not least, the criterion optimized by the method. Although the criterion ends the list, it often has a great influence on other characteristics. Depending on the criterion, indeed, some functionalities are available or not; similarly, the optimization of a given criterion is achieved more easily with some type of algorithm and may be more difficult with another one.

2.2 Expected functionalities

As mentioned in the previous chapter, the analysis of high-dimensional data amounts to identifying and eliminating the redundancies among the observed variables. This requires three main functionalities: an ideal method should indeed be able to

1. Estimate the number of latent variables.
2. Embed data in order to reduce their dimensionality.
3. Embed data in order to recover the latent variable.

Before detailing them, it is noteworthy that these three functionalities are not always available together in most methods. Very often, methods are able to either reduce the dimensionality or separate latent variables, but can rarely perform both. Furthermore, only a few methods include an estimator of the intrinsic dimensionality: most of them take the dimensionality as an external hyperparameter and need an additional algorithm to evaluate it.

In some cases, two or even three methods can be combined in order to achieve the three tasks: a first method gives the number of latent variables, a second one yields a low-dimensional representation of data, and, if necessary, a third one further transforms data in order to retrieve the latent variables.

2.2.1 Estimation of the number of latent variables

The first necessary step to extract information from high-dimensional data consists of computing the number of latent variables. Sometimes latent variables are also called *degrees of freedom*. But how many are there? How do we estimate their number given only a few observations in a data set?

Detailed answers to those questions are given in Chapter 3. At this point, it is just useful to know that the number of latent variables is often computed from a topological point of view, by estimating the *intrinsic dimension(ality)* of data. In contrast with a number of variables, which is necessarily an integer value, the intrinsic dimension hides a more generic concept and may take on real values. As the intrinsic dimension is the most common way to estimate the number of latent variables, the term “intrinsic dimension(ality)” is often used instead of “number of latent variables” further in this book.

The intrinsic dimension reveals the presence of a topological structure in data. When the intrinsic dimension P of data equals the dimension D of the embedding space, there is no structure: there are enough degrees of freedom so that an ϵ-ball centered on any data point can virtually be completely filled by other data points. On the contrary, when $P < D$, data points are often[1] constrained to lie in a well-delimited subspace. Consequently, a low intrinsic dimension indicates that a topological object or structure underlies the data

[1] Not always, due to the existence of fractal objects; see Chapter 3.

set. Figure 2.1 gives an example: a two-dimensional object (a surface or 2-manifold) is embedded in a three-dimensional Euclidean space. Intuitively, two parameters (or degrees of freedom or latent variables) suffice to fully describe the manifold. The intrinsic dimension estimated on a few points drawn from the surface confirms that intuition.

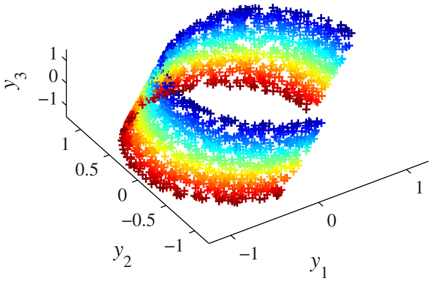


Fig. 2.1. A two-dimensional manifold embedded in a three-dimensional space. The data set contains only a finite number of points (or observations).

Without a good estimate of the intrinsic dimension, dimensionality reduction is no more than a risky bet since one does not known to what extent the dimensionality can be reduced.

2.2.2 Embedding for dimensionality reduction

The knowledge of the intrinsic dimension P indicates that data have some topological structure and do not completely fill the embedding space. Quite naturally, the following step would consist of re-embedding the data in a lower-dimensional space that would be better filled. The aims are both to get the most compact representation and to make any subsequent processing of data more easy. Typical applications include data compression and visualization.

More precisely, if the estimate of the intrinsic dimensionality P is reliable, then two assumptions can be made. First, data most probably hide a P-dimensional manifold.[2] Second, it is possible to re-embed the underlying P-dimensional manifold in a space having dimensionality between P and D, hopefully closer to P than D.

Intuitively, dimensionality reduction aims at re-embedding data in such way that the manifold structure is preserved. If this constraint is relaxed, then dimensionality reduction no longer makes sense. The main problem is,

[2] Of course, this is not necessarily true, as P is a *global* estimator and data may be a combination of several manifolds with various *local* dimensionalities.

of course, how to measure or characterize the structure of a manifold in order to preserve it.

Figure 2.2 shows a two-dimensional embedding of the manifold that was initially shown in a three-dimensional space in Fig. 2.1. After dimensionality reduction, the structure of the manifold is now completely unveiled: it is a rectangle. Obviously, for this toy example, that statement could have already been deduced by looking at the three-dimensional representation in Fig. 2.1. In this example, such a visual clue simply confirms that the dimensionality reduction worked properly and preserved the structure of the object. Actually, from the viewpoint of topology, the curvature of the rectangle in its three-dimensional embedding does not really matter, in contrast with the connectivity and local relationships between data points. More importantly, the dimensionality reduction establishes a one-to-one mapping between the three-dimensional points and the two-dimensional ones. This mapping allows us to go back to the initial embedding if necessary.

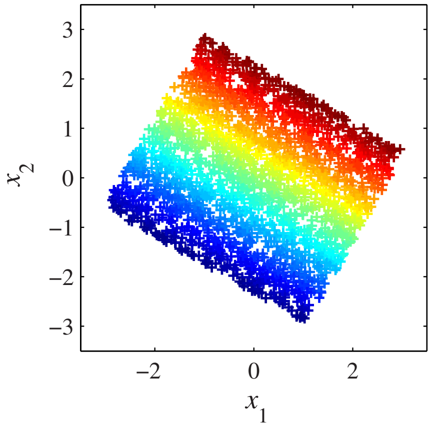


Fig. 2.2. Possible two-dimensional embedding for the object in Fig. 2.1. The dimensionality of the data set has been reduced from three to two.

2.2.3 Embedding for latent variable separation

Dimensionality reduction aims at decreasing the number of variables that describe data. In fact, most DR methods can only achieve that precise task. By comparison, the recovery of latent variables goes a step further than dimensionality reduction. Additional constraints are imposed on the desired low-dimensional representation.

These constraints are generally not related to topology. For example, it is often assumed that the latent variables that generated the data set are (statistically) independent from each other. In this case, the low-dimensional representation must also satisfy this property in order to state that the latent variables have been retrieved.

The example of Fig. 2.1, which has been re-embedded in Fig. 2.2, can be further processed, as in Fig. 2.3. In the latter representation, the two parameters of the representation, corresponding to the axes of the coordinate system, have been made independent. Intuitively, it can be seen that knowing the abscissa of a point gives no clue about the ordinate of the same point, and vice versa. This was not the case in Fig. 2.2: each abscissa determines a different interval where the ordinate may lie. It is noteworthy that the latent variable separation, whose result is illustrated in Fig. 2.3, has not been obtained directly from the three-dimensional data set in Fig. 2.1. Instead, it has been determined by modifying the low-dimensional representation of Fig. 2.2. And actually, most methods of latent variable separation are not able to reduce the dimensionality by themselves: they need another method or some kind of preprocessing to achieve it. Moreover, the additional constraints imposed on the desired representation, like statistical independence, mean that the methods are restricted to very simple data models. For example, observed variables are most often modeled as linear combinations of the latent ones, in order to preserve some of their statistical properties.

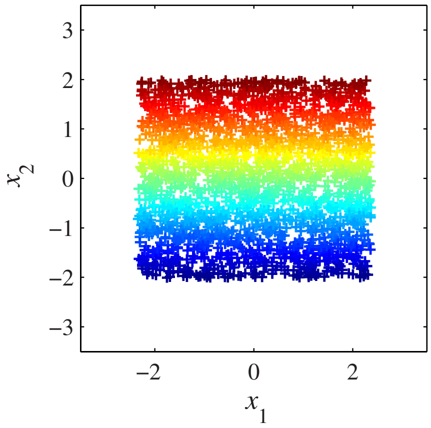


Fig. 2.3. Particular two-dimensional embedding for the object in Fig. 2.1. The latent variables, corresponding to the axes of the coordinate system, are independent from each other.

2.3 Internal characteristics

Behind the expected functionalities of an analysis method, less visible characteristics are hidden, though they play a key role. These characteristics are

- The model that data are assumed to follow.
- The type of algorithm that identifies the model parameters.
- The criterion to be optimized, which guides the algorithm.

2.3.1 Underlying model

All methods of analysis rely on the assumption that the data sets they are fed with have been generated according to a well-defined model. In colorful terms, the food must be compatible with the stomach: no vegetarian eats meat!

For example, principal component analysis (see Section 2.4) assumes that the dependencies between the variables are linear. Of course, the user should be aware of such a hypothesis, since the type of model determines the power and/or limitations of the method. Consequently for this model choice, PCA often delivers poor results when trying to project data lying on a nonlinear subspace. This is illustrated in Fig. 2.4, where PCA has been applied to the data set displayed in Fig. 2.1.

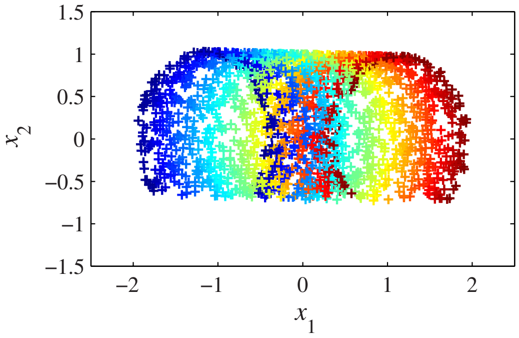


Fig. 2.4. Dimensionality reduction by PCA from 3 to 2 for the data set of Fig. 2.1. Obviously, data do not fit the model of PCA, and the initial rectangular distribution cannot be retrieved.

Hence, even for the relatively simple toy example in Fig. 2.1, methods based on a nonlinear data model seem to be preferable. The embedding of Fig. 2.2 is obtained by such a nonlinear method: the result is visually much more convincing.

The distinction between linear and nonlinear models is not the only one. For example, methods may have a continuous model or a discrete one. In the first case, the model parameters completely define a continuous function (or mapping) between the high- and low-dimensional spaces. In the second case, the model parameters determine only a few values of such a function.

2.3.2 Algorithm

For the same model, several algorithms can implement the desired method of analysis. For example, in the case of PCA, the model parameters are computed in closed form by using general-purpose algebraic procedures. Most often, these procedures work quickly, without any external hyperparameter to tune, and are guaranteed to find the best possible solution (depending on the criterion, see ahead). Nevertheless, in spite of many advantages, one of their major drawbacks lies in the fact that they are so-called batch methods: they cannot start working until the whole set of data is available.

When data samples arrive one by one, other types of algorithms exist. For example, PCA can also be implemented by so-called online or adaptative algorithms (see Subsection 2.4.4). Each time a new datum is available, on-line algorithms handle it independently from the previous ones and then 'forget' it. Unfortunately, such algorithms do not show the same desirable properties as algebraic procedures:

- By construction, they work iteratively (with a stochastic gradient descent for example).
- They can fall in a local optimum of the criterion, i.e., find a solution that is not exactly the best, but only an approximation.
- They often require a careful adjustment of several hyperparameters (e.g., learning rates) to fasten the convergence and avoid the above-mentioned local minima.

Although PCA can be implemented by several types of algorithms, such versatility does not hold for all methods. Actually, the more complex a model is, the more difficult it is to compute its parameters in closed form. Along with the data model, the criterion to be optimized also strongly influences the algorithm.

2.3.3 Criterion

Despite the criterion is the last item in this list of the method characteristics, it probably plays the most important role. The choice of the criterion often determines which functionalities the method will offer, intervenes in the data model, and always orients the implementation to a particular type of algorithm.

Typically, the criterion to be optimized is written as a mathematical formula. For example, a well-known criterion for dimensionality reduction is the mean square error. In order to compute this criterion, the dimensionality is first reduced and then expanded back, provided that the data model could be reversed. Most often the loss of information or deterioration of the data structure occurs solely in the first step, but the second is necessary in order to have a comparison reference. Mathematically, this reconstruction error can be written as

$$E_{\text{codec}} = E_{\mathbf{y}}\{\|\mathbf{y} - \text{dec}(\text{cod}(\mathbf{y}))\|_2^2\} \ , \tag{2.1}$$

where $E\{\,\}$ is the expectation operator; the dimensionality reduction and expansion are respectively denoted by the coding and decoding functions cod and dec:

$$\text{cod} : \mathbb{R}^D \rightarrow \mathbb{R}^P, \quad \mathbf{y} \mapsto \mathbf{x} = \text{cod}(\mathbf{y}) \ , \tag{2.2}$$

$$\text{dec} : \mathbb{R}^P \rightarrow \mathbb{R}^D, \quad \mathbf{x} \mapsto \mathbf{y} = \text{dec}(\mathbf{x}) \ . \tag{2.3}$$

As explained in the next section, PCA can be derived from the reconstruction error. Of course, other criteria exist. For example, statisticians may wish to get a projection that preserves the variance initially observable in the raw data. From a more geometrical or topological point of view, the projection of the object should preserve its structure, for example, by preserving the pairwise distances measured between the observations in the data set.

If the aim is latent variable separation, then the criterion can be decorrelation. This criterion can be further enriched by making the estimated latent variables as independent as possible. The latter idea points toward independent component analysis (ICA), which is out of the scope of this book. The interested reader can find more details in [95, 34] and references therein.

As shown in the next section, several criterions described above, like minimizing the reconstruction error, maximizing the variance preservation, maximizing the distance preservation, or even decorrelating the observed variables, lead to PCA when one considers a simple linear model.

2.4 Example: Principal component analysis

Principal component analysis (PCA in short) is perhaps one of the oldest and best-known methods in multivariate analysis and data mining. PCA was introduced by Pearson [149], who used it in a biological framework. Next, PCA was further developed by Hotelling [92] in the field of psychometry. In the framework of stochastic processes, PCA was also discovered independently by Karhunen [102] and was subsequently generalized by Loève [128]. This explains why PCA is also known as the "Karhunen-Loève transform" (or expansion) in this field.

2.4.1 Data model of PCA

The model of PCA essentially assumes that the D observed variables, gathered in the random vector $\mathbf{y} = [y_1, \ldots, y_d, \ldots, y_D]^T$, result from a linear transformation $\mathbf{W}$ of P unknown latent variables, written as $\mathbf{x} = [x_1, \ldots, x_p, \ldots, x_P]^T$:

$$\mathbf{y} = \mathbf{W}\mathbf{x} \ . \tag{2.4}$$

All latent variables are assumed to have a Gaussian distribution (see Appendix B). Additionally, transformation $\mathbf{W}$ is constrained to be an axis change, meaning that the columns $\mathbf{w}_d$ of $\mathbf{W}$ are orthogonal to each other and of unit norm. In other words, the D-by-P matrix $\mathbf{W}$ is a matrix such that $\mathbf{W}^T\mathbf{W} = \mathbf{I}_P$ (but the permuted product $\mathbf{W}\mathbf{W}^T$ may differ from $\mathbf{I}_D$). A last important but not too restrictive hypothesis of PCA is that both the observed variables $\mathbf{y}$ and the latent ones $\mathbf{x}$ are centered, i.e., $E_{\mathbf{y}}\{\mathbf{y}\} = \mathbf{0}_D$ and $E_{\mathbf{x}}\{\mathbf{x}\} = \mathbf{0}_P$.

Starting from this model, how can the dimension P and the linear transformation $\mathbf{W}$ be identified starting from a finite sample of the observed variables? Usually, the sample is an unordered set of N observations (or realizations) of the random vector $\mathbf{y}$:

$$\mathcal{Y} = \{\mathbf{y}(1), \ldots, \mathbf{y}(n), \ldots, \mathbf{y}(N)\} \ , \tag{2.5}$$

but it is often more convenient to write it in matrix form:

$$\mathbf{Y} = [\mathbf{y}(1), \ldots, \mathbf{y}(n), \ldots, \mathbf{y}(N)] \ . \tag{2.6}$$

Preprocessing

Before determining P and $\mathbf{W}$, it must be checked that the observations are centered. If this is not the case, they can be centered by removing the expectation of $\mathbf{y}$ from each observation $\mathbf{y}(n)$:

$$\mathbf{y}(n) \leftarrow \mathbf{y}(n) - E_{\mathbf{y}}\{\mathbf{y}\} \ , \tag{2.7}$$

where the left arrow means that the variable on the left-hand side is assigned a new value indicated in the right-hand side. Of course, the exact expectation of $\mathbf{y}$ is often unknown and must be approximated by the sample mean:

$$E_{\mathbf{y}}\{\mathbf{y}\} \approx \frac{1}{N}\sum_{n=1}^{N}\mathbf{y}(n) = \frac{1}{N}\mathbf{Y}\mathbf{1}_N \ . \tag{2.8}$$

With the last expression of the sample mean in matrix form, the centering can be rewritten for the entire data set as

$$\mathbf{Y} \leftarrow \mathbf{Y} - \frac{1}{N}\mathbf{Y}\mathbf{1}_N\mathbf{1}_N^T \ . \tag{2.9}$$

Once data are centered, P and $\mathbf{W}$ can be identified by PCA.

Nevertheless, the data set may need to be further preprocessed. Indeed, the components y_d of the observed vector $\mathbf{y}$ may come from very different origins. For example, in multivariate data analysis, one variable could be a weight expressed in kilograms and another variable a length expressed in millimeters.

But the same variables could as well be written in other units, like grams and meters. In both situations, it is expected that PCA detects the same dependencies between the variables in order to yield the same results. A simple way to solve this indeterminacy consists of standardizing the variables, i.e., dividing each y_d by its standard deviation after centering. Does this mean that the observed variables should always be standardized? The answer is negative, and actually the standardization could even be dangerous when some variable has a low standard deviation. Two cases should be distinguished from the others:

- When a variable is zero, its standard deviation is also zero. Trivially, the division by zero must be avoided, and the variable should be discarded. Alternatively, PCA can detect and remove such a useless zero-variable in a natural way.
- When noise pollutes an observed variable having a small standard deviation, the contribution of the noise to the standard deviation may be proportionally large. This means that discovering the dependency between that variable and the other ones can be difficult. Therefore, that variable should intuitively be processed exactly as in the previous case, that is, either by discarding it or by avoiding the standardization. The latter could only amplify the noise. By definition, noise is independent from all other variables and, consequently, PCA will regard the standardized variable as an important one, while the same variable would have been a minor one without standardization.

These two simple cases demonstrate that standardization can be useful but may not be achieved blindly. Some knowledge about the data set is necessary.

After centering (and standardization if appropriate), the parameters P and $\mathbf{W}$ can be identified by PCA. Of course, the exact values of P and $\mathbf{W}$ depend on the criterion optimized by PCA.

2.4.2 Criteria leading to PCA

PCA can be derived from several criteria, all leading to the same method and/or results. Two criteria — minimal reconstruction error and maximal preserved variance — are developed in the following two subsections. A third criterion — distance preservation — is detailed further in Subsection 4.2.2, which describes metric multidimensional scaling.

Minimal reconstruction error

This approach is due to Pearson [149]. Starting from the linear model of PCA (Eq. (2.4)), the coding and decoding functions (resp., Eq. (2.2) and Eq. (2.3)) can be rewritten as

$$\mathrm{cod} : \mathbb{R}^D \to \mathbb{R}^P, \quad \mathbf{y} \mapsto \mathbf{x} = \mathrm{cod}(\mathbf{y}) = \mathbf{W}^+\mathbf{y} \ , \tag{2.10}$$

$$\mathrm{dec} : \mathbb{R}^P \to \mathbb{R}^D, \quad \mathbf{x} \mapsto \mathbf{y} = \mathrm{dec}(\mathbf{x}) = \mathbf{W}\mathbf{x} \ , \tag{2.11}$$

where $\mathbf{W}^+ = (\mathbf{W}^T\mathbf{W})^{-1}\mathbf{W}^T = \mathbf{W}^T$ is the (left) pseudo-inverse of $\mathbf{W}$. Then the reconstruction mean square error (Eq. (2.1)) becomes:

$$E_{\text{codec}} = E_{\mathbf{y}}\{\|\mathbf{y} - \mathbf{W}\mathbf{W}^T\mathbf{y}\|_2^2\} \ , \tag{2.12}$$

As already remarked above, the equality $\mathbf{W}^T\mathbf{W} = \mathbf{I}_P$ holds, but $\mathbf{W}\mathbf{W}^T = \mathbf{I}_D$ is not necessarily true. Therefore, no simplification may occur in the reconstruction error. However, in a perfect world, the observed vector $\mathbf{y}$ has been generated precisely according to the PCA model (Eq. (2.4)). In this case only, $\mathbf{y}$ can be perfectly retrieved. Indeed, if $\mathbf{y} = \mathbf{W}\mathbf{x}$, then

$$\mathbf{W}\mathbf{W}^T\mathbf{y} = \mathbf{W}\mathbf{W}^T\mathbf{W}\mathbf{x} = \mathbf{W}\mathbf{I}_P\mathbf{x} = \mathbf{y} \ ,$$

and the reconstruction error is zero. Unfortunately, in almost all real situations, the observed variables in $\mathbf{y}$ are polluted by some noise, or do not fully respect the linear PCA model, yielding a nonzero reconstruction error. As a direct consequence, $\mathbf{W}$ cannot be identified perfectly, and only an approximation can be computed.

The best approximation is determined by developing and minimizing the reconstruction error. According to the definition of the Euclidean norm (see Subsection 4.2.1), E_{codec} successively becomes

$$\begin{aligned}
E_{\text{codec}} &= E_{\mathbf{y}}\{\|\mathbf{y} - \mathbf{W}\mathbf{W}^T\mathbf{y}\|_2^2\} \\
&= E_{\mathbf{y}}\{(\mathbf{y} - \mathbf{W}\mathbf{W}^T\mathbf{y})^T(\mathbf{y} - \mathbf{W}\mathbf{W}^T\mathbf{y})\} \\
&= E_{\mathbf{y}}\{\mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T\mathbf{W}\mathbf{W}^T\mathbf{y} + \mathbf{y}^T\mathbf{W}\mathbf{W}^T\mathbf{W}\mathbf{W}^T\mathbf{y}\} \\
&= E_{\mathbf{y}}\{\mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T\mathbf{W}\mathbf{W}^T\mathbf{y} + \mathbf{y}^T\mathbf{W}\mathbf{W}^T\mathbf{y}\} \\
&= E_{\mathbf{y}}\{\mathbf{y}^T\mathbf{y} - \mathbf{y}^T\mathbf{W}\mathbf{W}^T\mathbf{y}\} \\
&= E_{\mathbf{y}}\{\mathbf{y}^T\mathbf{y}\} - E_{\mathbf{y}}\{\mathbf{y}^T\mathbf{W}\mathbf{W}^T\mathbf{y}\} \ ,
\end{aligned} \tag{2.13}$$

where the first term is constant. Hence, minimizing E_{codec} turns out to maximize the term $E_{\mathbf{y}}\{\mathbf{y}^T\mathbf{W}\mathbf{W}^T\mathbf{y}\}$. As only a few observations $\mathbf{y}(n)$ are available, the latter expression is approximated by the sample mean:

$$E_{\mathbf{y}}\{\mathbf{y}^T\mathbf{W}\mathbf{W}^T\mathbf{y}\} \approx \frac{1}{N}\sum_{n=1}^{N}(\mathbf{y}(n))^T\mathbf{W}\mathbf{W}^T(\mathbf{y}(n)) \tag{2.14}$$

$$\approx \frac{1}{N}\operatorname{tr}(\mathbf{Y}^T\mathbf{W}\mathbf{W}^T\mathbf{Y}) \ , \tag{2.15}$$

where $\operatorname{tr}(\mathbf{M})$ denotes the trace of some matrix $\mathbf{M}$. To maximize this last expression, $\mathbf{Y}$ has to be factored by singular value decomposition (SVD; see Appendix A.1):

$$\mathbf{Y} = \mathbf{V}\mathbf{\Sigma}\mathbf{U}^T \ , \tag{2.16}$$

where $\mathbf{V}$, $\mathbf{U}$ are unitary matrices and where $\mathbf{\Sigma}$ is a matrix with the same size as $\mathbf{Y}$ but with at most D nonzero entries σ_d, called singular values and located on the first diagonal of $\mathbf{\Sigma}$. The D singular values are usually sorted in descending order. Substituting in the approximation of the expectation leads to

$$E_{\mathbf{y}}\{\mathbf{Y}^T\mathbf{W}\mathbf{W}^T\mathbf{Y}\} \approx \frac{1}{N}\mathrm{tr}(\mathbf{U}\mathbf{\Sigma}^T\mathbf{V}^T\mathbf{W}\mathbf{W}^T\mathbf{V}\mathbf{\Sigma}\mathbf{U}) \ . \tag{2.17}$$

Since the columns of $\mathbf{V}$ and $\mathbf{U}$ are orthonormal vectors by construction, it is easy to see that

$$\arg\max_{\mathbf{W}} \mathrm{tr}(\mathbf{U}\mathbf{\Sigma}^T\mathbf{V}^T\mathbf{W}\mathbf{W}^T\mathbf{V}\mathbf{\Sigma}\mathbf{U}) = \mathbf{V}\mathbf{I}_{D\times P} \ , \tag{2.18}$$

for a given P ($\mathbf{I}_{D\times P}$ is a matrix made of the first P columns of the identity matrix $\mathbf{I}_D$). Indeed, the above expression reaches its maximum when the P columns of $\mathbf{W}$ are colinear with the columns of $\mathbf{V}$ that are associated with the P largest singular values in $\mathbf{\Sigma}$. Additionally, it can be trivially proved that $E_{\text{codec}} = 0$ for $\mathbf{W} = \mathbf{V}$. In the same way, the contribution of a principal component $\mathbf{v}_d$ to the E_{codec} equals σ_d^2, i.e., the squared singular value associated with $\mathbf{v}_d$.

Finally, P-dimensional latent variables are approximated by computing the product

$$\hat{\mathbf{x}} = \mathbf{I}_{P\times D}\mathbf{V}^T\mathbf{y} \ . \tag{2.19}$$

Maximal preserved variance and decorrelation

This approach is due to Hotelling [92]. From a statistical point of view, it can be assumed that the latent variables in $\mathbf{x}$ are uncorrelated (no linear dependencies bind them). In practice, this means that the covariance matrix of $\mathbf{x}$, defined as

$$\mathbf{C}_{\mathbf{xx}} = E\{\mathbf{x}\mathbf{x}^T\} \ , \tag{2.20}$$

provided $\mathbf{x}$ is centered, is diagonal. However, after the axis change induced by $\mathbf{W}$, it is very likely that the observed variables in $\mathbf{y}$ are correlated, i.e., $\mathbf{C}_{\mathbf{yy}}$ is no longer diagonal. The goal of PCA is then to get back the P uncorrelated latent variables in $\mathbf{x}$. Assuming that the PCA model holds and the covariance of $\mathbf{y}$ is known, we find that

$$\begin{aligned}
\mathbf{C}_{\mathbf{yy}} &= E\{\mathbf{y}\mathbf{y}^T\} && (2.21)\\
&= E\{\mathbf{W}\mathbf{x}\mathbf{x}^T\mathbf{W}^T\} && (2.22)\\
&= \mathbf{W}E\{\mathbf{x}\mathbf{x}^T\}\mathbf{W}^T && (2.23)\\
&= \mathbf{W}\mathbf{C}_{\mathbf{xx}}\mathbf{W}^T \ . && (2.24)
\end{aligned}$$

Since $\mathbf{W}^T\mathbf{W} = \mathbf{I}$, left and right multiplications by, respectively, $\mathbf{W}^T$ and $\mathbf{W}$ lead to

$$\mathbf{C}_{\mathbf{xx}} = \mathbf{W}^T\mathbf{C}_{\mathbf{yy}}\mathbf{W} \ . \tag{2.25}$$

Next, the covariance matrix $\mathbf{C}_{\mathbf{yy}}$ can be factored by eigenvalue decomposition (EVD; see Appendix A.2):

$$\mathbf{C_{yy}} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T \ , \tag{2.26}$$

where $\mathbf{V}$ is a matrix of normed eigenvectors $\mathbf{v}_d$ and $\mathbf{\Lambda}$ a diagonal matrix containing their associated eigenvalues λ_d, in descending order. Because the covariance matrix is symmetric and semipositive definite, the eigenvectors are orthogonal and the eigenvalues are nonnegative real numbers. Substituting in Eq. (2.25) finally gives

$$\mathbf{C_{xx}} = \mathbf{W}^T\mathbf{V}\mathbf{\Lambda}\mathbf{V}^T\mathbf{W} \ . \tag{2.27}$$

This equality holds only when the P columns of $\mathbf{W}$ are taken colinear with P columns of $\mathbf{V}$, among D ones. If the PCA model is fully respected, then only the first P eigenvalues in $\mathbf{\Lambda}$ are strictly larger than zero; the other ones are zero. The eigenvectors associated with these P nonzero eigenvalues must be kept:

$$\mathbf{W} = \mathbf{V}\mathbf{I}_{D\times P} \ , \tag{2.28}$$

yielding

$$\mathbf{C_{xx}} = \mathbf{I}_{P\times D}\mathbf{\Lambda}\mathbf{I}_{D\times P} \ . \tag{2.29}$$

This shows that the eigenvalues in $\mathbf{\Lambda}$ correspond to the variances of the latent variables (the diagonal entries of $\mathbf{C_{xx}}$).

In real situations, some noise may corrupt the observed variables in $\mathbf{y}$. As a consequence, all eigenvalues of $\mathbf{C_{yy}}$ are larger than zero, and the choice of P columns in $\mathbf{V}$ becomes more difficult. Assuming that the latent variables have larger variances than the noise, it suffices to choose the eigenvectors associated with the largest eigenvalues. Hence, the same solution as in Eq. (2.28) remains valid, and the latent variables are estimated exactly as in Eq. (2.19).

If the global variance of $\mathbf{y}$ is defined as

$$\sigma_{\mathbf{y}}^2 = \operatorname{tr}(\mathbf{C_{yy}}) = \sum_{d=1}^{D} c_{d,d} = \sum_{d=1}^{D} \lambda_d \ , \tag{2.30}$$

then the proposed solution is guaranteed to preserve a maximal fraction of the global variance. From a geometrical point of view, the columns of $\mathbf{V}$ indicates the directions in $\mathbb{R}^D$ that span the subspace of the latent variables. If these columns are called *components*, the choice of the columns associated with the largest variances justifies the name “principal component analysis”.

To conclude, it must be emphasized that in real situations the true covariance of $\mathbf{y}$ is not known but can be approximated by the sample covariance:

$$\hat{\mathbf{C}}_{\mathbf{yy}} = \frac{1}{N}\mathbf{Y}\mathbf{Y}^T \ . \tag{2.31}$$

2.4.3 Functionalities of PCA

The success of PCA finds an explanation not only in its simplicity but also in the broad applicability of the method. Indeed, PCA easily makes available the three main functionalities that a user may expect in a method that analyzes high-dimensional data.

Intrinsic dimension estimation

If the model of PCA (Eq. (2.4)) is fully respected, then only the P largest eigenvalues of $\mathbf{C}_{\mathbf{yy}}$ will depart from zero. Hence, the rank of the covariance matrix (the number of nonzero eigenvalues) indicates trivially the number of latent variables. However, when having only a finite sample, the covariance can only be approximated. Moreover, the data probably do not entirely respect the PCA model (presence of noise, etc.). For all those reasons, all D eigenvalues are often different from zero, making the estimation of the intrinsic dimension more difficult.

A first way to determine it consists of looking at the eigenvalues as such. Normally, if the model holds reasonably well, large (significant) eigenvalues correspond to the variances of the latent variables, while smaller (negligible) ones are due to noise and other imperfections. Ideally, a rather visible gap should separate the two kinds of eigenvalues. The gap can be visualized by plotting the eigenvalues in descending order: a sudden fall should appear right after the Pth eigenvalue. If the gap is not visible, plotting minus the logarithm of the normalized eigenvalues may help:

$$0 \leq -\log \frac{\lambda_d}{\lambda_1} \quad . \tag{2.32}$$

In this plot, the intrinsic dimension is indicated by a sudden ascent.

Unfortunately, when the data dimensionality D is high, there may also be numerous latent variables showing a wide spectrum of variances. In the extreme, the variances of latent variables can no longer be distinguished from the variances related to noise. In this case, the intrinsic dimension P is chosen in order to preserve an arbitrarily chosen fraction of the global variance (Eq. (2.30)). Then the dimension P is determined so as to preserve at least this fraction of variance. For example, if it is assumed that the latent variables bear 95% of the global variance, then P is the smaller integer such that the inequality

$$0.95 \leq \frac{\sum_{d=1}^{P} \lambda_d}{\sum_{d=1}^{D} \lambda_d} = \frac{\operatorname{tr}(\mathbf{I}_{P\times D}\mathbf{\Lambda}\mathbf{I}_{D\times P})}{\sigma_{\mathbf{y}}^2} \tag{2.33}$$

holds. Sometimes the threshold is set on individual variances instead of cumulated ones. For example, all components having a variance lower than 1% of $\sigma_{\mathbf{y}}^2$ are discarded. The best way to set the threshold consists of finding a threshold that separates the significant variances from the negligible ones. This turns out to be equivalent to the visual methods proposed in the previous paragraph.

More complex methods exist to set the frontier between the latent and noise subspaces, such as Akaike's information criterion [4, 126] (AIC), the Bayesian information criterion [168] (BIC), and the minimum description length [153, 126] (MDL). These methods determine the value of P on the basis of information-theoretic considerations. Their particularization to PCA, with a noisy model, is well described in [34].

Projection for dimensionality reduction

After the estimation of the intrinsic dimensionality, PCA reduces the dimension by projecting the observed variables onto the estimated latent subspace in a linear way. Equation (2.19) shows how to obtain P-dimensional coordinates from D-dimensional ones. In that equation, the dimensionality reduction is achieved by the factor $\mathbf{I}_{P \times D}$, which discards the eigenvectors of $\mathbf{V}$ associated with the $D - P$ smallest eigenvalues. On the other hand, the factor $\mathbf{V}^T$ ensures that the dimensionality reduction minimizes the loss of information. Intuitively, this is done by canceling the linear dependencies between the observed variables.

Projection for latent variable separation

Beyond reduction of data dimensionality, PCA can also separate latent variables under certain conditions. In Eq. (2.19), the separation is achieved by the factor $\mathbf{V}^T$. As clearly stated in the PCA model, the observed variables can only be a rotation of the latent ones, which have a Gaussian distribution. These are, of course, very restrictive conditions that can be somewhat relaxed.

For example, in Eq. (2.4), the columns of $\mathbf{W}$ can be solely orthogonal instead of orthonormal. In this case, the latent variables will be retrieved up to a permutation and a scaling factor.

Additionally, if all latent variables have a Gaussian distribution but $\mathbf{W}$ is any matrix, then PCA can still retrieve a set of variables along orthogonal directions. The explanation is that a set of any linear combinations of Gaussian distributions is always equivalent to a set of orthogonal combinations of Gaussian distributions (see Appendix B).

From a statistical point of view, PCA decorrelates the observed variables $\mathbf{y}$ by diagonalizing the (sample) covariance matrix. Therefore, without consideration of the true latent variables, PCA finds a reduced set of uncorrelated variables from the observed ones. Actually, PCA cancels the second-order crosscumulants, i.e., the off-diagonal entries of the covariance matrix.

Knowing that higher-order cumulants, like the skewness (third order) and the kurtosis (fourth order), are null for Gaussian variables, it is not difficult to see that decorrelating the observed variables suffices to obtain fully independent latent variables. If latent variables are no longer Gaussian, then higher-order cumulants must be taken into account. This is what is done in independent component analysis (ICA, [95, 34]), for which more complex algorithms than PCA are able to cancel higher-order cross-cumulants. This leads to latent variables that are statistically independent.

2.4.4 Algorithms

As already unveiled in Subsection 2.4.2, PCA is often implemented by general-purpose algebraic procedures. The developments of PCA from the two different criteria described in that subsection show that PCA can work in two different ways:

- by SVD (singular value decomposition; see Appendix A.1) of the matrix $\mathbf{Y}$, containing the available sample.
- by EVD (eigenvalue decomposition; see Appendix A.2) of the sample covariance $\hat{\mathbf{C}}_{\mathbf{yy}}$.

Obviously, both techniques are equivalent, at least if the singular values and the eigenvalues are sorted in the same way:

$$\begin{aligned}\hat{\mathbf{C}}_{\mathbf{yy}} &= \frac{1}{N}\mathbf{Y}\mathbf{Y}^T && (2.34)\\ &= \frac{1}{N}(\mathbf{V}\boldsymbol{\Sigma}\mathbf{U}^T)(\mathbf{U}\boldsymbol{\Sigma}^T\mathbf{V}^T) && (2.35)\\ &= \mathbf{V}(\frac{1}{N}\boldsymbol{\Sigma}\boldsymbol{\Sigma}^T)\mathbf{V}^T && (2.36)\\ &= \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^T \quad . && (2.37)\end{aligned}$$

By the way, the last equality shows the relationship between the eigenvalues and the singular values: $\lambda_d = \sigma_d^2/N$. From a numerical point of view, the SVD of the sample is more robust because it works on the whole data set, whereas EVD works only on the summarized information contained in the covariance matrix. As a counterpart, from the computational point of view, SVD is more expensive and may be very slow for samples containing many observations.

The use of algebraic procedures makes PCA a batch algorithm: all observations have to be known before PCA starts. However, online or adaptive versions of PCA exist; several are described in [95, 34]. These implementations do not offer the same strong guarantees as the algebraic versions, but may be very useful in real-time application, where computation time and memory space are limited.

When estimating the latent variables, it must be pointed out that Eq. (2.19) is not very efficient. Instead, it is much better to directly remove the unnecessary columns in $\mathbf{V}$ and to multiply by $\mathbf{y}$ afterwards, without the factor $\mathbf{I}_{P\times D}$. And if PCA works by SVD of the sample, it is noteworthy that

$$\begin{aligned}\hat{\mathbf{X}} &= \mathbf{I}_{P\times D}\mathbf{V}^T\mathbf{Y} && (2.38)\\ &= \mathbf{I}_{P\times D}\mathbf{V}^T\mathbf{V}\boldsymbol{\Sigma}\mathbf{U}^T && (2.39)\\ &= \mathbf{I}_{P\times D}\boldsymbol{\Sigma}\mathbf{U}^T \quad . && (2.40)\end{aligned}$$

As $\boldsymbol{\Sigma}$ is diagonal, the cheapest way to compute $\hat{\mathbf{X}}$ consists of copying the first P columns of $\mathbf{U}$, multiplying them by the corresponding diagonal entry of $\boldsymbol{\Sigma}$, and, finally, transposing the result.

As already mentioned, PCA assumes that the observed variables are centered. Sometimes data are also standardized, meaning that each variable y_d is scaled in order to have unit variance. This is usually done when the observed variables come from various origins and have very different variances.

The standardization allows PCA not to consider observed variables with small variances as being noise and not to discard them in the dimensionality reduction. On the other hand, the standardization can sometimes amplify variables that are really negligible. User knowledge is very useful to decide if a scaling is necessary.

2.4.5 Examples and limitations of PCA

In order to illustrate the capabilities as well the limitations of PCA, toy examples may be artificially generated. For visualization's sake, only two latent variables are created; they are embedded in a three-dimensional space, i.e., three variables are observed. Three simple cases are studied here.

Gaussian variables and linear embedding

In this first case, the two latent variables, shown in the first plot of Fig. 2.5, have Gaussian distributions, with variances 1 and 4. The observed variables, displayed in the second plot of Fig. 2.5, are obtained by multiplying the latent ones by

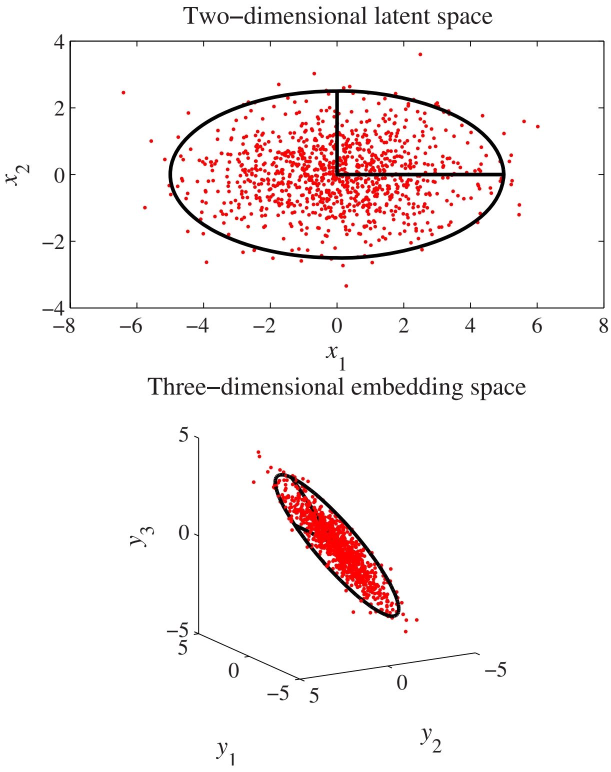
$$\mathbf{W} = \begin{bmatrix} 0.2 & 0.8 \\ 0.4 & 0.5 \\ 0.7 & 0.3 \end{bmatrix} . \tag{2.41}$$

As the mixing process (i.e., the matrix $\mathbf{W}$) is linear, PCA can perfectly reduce the dimensionality. The eigenvalues of the sample covariance matrix are 0.89, 0.11, and 0.00. The number of latent variables is then clearly two, and PCA reduces the dimensionality without any loss: the observed variables could be perfectly reconstructed from the estimated latent variables shown in Fig. 2.6. However, the columns of the mixing matrix $\mathbf{W}$ are *neither* orthogonal *nor* normed. Consequently, PCA cannot retrieve exactly the true latent variables. Yet as the latter have Gaussian distributions, PCA finds a still satisfying result: in Fig. 2.6, the estimated latent variables have Gaussian distributions but are scaled and rotated. This is visible by looking at the schematic representations of the distributions, displayed as solid and dashed ellipses. The ellipses are almost identical, but the axes indicating the directions of the true and estimated latent variables are different.

Nonlinear embedding

In this second case, the two latent variables are the same as in the previous case, but this time the mixing process is nonlinear:

$$\mathbf{y} = \begin{bmatrix} 4\cos(\frac{1}{4}x_1) \\ 4\sin(\frac{1}{4}x_1) \\ x_1 + x_2 \end{bmatrix} . \tag{2.42}$$


Two−dimensional latent space

Three−dimensional embedding space

Fig. 2.5. Two Gaussian latent variables (1000 observations, displayed in the first plot) are embedded in a three-dimensional space by a linear mixing process (second plot). The ellipse schematically represents the joint distribution in both spaces.

The observed variables are shown in Fig. 2.7. Faced with nonlinear dependencies between observed variables, PCA does not detect that only two latent variables have generated them: the normalized eigenvalues of the sample covariance matrix are 0.90, 0.05 and 0.05 again. The projection onto the first two principal components is given in Fig. 2.8. PCA is unable to completely reconstruct the curved object displayed in Fig. 2.7 with these two principal components (the reconstruction would be strictly planar!).

Unfortunately, the result of the dimensionality reduction is not the only disappointing aspect. Indeed, the estimated latent variables are completely different from the true ones. The schematic representation of the distribution is totally deformed.

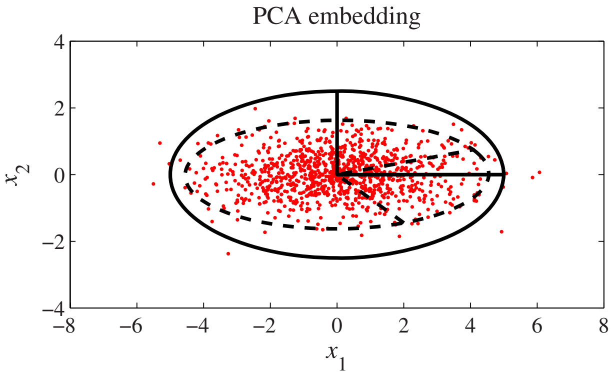


Fig. 2.6. Projection of the three-dimensional observations (second plot of Fig. 2.5) onto the two first principal components found by PCA. The solid line shows a schematic representation of the true latent distribution, whereas the dashed one corresponds to the estimated latent variables.

Non-Gaussian distributions

In this third and last case, the two latent variables are no longer Gaussian. As shown in the first plot of Fig. 2.9, their distribution is uniform. On the other hand, the mixing process is exactly the same as in the first case. Therefore, the eigenvalues of the sample covariance matrix are also identical (actually, very close, depending on the sample). The dimensionality reduction is performed without loss. However, the latent variable separation becomes really problematic. As in the first case, the estimated latent variables shown in Fig. 2.10 are rotated and scaled because the columns of **W** are not orthonormal. But because the latent variables have uniform distributions and not Gaussian ones, the scale factors and rotations make the estimated latent variables no longer uniformly distributed!

Concluding remarks about PCA

In the ideal situation, when its model is fully respected, PCA appears as a very polyvalent method to analyze data. It determines data dimensionality, builds an embedding accordingly, and retrieves the latent variables. In practice, however, the PCA model relies on assumptions that are much too restrictive, especially when it comes to latent variable separation. When only dimensionality reduction is sought, the sole remaining but still annoying assumption imposes that the dependencies between the observed variables are (not far from being) linear.

The three toy examples detailed earlier clearly demonstrate that PCA is not powerful enough to deal with complex data sets. This suggests designing other methods, maybe at the expense of PCA simplicity and polyvalence.

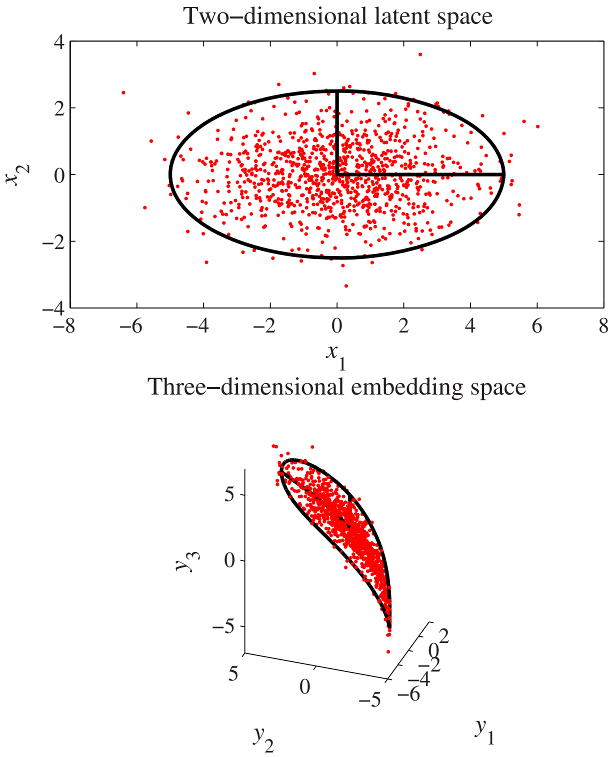


Fig. 2.7. Two Gaussian latent variables (1000 observations, displayed in the first plot) are embedded in a three-dimensional space by a nonlinear mixing process (second plot). The ellipse schematically represents the joint distribution in both spaces.

Two directions may be explored: true latent variable separation or merely dimensionality reduction. In the former case, requirements are high since an embedding has to be found that not only reduces the data dimensionality but also recovers the latent variables. If a linear model is kept, PCA can be refined in order to process non-Gaussian distributions; this leads to ICA for example. If the model cannot be assumed to be linear then latent variable separation becomes theoretically very difficult, at least when relying on statistical concepts like independence. In cases where a nonlinear model must be considered, the data analysis most often cannot go further than dimensionality reduction. Extending PCA to nonlinear models still remains an appealing challenge. Chapters 4 and 5 deal with pure dimensionality reduction. Without the necessicity of retrieving *exactly* the latent variables, more freedom is left and numerous models become possible. Nevertheless, most of them rely

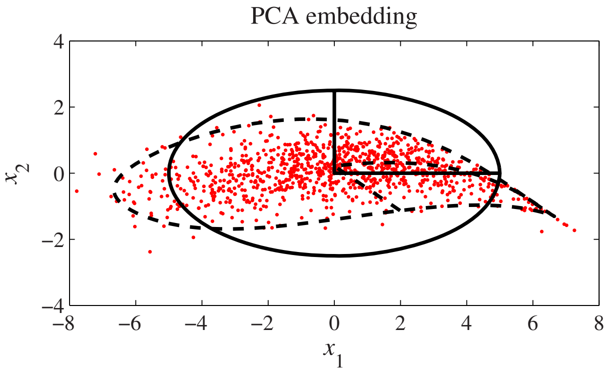


Fig. 2.8. Projection of the three-dimensional observations (second plot of Fig. 2.7) onto the first two principal components found by PCA. The solid line shows a schematic representation of the true latent distribution, whereas the dashed one corresponds to the estimated latent variables.

on geometrical considerations, ranging from simple distance measurements (Chapters 4) to more complex ideas from topology (Chapters 5).

2.5 Toward a categorization of DR methods

Before a detailed description of several DR methods, this section proposes a (nonexhaustive) list of several qualifications that will help to characterize and categorize the various methods. These qualifications mainly regard the purpose of the method, its underlying model, the mathematical criterion to be optimized and the algorithm that performs the optimization. Twelve possible qualifications are for instance

- hard vs. soft dimensionality reduction,
- traditional vs. generative model,
- linear vs. nonlinear model,
- continuous vs. discrete model,
- implicit vs. explicit mapping,
- integrated vs. external estimation of the dimensionality,
- layered vs. standalone embeddings,
- single vs. multiple coordinate systems,
- optional vs. mandatory vector quantization,
- batch vs. online algorithm,
- exact vs. approximate optimization,
- the type of criterion to be optimized.

The remainder of this section describes these features in detail.

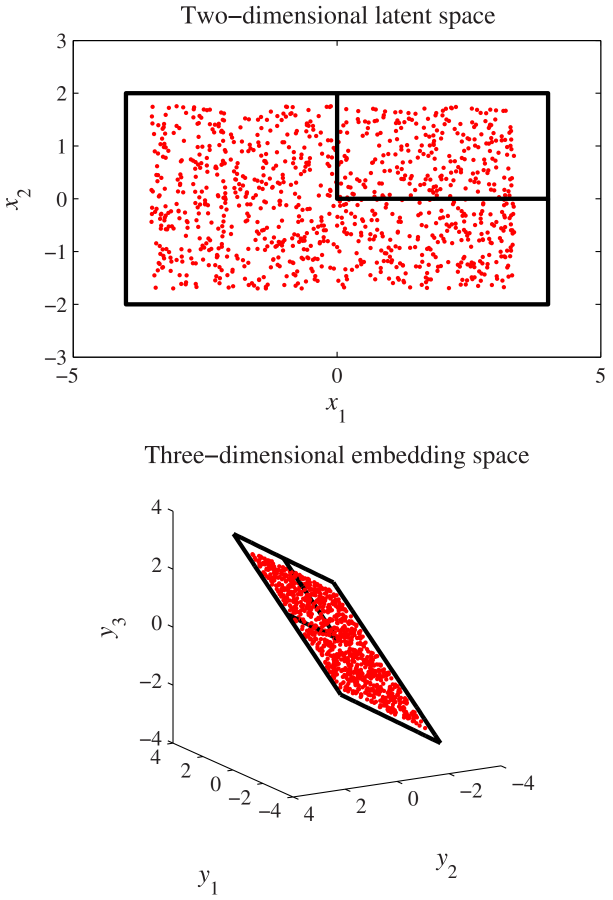


Fig. 2.9. Two uniform latent variables (1000 observations, displayed in the first plot) are embedded in a three-dimensional space by a linear mixing process (second plot). The rectangle schematically represents the joint distribution in both spaces.

2.5.1 Hard vs. soft dimensionality reduction

The problem of dimensionality reduction arises in many different applications. Depending on them, a first distinction between methods regards the ratio between the initial dimension of data and the desired dimension after re-embedding [33].

Hard dimensionality reduction is suited for problems in which the data have a dimension ranging from hundreds to maybe hundreds of thousands of variables. In such a case, a drastic dimensionality reduction is usually sought, possibly one of several orders of magnitude. The variables are often massively repeated measures of a certain phenomenon of interest in different points of

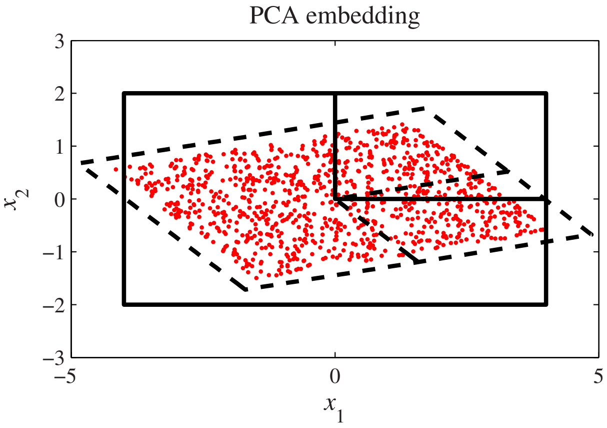


Fig. 2.10. Projection of the three-dimensional observations (second plot of Fig. 2.9) onto the first two principal components found by PCA. The solid line shows a schematic representation of the true latent distribution, whereas the dashed one corresponds to the estimated latent variables.

space (or at different instants over time). To this class belong classification and pattern recognition problems involving images or speech. Most often, the already difficult situation resulting from the huge number of variables is complicated by a low number of available samples. Methods with a simple model and few parameters like PCA are very effective for hard dimensionality reduction.

Soft dimensionality reduction is suited for problems in which the data are not too high-dimensional (less than a few tens of variables). Then no drastic dimensionality reduction is needed. Usually, the components are observed or measured values of different variables, which have a straightforward interpretation. Many statistical studies and opinion polls in domains like social sciences and psychology fall in this category. By comparison with hard problems described above, these applications usually deal with sufficiently large sample sizes. Typical methods include all the usual tools for multivariate analysis.

Finally, visualization problems lie somewhere in-between the two previous classes. The initial data dimensionality may equal any value. The sole constraint is to reduce it to one, two, or three dimensions.

2.5.2 Traditional vs. generative model

The *model* associated with a method actually refers to the way the method connects the latent variables with the observed ones. Almost each method makes different assumptions about this connection. It is noteworthy that this connection can go in both directions: from the latent to the observed variables or from the observed to the latent variables. Most methods use the second solution, which is the simplest and most used one, since the method basically goes in the same direction: the goal is to obtain an estimation of the latent variables starting from the observed ones. More principled methods prefer the first solution: they model the observed variables as a function of the unknown latent variables. This more complex solution better corresponds to the real way data are generated but often implies that those methods must go back and forth between the latent and observed variables in order to determine the model parameters. Such *generative models* are seldom encountered in the field of dimensionality reduction.

2.5.3 Linear vs. nonlinear model

The distinction between methods based on a linear or a nonlinear model is probably the straightest way to classify them. This explains why methods using a linear (resp., nonlinear) model are simply called linear (resp., nonlinear) methods. Both linear and nonlinear dimensionality reduction are denoted by the acronyms LDR and NLDR, respectively, in the remainder of this book.

Nonlinear methods are often more powerful than linear ones, because the connection between the latent variables and the observed ones may be much richer than a simple matrix multiplication. On the other hand, their model often comprises many parameters, whose identification requires large amounts of data.

For example, if PCA projects D-dimensional vectors onto a P-dimensional plane, the model comprises $\mathcal{O}(PD)$ parameters, and already $P + 1$ points suffice to entirely determine them. For a nonlinear method like an SOM (see Subsection 5.2.1), the number of parameters grows much more quickly: $3^P D$ parameters hardly suffice to describe a basic nonlinear model.

2.5.4 Continuous vs. discrete model

A third distinction about the data model regards its continuity. For example, the model of PCA given in Section 2.4, is continuous: it is a linear transform of the variables. On the other hand, the model of an SOM is discrete: it consists of a finite set of interconnected points.

The continuity is a very desirable property when the dimensionality reduction must be generalized to other points than those used to determine the model parameters. When the model is continuous, the dimensionality reduction is often achieved by using a parameterized function or mapping between the initial and final spaces. In this case, applying the mapping to new points yields their coordinates in the embedding. With only a discrete model, new points cannot be so easily re-embedded: an interpolation procedure is indeed necessary to embed in-between points.

2.5.5 Implicit vs. explicit mapping

A fourth distinction about the model, which is closely related to the previous one, regards the way a method maps the high- and low-dimensional spaces. Mainly two classes of mappings exist: explicit and implicit.

An explicit mapping consists of directly associating a low-dimensional representation with each data point. Hence, using an explicit mapping clearly means that the data model is discrete and that the generalization to new points may be difficult. Sammon's nonlinear mapping (see Subsection 4.2.3) is a typical example of explicit mapping. Typically, the parameters of such a mapping are coordinates, and their number is proportional to the number of observations in the data set.

On the other hand, an implicit mapping is defined as a parameterized function. For example, the paramaters in the model of PCA define a hyperplane. Clearly, there is no direct connection between those parameters and the coordinates of the observations stored in the data set. Implicit mappings often originate from continuous models, and generalization to new points is usually straightforward.

A third intermediate class of mappings also exists. In this class may be gathered all models that define a mapping by associating a low-dimensional representation not to each data point, but to a subset of data points. In this case, the number of low-dimensional representations does not depend strictly on the number of data points, and the low-dimensional coordinates may be considered as generic parameters, although they have a straightforward geometric meaning. All DR methods, like SOMs, that involve some form of vector quantization (see Subection 2.5.9 ahead) belong to this class.

2.5.6 Integrated vs. external estimation of the dimensionality

When considering dimensionality reduction, a key point to discuss is the presence of an estimator of the intrinsic dimensionality. The case of PCA appears as an exception, as most other methods are deprived of an integrated estimator. Actually, they take the intrinsic dimensionality as an external hyperparameter to be given by the user. In that sense, this hyper-parameter is preferably called the *embedding dimension(ality)* rather than the intrinsic dimensionality of data. This is justified by the fact that the user may wish to visualize the data on a plane and thus force a two-dimensional embedding even if the intrinsic dimensionality is actually higher. On the contrary, methods that are not powerful enough to embed highly curved P-manifolds may need more than P dimensions to work properly.

Anyway, if a dimensionality estimator is not integrated in the dimensionality reduction itself, this functionality must be performed by an external procedure. Some typical techniques to estimate the intrinsic dimensionality of manifolds are described in Chapter 3.

2.5.7 Layered vs. standalone embeddings

When performing PCA on data, all embeddings of dimensionality ranging between 1 and D are computed at once. The different embeddings are obtained by removing the coordinates along one or several of the D eigenvectors. Since the eigenvectors are orthogonal by construction, the removal of an eigenvector obviously does not change the coordinates along the kept ones. In other words, PCA proves to be an *incremental* method that produces *layered* embeddings: adding or removing a dimension does not require any change of the coordinates along the other dimensions. For instance, computing a 2D embedding can be done by taking the leading eigenvector, which specifies coordinates along a first dimension, and then the second eigenvector, in decreasing order of eigenvalue magnitude. If a 3D embedding is needed, it suffices to retrieve the 2D one, take the third eigenvector, and append the corresponding coordinates.

All methods that translate the dimensionality reduction into an eigenproblem and assemble eigenvectors to form an embedding share this capability and are called *spectral* methods. To some extent, the embeddings of different dimensionality provided by spectral methods are not independent since coordinates are added or removed but never changed when the target dimensionality increases or decreases.

In contrast, methods relying on other principles (nonspectral methods) do not offer such a comfort. When they compute an embedding of a given dimensionality, only that specific embedding is determined. If the target dimensionality changes, then all coordinates of the new embedding must be computed again. This means that the method builds *standalone* embeddings for each specified dimensionality. Although such methods seem to waste computational power, the independence of the embeddings can also be an advantage: for each dimensionality, the embedding is specifically optimized.

2.5.8 Single vs. multiple coordinate systems

Strictly speaking, dimensionality reduction does not imply establishing a low-dimensional representation in a single system of coordinates. For example, a natural extension of PCA to a nonlinear model consists in dividing a nonlinear manifold into small pieces, like a (curved) jigsaw. If these pieces are small enough, they may be considered as linear and PCA may be applied to each of them. Obviously, each PCA is independent from the others, raising the difficulty of patching together the projections of each piece of the manifold in the low-dimensional space.

One of the rare applications for which multiple systems of coordinates raise no difficulties is data compression. But in this domain more specific methods exist that reach a better compression rate by considering the data as a binary flow instead of coordinates in a Cartesian space.

Most other applications of dimensionality reduction, like visualization, usually need a single system of coordinates. Indeed, if the data lie on a smooth manifold, what would be the justification to cut it off into small pieces? One of the goals of dimensionality reduction is precisely to discover how the different parts of a manifold connect to each other. When using disconnected pieces, part of that information is lost, and furthermore it becomes impossible to visualize or to process the data as a whole.

The most widely known method using several coordinates systems is certainly the local PCA introduced by Kambathla and Leen [101]. They perform a simple vector quantization (see ahead or Appendix D for more details) on the data in order to obtain the tesselation of the manifold. More recent papers follow a similar approach but also propose very promising techniques to patch together the manifold pieces in the low-dimensional embedding space. For example, a nonparametric technique is given in [158, 166], whereas probabilistic ones are studied in [159, 189, 178, 29].

2.5.9 Optional vs. mandatory vector quantization

When the amount of available data is very large, the user may decide to work with a smaller set of representative observations. This operation can be done automatically by applying a method of vector quantization to the data set (see App. D for more details). Briefly, vector quantization replaces the original observations in the data set with a smaller set of so-called prototypes or centroids. The goal of vector quantization consists of reproducing as well as possible the shape of the initial (discrete) data distribution with the prototypes.

Unfortunately, the ideal case where data are overabondant seldom happens in real applications. Therefore, the user often skips the vector quantization and keeps the initial data set. However, some methods are designed in such a way that vector quantization is mandatory. For example, SOMs belong to this class (see Chapter 5).

2.5.10 Batch vs. online algorithm

Depending on the application, data observations may arrive consecutively or, alternatively, the whole data set may be available at once. In the first case, an online algorithm is welcome; in the second case, an offline algorithm suffices. More precisely, offline or “batch” algorithms cannot work until the whole set of observations is known. On the contrary, online algorithms typically work with no more than a single observation at a time.

For most methods, the choice of the model largely orients the implementation toward one or the other type of algorithm. Generally, the simpler the model is, the more freedom is left into the implementation. For example, as already seen in Section 2.4, the simple model of PCA naturally leads to a simple batch procedure, although online variants are possible as well. On the other hand, the more complex model of a SOM favors an online algorithm (a batch version also exists, but it is seldom used).

Actually, the behavior of true online algorithms is rather complex, especially when the data sequence does not fulfill certain conditions (like stationarity or ergodicity). For this reason, batch algorithms are usually preferred. Fortunately, most online algorithms can be made batch ones using the Robbins–Monro procedure [156]. The latter simulates a possibly infinite data sequence by repeating a finite set of observations; by construction, this method ensures some kind of stationarity. Each repetition is called an *epoch*; if the order of the available observations does not matter, then they are usually ramdomly permuted before each epoch, in order to avoid any influence on the algorithm convergence. The Robbins–Monro procedure guarantees the convergence on a solution if the algorithm satisfies certain conditions as epochs go by. These conditions typically regard parameters like step sizes, learning rates, and other time-varying parameters of online algorithms.

In this book, most algorithms that are said to be online are, in fact, implemented with the Robbins–Monro procedure for the various examples and comparisons. More details can be found in Appendix C.2.

2.5.11 Exact vs. approximate optimization

The kind of optimization that an algorithm can achieve is closely related to its offline or online character. Most often, batch algorithms result from some analytical or algebraic developments that give the solution in closed form, like PCA. Given a finite data set, which is known in advance, a batch algorithm like PCA can be proved to compute the optimal solution.

On the opposite, online or adaptive algorithms are often associated with generic optimization procedures like stochastic gradient descent (see App. C.2). Such procedures do not offer strong guarantees about the result: the convergence may fail. Nonetheless, on-line algorithms transformed into batch ones by the Robbins–Monro procedure are guaranteed to converge on a solution, after a certain number of epochs.[3] Unfortunately, this solution may be a local optimum.

Although they are slow and not optimal, generic optimization procedures have a major advantage: they are able to optimize rather complicated objective functions. On the other hand, solutions in closed form are only available for very simple objective functions; the latter ones are typically required to be not only differentiable but also concave (or convex).

2.5.12 The type of criterion to be optimized

Last but not least, the criterion that guides the dimensionality reduction is probably the most important characteristic of a DR method, even before the model specification. Actually, the data model and the algorithm are often fitted in order to satisfy the constraints imposed by the chosen criterion.

[3] An infinite number, according to the theory; see [156].

As already mentioned, the domain of dimensionality reduction is mainly motivated by geometrical considerations. From this point of view, data are interpreted as a cloud of points in a geometrical space; these points are often assumed to lie on a smooth manifold. A way to characterize a manifold independently from any coordinate system consists of discovering and making explicit the relationships between the points of the manifold. Human perception naturally focuses on proximity relations: when looking at a point in the manifold, the eyes immediately distinguish neighboring points from those lying farther away. Therefore, a good embedding of the manifold should reproduce those proximity relationships as well as possible.

Formally, this means that a criterion for a good DR method should efficiently measure the proximities observed in data and quantify their preservation in the embedding. There are two main ways to measure proximities between points, as detailed below.

The straightest one consists of computing pairwise distances between the points. As an advantage, distances are scalar values that can be easily compared to each other. Thus, a possible criterion for dimensionality reduction is distance preservation: the pairwise distances measured between the embedded points should be as close as possible to the ones measured between the initial data points. The next chapter explores this direction and shows how distance preservation can be translated in various objective functions.

On the other hand, a less intuitive but more satisfying way to measure proximities would be qualitative only. In this case, the exact value of the distances does not matter: for example, one just knows that, “From point **a**, point **b** is closer than point **c**.” The translation of such qualitative concepts into an objective function proves more difficult than for distances. Chapter 5 presents solutions to this problem.

3

Estimation of the Intrinsic Dimension

Overview. This chapter introduces the concept of intrinsic dimension along with several techniques that can estimate it. Briefly put, the intrinsic dimension can be interpreted as the number of latent variables, which is often smaller than the number of observed variables. The estimation of the number of latent variables is an essential step in the process of dimensionality reduction, because most DR methods need that number as an external and user-defined parameter. Many estimators of the intrinsic dimension come from fractal geometry. Other estimators described in this chapter are related to PCA or based on a trial-and-error approach. The chapter ends by applying the various estimators to some simple toy examples.

3.1 Definition of the intrinsic dimension

In a few words, the *intrisinc dimension(ality)* of a random vector $\mathbf{y}$ is usually defined as the minimal number of parameters or latent variables needed to describe $\mathbf{y}$. Although this informal definition seems to be clear in practice, it is formally ambiguous due to the existence of strange geometrical objects like space-filling curves (see an example in Fig. 3.1).

A better definition uses the classical concept of topological dimension [200]: the intrinsic dimension of $\mathbf{y}$ equals the topological dimension of the support $\mathcal{Y}$ of the distribution of $\mathbf{y}$. The definition requires some additional notions. Given a topological space $\mathcal{Y}$, the *covering* of a subset $\mathcal{S}$ is a collection $\mathcal{C}$ of open subsets in $\mathcal{Y}$ whose union contains $\mathcal{S}$. A *refinement* of a covering $\mathcal{C}$ of $\mathcal{S}$ is another covering $\mathcal{C}'$ such that each set in $\mathcal{C}'$ is contained in some set in $\mathcal{C}$. Because a D-dimensional set can be covered by open balls such that each point belongs to maximum $(D+1)$ open balls, the following statement holds: a subset $\mathcal{S}$ of a topological space $\mathcal{Y}$ has topological dimension D_{top} (a.k.a. Lebesgue covering dimension) if every covering $\mathcal{C}$ of $\mathcal{S}$ has a refinement $\mathcal{C}'$ in which every point of $\mathcal{S}$ belongs to at most $(D_{\text{top}}+1)$ sets in $\mathcal{C}'$, and D_{top} is

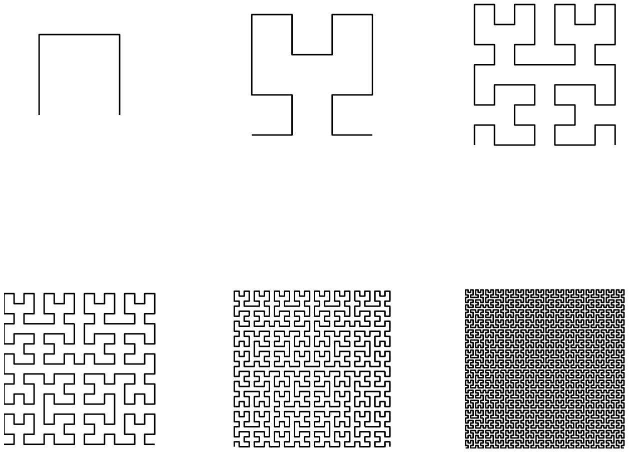


Fig. 3.1. A space-filling curve. This curve, invented by Hilbert in 1891 [86], is a one-dimensional object that evolves iteratively and progressively fills a square— a two-dimensional object! —. The first six iteration steps that are displayed show how the curve is successively refined, folded on itself in a similar way as a cabbage leaf.

the smallest such integer. For example, the Lebesgue covering dimension of the usual Euclidean space $\mathbb{R}^D$ is D.

Technically, the topological dimension is very difficult to estimate if only a finite set of points is available. Hence, practical methods use various other definitions of the intrinsic dimension. The most usual ones are related with the fractal dimension, whose estimators are studied in Section 3.2. Other definitions are based on DR methods and are summarized in Section 3.3.

Before going into further details, it is noteworthy that the estimation of the intrinsic dimension should remain coherent with the DR method: an estimation of the dimension with a nonlinear model, like the fractal dimension, makes no sense if the dimensionality reduction uses a linear model, like PCA.

3.2 Fractal dimensions

A usual generalization of the topological dimension is the *fractal dimension.* While the topological dimension defined above regards topological subsets like manifolds and yields an integer value, the fractal dimension relates to so-called fractal objects and is a real number.

Actually, the adjective *fractal* designates objects or quantities that display self-similarity, in a somewhat technical sense, on all scales. The object does not need to exhibit *exactly* the same structure on all scales, but the same type of structures must appear [200]. A classical example of a fractal object is a coastline. Figure 3.2 illustrates the coastline of Koch's island. As pointed out by Mandelbrot in his pioneering work [131, 132, 133], the length of such a coastline is different depending on the length ruler used to measure it. This paradox is known as the *coastline paradox*: the shorter the ruler, the longer the length measured.

The term *fractal dimension* [200] sometimes refers to what is more commonly called the capacity dimension (see Subsection 3.2.2). However, the term can also refer to any of the dimensions commonly used to characterize fractals, like the capacity dimension, the correlation dimension, or the information dimension. The q-dimension unifies these three dimensions.

3.2.1 The q-dimension

Let μ be a Borel probability measure on a metric space $\mathcal{Y}$ (a space provided with a metric or distance function; see Subsection 4.2 for more details). For $q \geq 0$ and $\epsilon > 0$, one defines

$$C_q(\mu, \epsilon) = \int [\mu(\bar{B}_\epsilon(\mathbf{y}))]^{q-1} d\mu(\mathbf{y}) \ , \tag{3.1}$$

where $\bar{B}_\epsilon(\mathbf{y})$ is the closed ball of radius ϵ centered on $\mathbf{y}$. Then, according to Pesin's definition [151, 152], for $q \geq 0$, $q \neq 1$, the lower and upper q-dimensions of μ are

$$D_q^-(\mu) = \liminf_{\epsilon \to 0} \frac{\log C_q(\mu, \epsilon)}{(q-1)\log \epsilon} \ , \tag{3.2}$$

$$D_q^+(\mu) = \limsup_{\epsilon \to 0} \frac{\log C_q(\mu, \epsilon)}{(q-1)\log \epsilon} \ . \tag{3.3}$$

If $D_q^-(\mu) = D_q^+(\mu)$, their common value is denoted $D_q(\mu)$ and is called the q-dimension of μ. It is expected that $D_q(\mu)$ exists for sufficiently regular fractal measures (smooth manifolds trivially fulfill this condition). For such a measure, the function $q \to D_q(\mu)$ is called the *dimension spectrum* of μ.

An alternative definition for $D_q^-(\mu)$ and $D_q^+(\mu)$ originates from the physics literature [83]. For $\epsilon > 0$, instead of using closed balls, the support of μ is covered with a (multidimensional) grid of cubes with edge length ϵ. Let $N(\epsilon)$ be the number of cubes that intersect the support of μ, and let the natural measures of these cubes be $p_1, p_2, \ldots, p_{N(\epsilon)}$. Since the p_i may be seen as the probability that these cubes are populated, they are normalized:

$$\sum_{i=1}^{N(\epsilon)} p_i = 1 \ . \tag{3.4}$$

Then

$$D_q^-(\mu) = \liminf_{\epsilon \to 0} \frac{\log \sum_{i=1}^{N(\epsilon)} p_i^q}{(q-1)\log \epsilon} \ , \tag{3.5}$$

$$D_q^+(\mu) = \limsup_{\epsilon \to 0} \frac{\log \sum_{i=1}^{N(\epsilon)} p_i^q}{(q-1)\log \epsilon} \ . \tag{3.6}$$

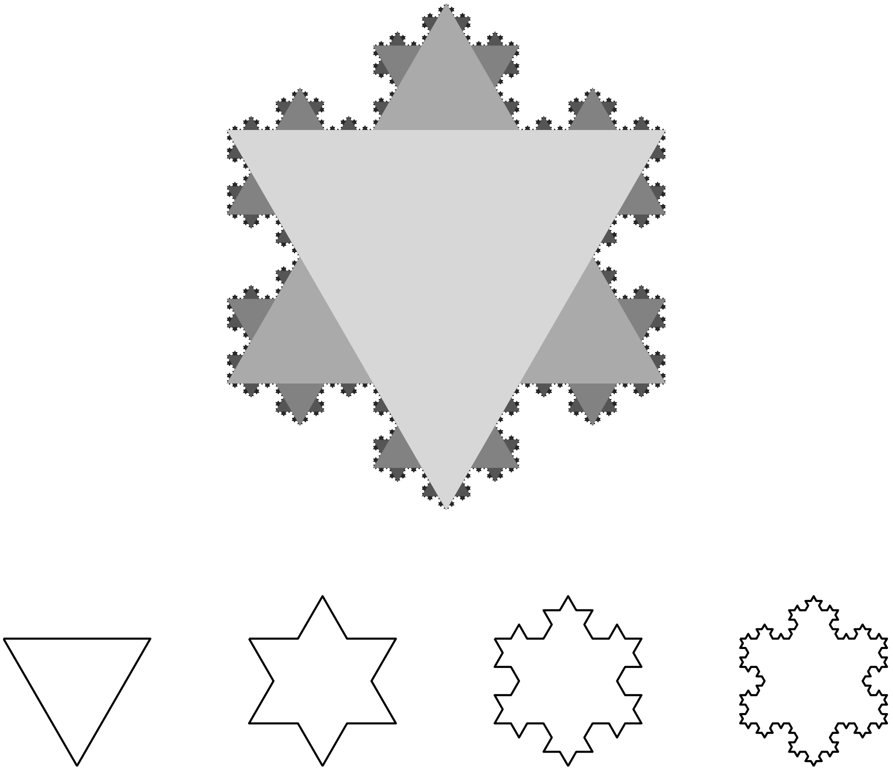


Fig. 3.2. Koch's island (or snowflake) [200]. This classical fractal object was first described by Helge von Koch in 1904. As shown in the bottom of the figure, it is built by starting with an equilateral triangle, removing the inner third of each side, replacing it with two edges of a three-times-smaller equilateral triangle, and then repeating the process indefinitely. This recursive process can be encoded as a Lindenmayer system (a kind of grammar) with initial string $S(0) =$ 'F−−F−−F' and string-rewriting rule 'F' → 'F+F−−F+F'. In each string $S(i)$, 'F' means "Go forward and draw a line segment of given length", '+' means "Turn on the left with angle $\frac{1}{3}\pi$" and '−' means "Turn on the right with angle $\frac{1}{3}\pi$". The drawings corresponding to strings $S(0)$ to $S(3)$ are shown in the bottom of the figure, whereas the main representation is the superposition of $S(0)$ to $S(4)$. Koch's island is a typical illustration of the coastline paradox: the length of the island boundary depends on the ruler used to measure it; the shorter the ruler, the longer the coastline. Actually, it is easy to see that for the representation of $S(i)$, the length of the line segment is $L(i) = L_\triangledown \left(\frac{1}{3}\right)^i$, where $L_\triangledown = L(0)$ is the side length of the initial triangle. Similarly, the number of corners is $N(i) = 3 \cdot 4^i$. Then the true perimeter associated with $S(i)$ is $l(i) = L(i)N(i) = 3L_\triangledown \left(\frac{4}{3}\right)^i$.

For $q \geq 0$, $q \neq 1$, these limits do not depend on the choice of the ϵ-grid, and give the same values as Eqs. (3.2) and (3.3). The main advantage of that second definition is that its associated formulas are more tractable in practice due to the use of sums and grids intead of integrals and balls.

3.2.2 Capacity dimension

When setting q equal to zero in the second definition (Eq. (3.5) and (3.6)) and assuming that the equality $D_q^-(\mu) = D_q^+(\mu)$ holds, one gets the *capacity dimension* [200, 152]:

$$\begin{aligned} d_{\text{cap}} = D_0(\mu) = & \lim_{\epsilon \to 0} \frac{\log \sum_{i=1}^{N(\epsilon)} p_i^0}{(0-1)\log \epsilon} \\ = & -\lim_{\epsilon \to 0} \frac{\log \sum_{i=1}^{N(\epsilon)} 1}{\log \epsilon} \\ = & -\lim_{\epsilon \to 0} \frac{\log N(\epsilon)}{\log \epsilon} \quad . \end{aligned} \tag{3.7}$$

In this definition, d_{cap} does not depend on the natural measures p_i. In practice, d_{cap} is also known as the 'box-counting' dimension [200]. When the manifold is not known analytically and only a few data points are available, the capacity dimension is quite easy to estimate:

1. Determine the hypercube that circumscribes all the data points.
2. Decompose the obtained hypercube in a grid of smaller hypercubes with edge length ϵ (these "boxes" explain the name of the method).
3. Determine $N(\epsilon)$, the number of hypercubes that are occupied by one or several data points.
4. Apply the log function, and divide by $\log \epsilon$.
5. Compute the limit when ϵ tends to zero; this is d_{cap}.

Unfortunately, the limit to be computed in the last step appears to be the sole obstacle to the overall simplicity of the technique. Subsection 3.2.6 below gives some hints to circumvent this obstacle.

The intuitive interpretation of the capacity dimension is the following. Assuming a three-dimensional space divided in small cubic boxes with a fixed edge length ϵ, the box-counting dimension is closely related to the proportion of occupied boxes. For a growing one-dimensional object placed in this compartmentalized space, the number of occupied boxes grows proportionally to the object length. Similarly, for a growing two-dimensional object, the number of occupied boxes grows proportionally to the object surface. Finally, for a growing three-dimensional object, the number of occupied boxes grows proportionally to the object volume. Generalizing to a P-dimensional object like a P-manifold embedded in $\mathbb{R}^D$, one gets

$$N(\epsilon) \propto \epsilon^P \quad . \tag{3.8}$$

And, trivially,

$$P \propto \frac{\log N(\epsilon)}{\log \epsilon} \quad . \tag{3.9}$$

To complete the analogy, the hypothesis of a growing object has to be replaced with the reciprocal one: the size of the object remains unchanged but the edge length ϵ of the boxes decreases, yielding the precise estimate of the dimension at the limit.

As an illustration, the capacity dimension can be computed analytically for the coastline of Koch's island (Fig. 3.2). In this particular case, the development is made easier by choosing triangular boxes. As shown in the caption to Fig. 3.2, the length of the line segment for the ith iteration of the Lindenmayer system is $L(i) = L_\nabla 3^{-i}$, where $L_\nabla = L(0)$ is the edge length of the initial triangle. The number of corners is $N(i) = 3 \cdot 4^i$. It is easy to see that if the grid length ϵ equals $L(i)$, the number of occupied boxes is $N(i)$. Consequently,

$$\begin{aligned} d_{\text{cap}} &= -\lim_{i\to\infty} \frac{\log N(i)}{\log L(i)} \\ &= -\lim_{i\to\infty} \frac{\log(3 \cdot 4^i)}{\log(L_\nabla 3^{-i})} \\ &= -\lim_{i\to\infty} \frac{\log 3 + i \log 4}{\log L_\nabla - i \log 3} \\ &= \frac{\log 4}{\log 3} = 1.261859507 \ . \end{aligned} \tag{3.10}$$

3.2.3 Information dimension

The information dimension corresponds to the case where $q = 1$. Care must be taken in order to avoid a denominator equal to zero in Eqs. (3.5) and (3.6). Assuming again that equality $D_q^-(\mu) = D_q^+(\mu)$ holds, it follows that

$$d_{\text{inf}} = \lim_{q\to 1} D_q(\mu) = \lim_{q\to 1} \lim_{\epsilon\to 0} \frac{\log \sum_{i=1}^{N(\epsilon)} p_i^q}{(q-1)\log \epsilon} \tag{3.11}$$

$$= \lim_{\epsilon\to 0} \frac{1}{\log \epsilon} \lim_{q\to 1} \frac{\log \sum_{i=1}^{N(\epsilon)} p_i^q}{q-1} \quad . \tag{3.12}$$

Since the p_i are normalized (Eq. (3.4)), the numerator of the right factor trivially tends to zero:

$$\lim_{q\to 1} \log \sum_{i=1}^{N(\epsilon)} p_i^q = \log \sum_{i=1}^{N(\epsilon)} p_i = \log 1 = 0 \ , \tag{3.13}$$

and so does the denominator:

$$\lim_{q \to 1} q - 1 = 0 \ . \tag{3.14}$$

Hence, using l'Hospital's rule, the numerator and denominator can be replaced with their respective derivatives:

$$\begin{aligned} d_{\text{inf}} = \lim_{q \to 1} D_q(\mu) &= \lim_{\epsilon \to 0} \frac{1}{\log \epsilon} \lim_{q \to 1} \frac{\sum_{i=1}^{N(\epsilon)} p_i^q \log p_i}{1} \\ &= \lim_{\epsilon \to 0} \frac{\sum_{i=1}^{N(\epsilon)} p_i \log p_i}{\log \epsilon} \ . \end{aligned} \tag{3.15}$$

It is noteworthy that the numerator in the last expression resembles Shannon's entropy in information theory [40], justifying the name of $D_1(\mu)$.

The information dimension is mentioned here just for the sake of completeness. Because the p_i are seldom known when dealing with a finite number of samples, its evaluation remains difficult, except when the p_i are assumed to be equal, meaning that all occupied boxes have the same probability to be visited:

$$\forall i, \, p_i = \frac{1}{N(\epsilon)} \ . \tag{3.16}$$

In this case, it turns out that the information dimension reduces to the capacity dimension:

$$\begin{aligned} d_{\text{inf}} &= \lim_{\epsilon \to 0} \frac{\sum_{i=1}^{N(\epsilon)} N(\epsilon)^{-1} \log N(\epsilon)^{-1}}{\log \epsilon} \\ &= - \lim_{\epsilon \to 0} \frac{\log N(\epsilon)}{\log \epsilon} \\ &= d_{\text{cap}} \ . \end{aligned} \tag{3.17}$$

3.2.4 Correlation dimension

The correlation dimension, introduced by Grassberger and Procaccia [76], corresponds to the case where $q = 2$. The term *correlation* refers to the fact that the probabilities or natural measures p_i are squared. In contrast with both the capacity and information dimensions, the derivation of the correlation dimension is easier starting from the first definition of the q-dimension (Eqs. (3.2) and (3.3)). When the manifold or fractal object is only known by a countable set of points $\mathcal{Y} = \{\mathbf{y}(1), \ldots, \mathbf{y}(n), \ldots, \mathbf{y}(N)\}$, the correlation integral $C_2(\mu, \epsilon)$ (Eq. (3.1) with $q = 2$) can be discretized and replaced with the limit of the correlation sum:

$$C_2(\epsilon) = \lim_{N \to \infty} \frac{1}{N(N-1)} \sum_{\substack{i=1 \\ i<j}}^{N} H(\epsilon - \|\mathbf{y}(i) - \mathbf{y}(j)\|_2) \tag{3.18}$$

$$= P(\|\mathbf{y}(i) - \mathbf{y}(j)\|_2 \leq \epsilon) \ , \tag{3.19}$$

where $H(u)$ is a step function, defined as

$$H(u) = \begin{cases} 0 \text{ if } u < 0 \\ +1 \text{ if } u \geq 0 \end{cases} \quad . \tag{3.20}$$

In Eq. (3.18), H simulates a closed ball of radius ϵ centered on each available point. Then, assuming that the upper and lower limits coincide in Eq. (3.2) and (3.3), the correlation dimension is written as

$$d_{\text{cor}} = D_2 = \lim_{\epsilon \to 0} \frac{\log C_2(\epsilon)}{\log \epsilon} \quad . \tag{3.21}$$

Like the capacity dimension, this discrete formulation of the correlation dimension no longer depends on the natural measures p_i of the support μ. Given a set of points $\mathcal{Y}$, the correlation dimension is easily estimated by the following procedure:

1. Compute the distances for all possible pairs of points $\{\mathbf{y}(i), \mathbf{y}(j)\}$.
2. Determine the proportion of distances that are less than or equal to ϵ.
3. Apply the log function, and divide by $\log \epsilon$.
4. Compute the limit when ϵ tends to zero; this is d_{cor}.

The second step yields only an approximation $\hat{C}_2(\epsilon)$ of $C_2(\epsilon)$ computed with the available N points. But again, the difficult step is the last one. Section 3.2.6 brings some useful hints.

Intuitively, the interpretation of the correlation dimension is very similar to the one associated with the capacity dimension. Instead of adopting a global point of view (the number of boxes that an object or manifold occupies), a closer view is necessary. When looking at the data set on the scale of a single point, $C_2(\epsilon)$ is the number of neighboring points lying closer than a certain threshold ϵ. This number grows as a length for a 1D object, as a surface for a 2D object, as a volume for a 3D object, and so forth. Generalizing for P dimensions gives

$$C_2(\epsilon) \propto \epsilon^P \quad . \tag{3.22}$$

And again,

$$P \propto \frac{\log C_2(\epsilon)}{\log \epsilon} \quad . \tag{3.23}$$

3.2.5 Some inequalities

Considering the q-dimension D_q, if $q_1 < q_2$, then the inequality $D_{q_2} \leq D_{q_1}$ holds [146]. As a consequence, it follows that

$$D_2 \leq D_1 \leq D_0 \ , \qquad \text{i.e.,} \qquad d_{\text{cor}} \leq d_{\text{inf}} \leq d_{\text{cap}} \quad . \tag{3.24}$$

3.2.6 Practical estimation

When the knowledge of a manifold or fractal object is limited to a finite number of points, the capacity and correlation dimensions are much more easily computed than the information dimension. However, the theoretical estimation of their respective formulas includes a limit toward zero, for either the radius of the balls (correlation dimension) or the edge of the boxes (capacity dimension). This is clearly impossible in practice.

Concretely, only values of $N(\epsilon)$ or $\hat{C}_2(\epsilon)$ are available for ϵ ranging between the smallest and largest distances measured in the data set. A trick to circumvent this obstacle consists of applying l'Hospital's rule, knowing that both the numerator and the denominator tend to $-\infty$. For example, in the case of the correlation dimension

$$\begin{aligned} d_{\text{cor}} &= \lim_{\epsilon \to 0} \frac{\log \hat{C}_2(\epsilon)}{\log \epsilon} && (3.25)\\ &= \lim_{\epsilon \to 0} \frac{\partial \log \hat{C}_2(\epsilon)}{\partial \epsilon} \Big/ \frac{\partial \log \epsilon}{\partial \epsilon} && (3.26)\\ &= \lim_{\epsilon \to 0} \frac{\partial \log \hat{C}_2(\epsilon)}{\partial \log \epsilon} && (3.27)\\ &= \lim_{\epsilon_1, \epsilon_2 \to 0} \frac{\log \hat{C}_2(\epsilon_2) - \log \hat{C}_2(\epsilon_1)}{\log \epsilon_2 - \log \epsilon_1} \quad . && (3.28) \end{aligned}$$

According to the literature, the differentiation brings a better estimate when the limit is omitted. Consequently, one defines the scale-dependent correlation dimension as

$$\hat{d}_{\text{cor}}(\epsilon_1, \epsilon_2) = \frac{\log \hat{C}_2(\epsilon_2) - \log \hat{C}_2(\epsilon_1)}{\log \epsilon_2 - \log \epsilon_1} \quad , \qquad (3.29)$$

which is practically computed as the average slope of the curve in a log-log plot of $\hat{C}_2(\epsilon)$ versus ϵ. The values of ϵ_1 and ϵ_2 are set between the minimal and maximal pairwise distances measured in the available data set.

As $\hat{d}_{\text{cor}}$ depends on scale, what are the best values for ϵ_1 and ϵ_2? Since the number of points is finite, choosing small values is hopeless: the estimation gives a dimension near zero; zero is indeed the dimension of isolated points. Similarly, high values for ϵ_1 and ϵ_2 do not make any sense either, since Eq. (3.21) contains a limit toward zero. In this case, the dimension estimate also vanishes since the set of points, seen from a remote point of view, also looks like a single (fuzzy) isolated point. Between these two extreme choices lies the adequate solution. Usually, the best estimate of $\hat{d}_{\text{cor}}$ is obtained in the largest region where the slope of $\hat{C}_2(\epsilon)$ is almost constant in the log-log plot. This region is often called a "plateau".

For example, the scale-dependent correlation dimension $\hat{d}_{\text{cor}}$ can be computed for the coastline of Koch's island (see Fig. 3.2). The data set contains all corners of the seventh iteration of the associated Lindenmayer system

($N(7) = 3 \cdot 4^7 = 49,152$), which is represented in the first plot of Fig. 3.3. (Only corners can be taken into account since sides are indefinitely refined.) The log-log plot of the estimated correlation sum $\hat{C}_2(\epsilon)$ is displayed in the second plot of Fig. 3.3. Obviously, as the data set is generated artificially, the result is nearly perfect: the slope of the curve is almost constant between $\epsilon_1 \approx \exp(-6) = 0.0025$ and $\epsilon_2 \approx \exp(0) = 1$. However, the manual adjustment of a line onto the curve is a tedious task for the user.

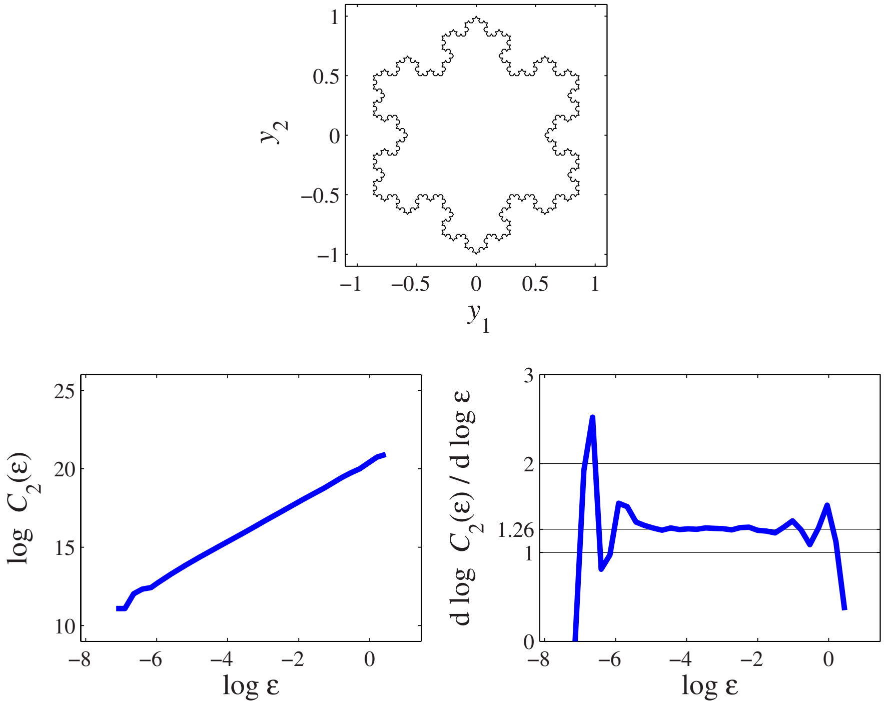


Fig. 3.3. Correlation dimension of Koch's island. The first plot shows the coastline, whose corners are the data set for the estimation of the correlation dimension. The log-log plots of the estimated correlation sum $\hat{C}_2(\epsilon)$ and its numerical derivative are displayed below.

Alternatively, the correlation dimension can be estimated by computing the numerical derivative of $\log \hat{C}_2(\exp \upsilon)$, with $\upsilon = \log \epsilon$:

$$\hat{d}_{\text{cor}} = \frac{d}{d\upsilon} \log \hat{C}_2(\exp \upsilon) \quad . \tag{3.30}$$

This turns out to compute the slope of $\hat{C}_2(\epsilon)$ in a log-log plot.

For any function $f(x)$ known at regularly spaced values of x, the numerical derivative can be computed as a second-order estimate, written as

$$f'(x) = \frac{f(x + \Delta x) - f(x - \Delta x)}{2\Delta x} + \mathcal{O}(\Delta x^3) \tag{3.31}$$

and based on Taylor's polynomial expansion of an infinitely differentiable function $f(x)$. The numerical derivative of $\log \hat{C}_2(\exp v)$ directly yields the dimension for any value of $v = \log \epsilon$; the result is displayed in the third plot of Fig. 3.3 for the coastline of Koch's island. As expected, the estimated correlation dimension is very close to the capacity dimension computed analytically at the end of Subsection 3.2.2.

The use of a numerical derivative is usually criticized because it yields a chopping and changing estimate. Nevertheless, in normal situations, it works rather well, is visually more satisfying, and, last but not least, provides a result that proves a bit less user-dependent. Indeed, the manual adjustment of a line onto a curve is essentially a matter of personal perception.

Finally, the following example illustrates the fact that the estimated correlation dimension depends on the observation scale. The manifold is a spiral, written as

$$\mathbf{y} = \sqrt{x} \begin{bmatrix} \cos(10\pi\sqrt{x}) \\ \sin(10\pi\sqrt{x}) \end{bmatrix} + \mathbf{n} \ , \tag{3.32}$$

where the unique parameter x goes from 0 to 1 and $\mathbf{n}$ is white Gaussian noise with standard deviation 0.005. By construction, this spiral is a 1-manifold embedded in $\mathbb{R}^2$, and a quick look at the first plot of Fig. 3.4 confirms it visually. However, the correlation dimension gives a more contrasted result (second and third plots of Fig. 3.4). In the second plot, from left to right, the correlation is growing constantly, then seems to "slow down" and is finally growing again until it reaches its maximal value. The explanation of this behavior can be found in the third plot, by considering the derivative:

1. For extremely small values of ϵ, the correlation sum remains on the scale of isolated points. This interval is shown as the black box, the same color as the points of the spiral in the first plot. These points are 0-manifold and the estimated dimension is low indeed.
2. For barely larger values of ϵ, the correlation sum measures the dimension of the noise. This interval is shown as a dark gray box, which corresponds to the small square box of the same color in the first plot. As noise occupies all dimensions of space, the dimension is two.
3. For still larger values of ϵ, the correlation sum begins to take into account entire pieces of the spiral curve. Such a piece is shown in the light gray rectangular box in the first plot. On this scale, the spiral is a 1-manifold, as intuitively expected and as confirmed by the estimated correlation dimension.
4. For values of ϵ close to the maximal diameter of the spiral, the correlation dimension encompasses distances across the whole spiral. These values are shown as a white box in the third plot, corresponding to the entire white box surrounding the spiral in the first plot. On this scale the spiral

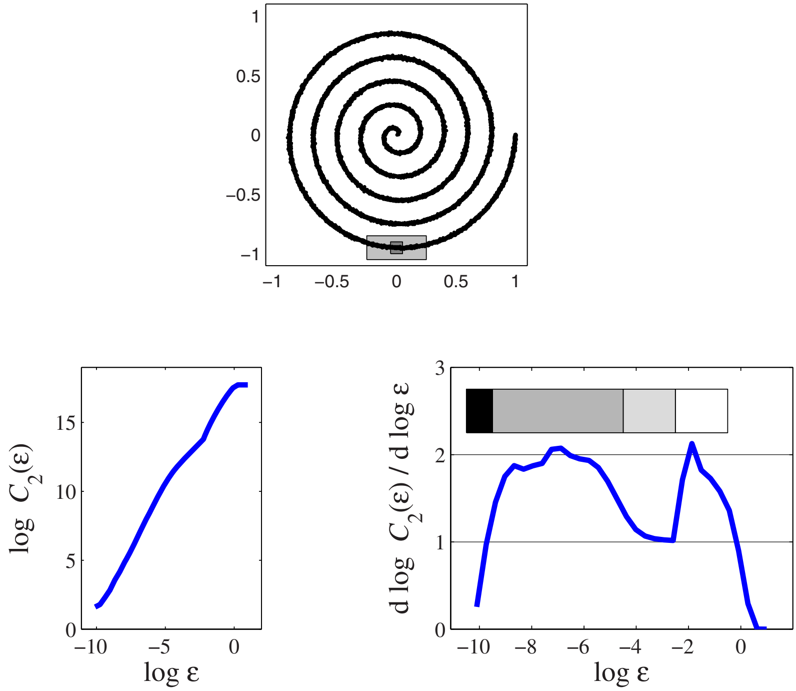


Fig. 3.4. Correlation dimension of a noisy spiral. The first plot shows the data set (10,000 points). The log-log plots of the estimated correlation sum $\hat{C}_2(\epsilon)$ and its numerical derivative are displayed below. The black, dark gray, light gray, and white boxes in the third plot illustrate that the correlation dimension depends on the observation scale. They correspond, respectively, to the scale of the isolated points, the noise, pieces of the spiral curve, and the whole spiral.

appears as a plane with some missing points, and indeed the dimension equals two.

5. For values of ϵ far beyond the diameter, the correlation dimension sees the spiral as a smaller and smaller fuzzy spot. Intuitively, this turns out to zoom out in the first plot of Fig. 3.4. This explains why the dimension vanishes for very large values of ϵ (no box is drawn).

All those variations of the estimated correlation dimension are usually called microscopic effects (1 and 2), lacunarity effects (3), and macroscopic effects (4 and 5) [174]. Other macroscopic effects that are not illustrated here are, for example, side and corner effects. For example, when computing the correlation dimension of a square, the number of points inside a ball of radius ϵ is always proportional to ϵ^2. However, a multiplicative coefficient should be taken into account. Assuming that inside the square this coefficient equals 1, then near a side, it is only 1/2; and near a corner, it further decrease towards 1/4.

Therefore, the estimated dimension not only depends on scale but also on the "location" in space where it is estimated!

3.3 Other dimension estimators

Other methods that are not primarily intended to compute the fractal dimension can though be used to evaluate the dimensionality of a manifold. Principal component analysis, studied in Section 2.4, is the most-known example: this DR method integrates an estimator of the intrinsic dimensionality that is based on the same model. Despite this nice coherence, the model of PCA is linear (see Eq. (2.4)), meaning that the estimator works only for manifolds containing linear dependencies (i.e., linear subspaces). For more complex manifolds, PCA gives at best an estimate of the *global* dimensionality of an object.

For example, PCA estimates the dimension of the spiral in the first plot of Fig. 3.4 as two. In other words, PCA suffers from the first macroscopic effect mentioned at the end of the previous subsection. On the other hand, the correlation dimension succeeds in giving the dimension on all scales. Hence, a promising way to explore is the use of PCA on a *local* scale.

3.3.1 Local methods

The idea behind local methods consists of decomposing the space into small patches, or "space windows", and to consider each of them separately. To some extent, this idea is closely related to the use of boxes and balls in the capacity and correlation dimensions.

The most widely known local method is based on the nonlinear generalization of PCA already sketched in Subsection 2.5.8. Briefly put, the space windows are determined by clustering the data. Usually, this is achieved by vector quantization (see Appendix D). In a few words, vector quantization processes a set of points by replacing it with a smaller set of "representative" points. Usually, the probability distribution function of these points resembles that of the initial data set, but their actual distribution is, of course, much sparser. If each point is mapped to the closest representative point, then the space windows are defined as the subsets of points that are mapped to the same representative point. Next, PCA is carried out locally, on each space window, assuming that the manifold is approximately linear on the scale of a window. Finally, the dimensionality of the manifold is obtained as the average estimate yielded by all local PCAs. Usually, each window is weighted by the number of points it contains before computing the mean.

Moreover, it is noteworthy that not just the mean can be computed: other statistics, like standard deviations or minimal and maximal values, may help to check that the dimensionality remains (nearly) identical over all space windows. Hence, local PCA can detect spatial variations of the intrinsic dimensionality. This is a great difference with other methods, like fractal dimension, that usually assume that dimensionality is a global property of data.

For the noisy spiral of Fig. 3.4, the local PCA approach yields the result shown in Fig. 3.5. The first plot is a copy of the spiral data set, but the boundaries of the space windows are added in gray (70 windows have been built). The second plot shows the fraction of variance spanned by the first principal component of each space window, as a function of the number of space windows. In the third plot, the three curves indicate the dimensionality for three variance thresholds (0.97, 0.98, and 0.99). As can be seen, the dimension given by local PCA is scale-dependent, like the correlation dimension. Actually, the scale is implicitly determined by the number of space windows. If this number is too low, the windows are too large and PCA "sees" the macroscopic structure of the spiral, which is two-dimensional. At nearly 70, the value that corresponds to the number of space windows in the first plot, the size of the windows is close to the optimum and PCA "sees" small pieces of the spiral curve: the dimension is one. If the number of windows further increases, the windows become too small: the noise scale is attained and PCA needs two components to explain the variance.

By comparison with the fractal dimensions like the correlation dimension, the local PCA requires more data samples to yield an accurate estimate. This is because local PCA works by dividing the manifold into nonoverlapping patches. On the contrary, the correlation dimension places a ball on each point of the data set. As a counterpart, local PCA is faster ($\mathcal{O}(N)$) than the correlation dimension ($\mathcal{O}(N^2)$), at least for a single run. Otherwise, if local PCA is repeated for many different numbers of space windows, as in Fig. 3.5, then the computation time grows.

The local PCA approach has been proposed by Kambathla and Leen [101] as a DR method. Because this method does not provide an embedding in a single coordinate system in a natural way, it does not encounter much success, except in data compression. Fukanaga and Olsen [72], on the other hand, followed the same approach more than two decades before Kambathla and Leen in order to estimate the intrinsic dimensionality of data.

3.3.2 Trial and error

Instead of generalizing the use of PCA to nonlinear manifolds by dividing the space into small patches, PCA could be replaced with other DR methods that inherently rely on a nonlinear model. As already mentioned, most DR methods do not integrate a dimensionality estimator as PCA. But on the other hand, some of these methods minimize a reconstruction error E_{codec} (Eq. (2.1)), exactly as PCA does (see Subsection 2.4.2). Actually, the reconstruction error depends on the embedding dimensionality P. If $P = D$, then trivially E_{codec} is exactly zero, since the manifold to be embedded can simply be copied without

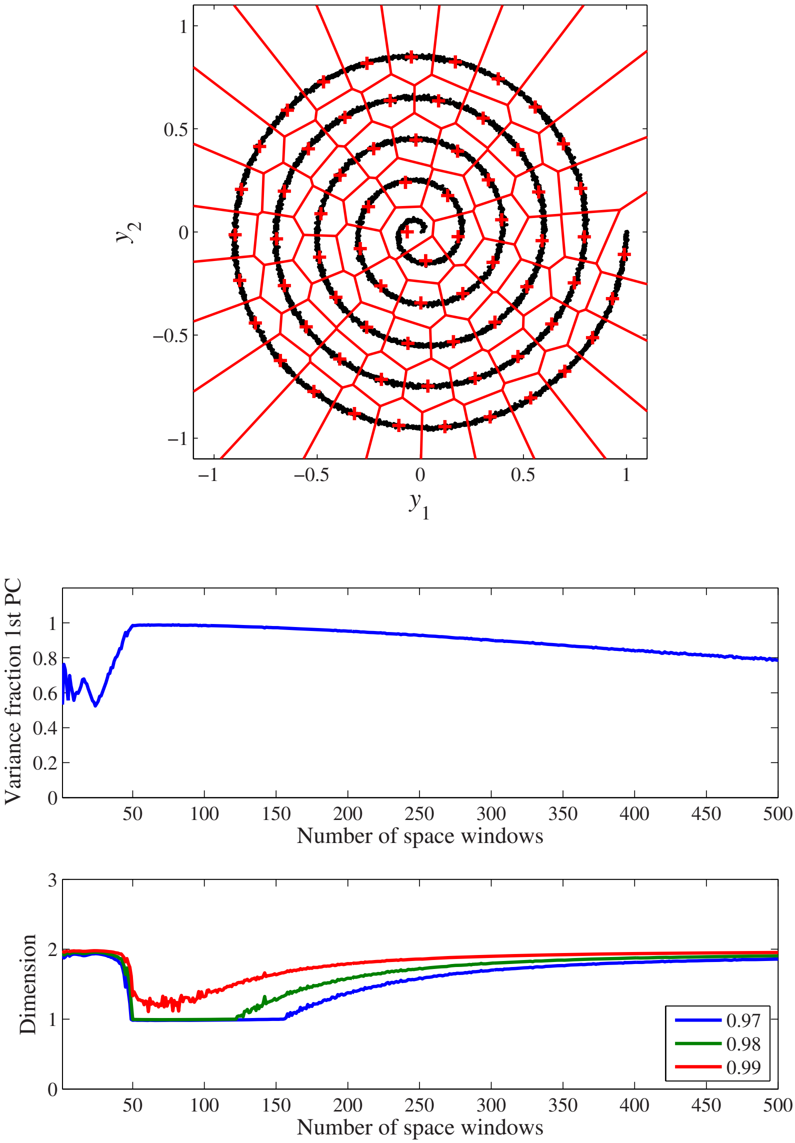


Fig. 3.5. Intrinsic dimensionality of the noisy spiral shown in Fig. 3.4, estimated by local PCA. The first plot shows the spiral again, but the boundaries of the space windows are added in gray (70 windows). The second plot shows the fraction of the total variance spanned by the first principal component of each cluster or space window. This fraction is actually computed as an average for different numbers of windows (in abscissa). The third plot shows the corresponding dimensionality (computed by piecewise linear interpolation) for three variance fractions (0.97, 0.98, and0.99).

any change. If $P = 0$, then the error reaches its maximal value, equal to the global variance ($\mathrm{tr}(\mathbf{C_{yy}})$). For $0 < P < D$, the error varies between these two extrema but cannot be predicted exactly. However, one may expect that the error will remain low if P is greater than the intrinsic dimensionality of the manifold to be embedded. On the contrary, if P goes below the intrinsic dimensionality, the dimensionality reduction may cause a sudden increase in E_{codec}.

With this ideas in mind, the following procedure can yield an estimate of the intrinsic dimensionality:

1. For a manifold embedded in a D-dimensional space, reduce dimensionality successively to $P = 1, 2, \ldots, D$; of course, if some additional information about the manifold is available, the search interval may be smaller.
2. Plot E_{codec} as a function of P.
3. Choose a threshold, and determine the lowest value of P such that E_{codec} goes below it: this is the estimate of the intrinsic dimensionality of the manifold.

The choice of the threshold in the last step is critical since the user determines it arbitrarily. But most of the time the curve E_{codec} versus P is very explicit: an elbow is usually clearly visible when P equals the intrinsic dimensionality. An example is given in the following section.

An additional refinement of the procedure consists of using statistical estimation methods like cross validation or bootstrapping. Instead of computing an embedding for a certain dimensionality only once, those methods repeat the dimensionality reduction on several subsets that are randomly drawn from the available data. This results in a better estimation of the reconstruction errors, and therefore in a more faithful estimation of the dimensionality at the elbow.

The main disadvantage of the procedure, especially with cross validation or bootstrapping, lies in its huge computational requirements. This is particularly annoying when the user is not interested in the embedding, but merely in the value of the intrinsic dimensionality. Moreover, if the DR method is not incremental (i.e., does not produce incremental embeddings, see Subsection 2.5.7), the computation time dramatically increases.

3.4 Comparisons

This section attempts to compare the above-mentioned methods in the case of an artificially generated manifold whose dimensionality is known beforehand. The first and simplest method is PCA; the others are the correlation dimension, the local PCA, and finally the “trial and error” method.

3.4.1 Data Sets

The proposed manifold has already been used in [45]. In a three-dimensional cube ($[-1,+1]^3$), 10 distance sensors are placed at random locations. With these sensors, each position of the cube can be encoded as the vector containing the distances toward the 10 captors. Obviously, these distances are not independent: nonlinear relationships bind them. And, of course, the number of free parameters is actually three, i.e. the dimensionality of the cube where points are picked out.

For the experiments below, the positions of the 10 sensors are

x_1	x_2	x_3
+0.026	+0.241	+0.026
+0.236	+0.193	−0.913
−0.653	+0.969	−0.700
+0.310	+0.094	+0.876
+0.507	+0.756	+0.216
−0.270	−0.978	−0.739
−0.466	−0.574	+0.556
−0.140	−0.502	−0.155
+0.353	−0.281	+0.431
−0.473	+0.993	+0.411

Three data sets are made available for the dimensionality estimation: they contain, respectively, 100, 1000, and 10,000 observations. The three-dimensional points that generated them are uniformly distributed in the three-dimensional cube $[-1,+1]^3$. Once the 10 distances are computed, white Gaussian noise is added, with standard deviation equal to 0.01.

3.4.2 PCA estimator

Figure 3.6 shows the results of PCA applied globally on the three data sets. As can be seen, the number of observations does not greatly influence the results. For the three data sets, the normalized variances vanish starting from the fifth principal component. Clearly, this not a good result. But this overestimation of the intrinsic dimension is not unexpected: PCA works with a linear model, which is unable to cope with the nonlinear dependences hidden in the data sets.

3.4.3 Correlation dimension

The results of the correlation dimension are given in Fig. 3.7. This method is much more sensitive to the number of available observations. For 100 observation, the numerical derivative is chopping and changing, although the right dimensionality can already be guessed. For 1000 and 10,000 observations, the

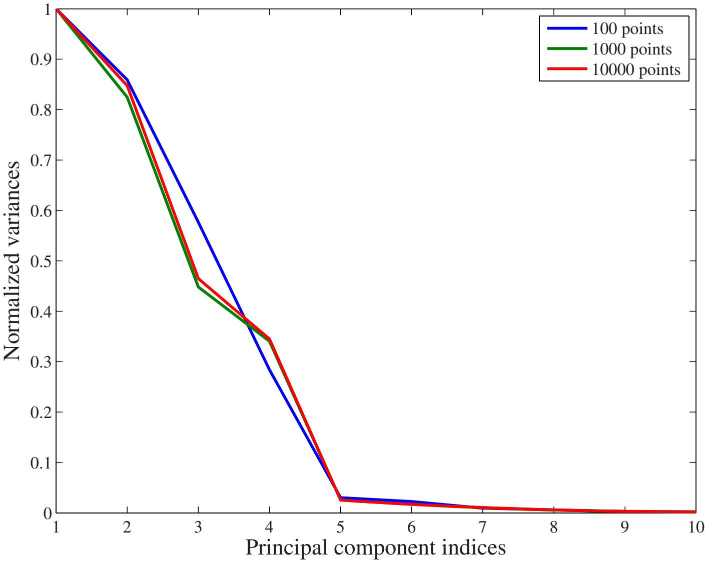


Fig. 3.6. Estimation of the intrinsic dimensionality for the three "sensor" data sets (100, 1000, and 10,000 observations), by using PCA applied globally.

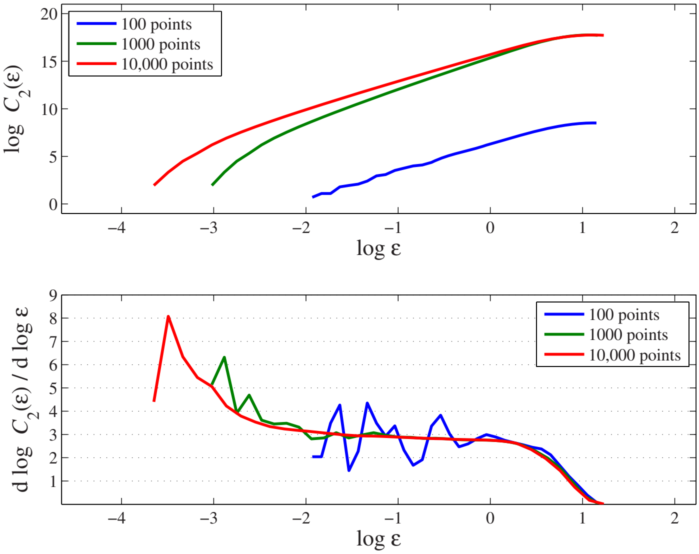


Fig. 3.7. Estimation of the intrinsic dimensionality for the three 'sensor' data sets (100, 1000 and 10000 observations), by using the correlation dimension.

results are nearly perfect: the estimated dimensionality is three, as expected. It can be remarked that the noise dimensionality appears more clearly as the number of observations grows. Nevertheless, even for 10,000 observations, the noise dimensionality remains underestimated. Moreover, edge effects appear for $-1 \leq \log \epsilon \leq 0$: the dimensionality is slightly underestimated.

From the computational point of view, the correlation dimension is much slower than PCA but yields higher quality results.

3.4.4 Local PCA estimator

The results of local PCA are displayed in Fig. 3.8. Actually, only the results for the two largest data sets are shown. For 100 observations, the space windows do not contain enough observations to yield a trustworthy estimate. As the correlation dimension, the local PCA yields the right dimensionality. This can be seen in both plots: the largest three normalized eigenvalues remain high for any number of windows, while the fourth and subsequent ones are negligible. It is noteworthy that for a single window the result of local PCA is trivially the same as for PCA applied globally. But as the number of windows is increasing, the fourth normalized eigenvalue is decreasing slowly. This indicates that the

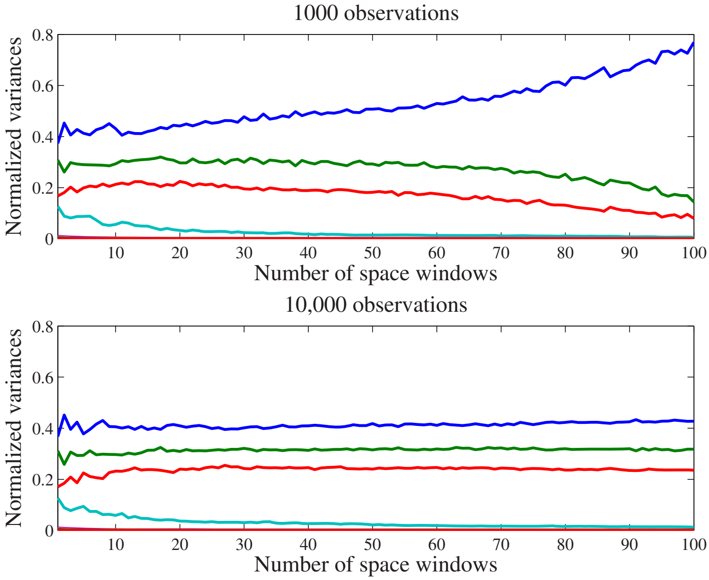


Fig. 3.8. Estimation of the intrinsic dimensionality for the two largest "sensor" data sets (1000 and 10,000 observations) by local PCA. The normalized eigenvalues are shown according to the number of space windows.

division into an increasing number of space windows allows us to capture the nonlinear shape of the underlying manifold. The smaller the windows are, the stronger the assumption that the manifold is locally linear holds. Nevertheless, if the windows become too numerous and too small, PCA is no longer reliable, because the windows do not contain enough points. This is especially visible in the first plot of Fig. 3.8: the second and third eigenvalues are slowly vanishing, whereas the first one is becoming more and more dominant.

Local PCA is obviously much slower than global PCA, but still faster than the correlation dimension, at least if the number of windows does not sweep an interval that is too wide.

3.4.5 Trial and error

For the trial-and-error technique, the chosen DR method is Sammon's nonlinear mapping (see Subsection 4.2.3), using the Euclidean distance. This choice is justified by the fact that the method minimizes an explicit error criterion, called "Sammon's stress". This criterion is not exactly a reconstruction error as defined in Eq. (2.1), but it is closely related to it.

The results are shown only for the two smallest data sets in Fig. 3.9, because the computation time proves too long for 10,000 observations. As can

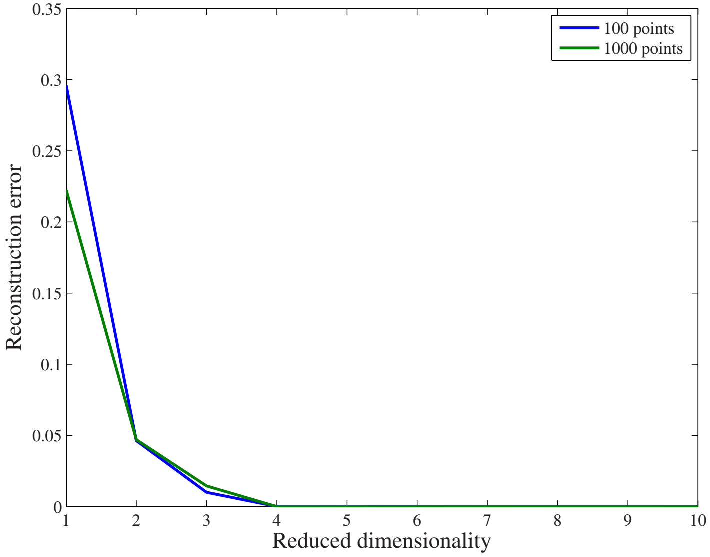


Fig. 3.9. Estimation of the intrinsic dimensionality for the two smallest "sensor" data sets (100 and 1000 observations) by trial and error with Sammon's nonlinear mapping.

be seen, the number of points does not play an important role. At first sight, the estimated dimension is four, since the error is almost zero starting from this number. Actually, the DR method slightly overestimates the dimensionality, like PCA applied globally. Although the method relies on a nonlinear model, the manifold may still be too curved to achieve a perfect embedding in a space having the same dimension as the exact manifold dimensionality. One or two extra dimensions are welcome to embed the manifold with some more freedom. This explains why the overestimation observed for PCA does not disappear but is only attenuated when switching to an NLDR method.

3.4.6 Concluding remarks

Among the four compared methods, PCA applied globally on the whole data set undoubtly remains the simplest and fastest one. Unfortunately, its results are not very convincing: the dimension is almost always overestimated if data do not perfectly fit the PCA model. Since the PCA criterion can be seen as a reconstruction error, PCA is actually a particular case of ‘trial and error’ method. And because PCA is an incremental method, embeddings for all dimensions— and the corresponding errors —are computed at once.

When replacing PCA with a method relying on a nonlinear model, these nice properties are often lost. In the case of Sammon’s nonlinear mapping, not only is the method much slower due to its more complex model, but additionally the method is no longer incremental! Put together, these two drawbacks make the trial-and-error method very slow. Furthermore, the overestimation that was observed with PCA does not disappear totally.

The use of local PCA, on the other hand, seems to be a good tradeoff. The method keeps all advantages of PCA and combines them with the ability to handle nonlinear manifolds. Local PCA runs fast if the number of windows does not sweep a wide interval. But more importantly, local PCA has given the right dimensionality for the studied data sets, along with the correlation dimension.

Eventually, the correlation dimension clearly appears as the best method to estimate the intrinsic dimensionality. It is not the fastest of the four methods, but its results are the best and most detailed ones, giving the dimension on all scales.

4

Distance Preservation

Overview. This chapter deals with methods that reduce the dimensionality of data by using distance preservation as the criterion. In the ideal case, the preservation of the pairwise distances measured in a data set ensures that the low-dimensional embedding inherits the main geometric properties of data, like the global shape or the local neighborhood relationships. Unfortunately, in the nonlinear case, distances cannot be perfectly preserved. The chapter reviews various methods that attempt to overcome this difficulty. These methods use different kinds of distances (mainly spatial or graph distances); they also rely on different algorithms or optimization procedures to determine the embedding.

4.1 State-of-the-art

Historically, distance preservation has been the first criterion used to achieve dimensionality reduction in a nonlinear way. In the linear case, simple criteria like maximizing the variance preservation or minimizing the reconstruction error, combined with a basic linear model, lead to robust methods like PCA. In the nonlinear case however, the use of the same simple criteria requires the definition of more complex data models. Unfortunately, the definition of a generative model in the nonlinear case proves very difficult: there are many different ways to model nonlinear manifolds, whereas there are only a few (equivalent) ways to define a hyperplane.

In this context, distance preservation appears as a nongenerative way to perform dimensionality reduction. The criterion does not need any explicit model: no assumption is made about the mapping from the latent variables to the observed ones. Intuitively, the motivation behind distance preservation is that any manifold can be fully described by pairwise distances. Hence, if a low-dimensional representation can be built in such a way that the initial distances are reproduced, then the dimensionality reduction is successful: the information content conveyed by the manifold— its geometrical structure —is preserved. It is clear that if close points are kept close, and if far points remain far, then the initial manifold and its low-dimensional embedding share the same shape.

The next three sections of this chapter review some of the best-known DR methods that use the principle of distance preservation; they are called distance-preserving methods in short. Each of the three sections focuses on a particular type of distance. Section 4.2 introduces the most common distance measures, like the Euclidean one, and methods that are based on it. Next, Section 4.3 describes geodesic and graph distances, which have peaked much interest in the last few years. Finally, Section 4.4 deals with even more exotic distance measures that are related to kernel functions and kernel learning.

Additional examples and comparisons between the described methods can be found in Chapter 6.

4.2 Spatial distances

Spatial distances, like the Euclidean distance, are the most intuitive and natural way to measure distances in the real (Euclidean) world. The adjective *spatial* indicates that these metrics compute the distance separating two points of the space, without regards to any other information like the presence of a submanifold: only the coordinates of the two points matter. Although these metrics are probably not the most appropriate for dimensionality reduction (see Section 4.3.1), their simplicity makes them very appealing. Subsection 4.2.1 introduces some facts about and definitions of distances, norms, and scalar products; then it goes on to describe the methods that reduce dimensionality by using spatial distances.

4.2.1 Metric space, distances, norms and scalar product

A space $\mathcal{Y}$ with a distance function $d(\mathbf{a},\mathbf{b})$ between two points $\mathbf{a},\mathbf{b} \in \mathcal{Y}$ is said to be a *metric space* if the distance function respects the following axioms:

- **Nondegeneracy.** For any points $\mathbf{a}$ and $\mathbf{b}$ in the space $d(\mathbf{a},\mathbf{b}) = 0$ if and only if $\mathbf{a} = \mathbf{b}$.
- **Triangular inequality.** For any points $\mathbf{a}$, $\mathbf{b}$ and $\mathbf{c}$ in the space $d(\mathbf{a},\mathbf{b}) \leq d(\mathbf{c},\mathbf{a}) + d(\mathbf{c},\mathbf{b})$.

Other usual and desired properties for the distance function, like the symmetry and the nonnegativity, trivially follow from these two axioms. This comes from the specific formulation of the triangular inequality. If the latter is defined as $d(\mathbf{a},\mathbf{b}) \leq d(\mathbf{a},\mathbf{c}) + d(\mathbf{c},\mathbf{b})$, then the symmetry and nonnegativity must be added as axioms. But with the first definition, they can be derived as follows:

- **Nonnegativity.** If $\mathbf{a} = \mathbf{b}$, the triangular inequality becomes

$$d(\mathbf{a}, \mathbf{a}) \leq d(\mathbf{c}, \mathbf{a}) + d(\mathbf{c}, \mathbf{a}) = 2d(\mathbf{c}, \mathbf{a}) \ . \tag{4.1}$$

  Simplifying with the help of nondegeneracy results in:

$$0 \leq d(\mathbf{c}, \mathbf{a}) \ . \tag{4.2}$$

- **Symmetry.** If $\mathbf{a} = \mathbf{u}$ and $\mathbf{b} = \mathbf{c} = \mathbf{v}$, the triangular inequality becomes

$$d(\mathbf{u}, \mathbf{v}) \leq d(\mathbf{v}, \mathbf{u}) + d(\mathbf{v}, \mathbf{v}) = d(\mathbf{v}, \mathbf{u}) \ , \tag{4.3}$$

  by the use of nondegeneracy. Similarly, if $\mathbf{a} = \mathbf{v}$ and $\mathbf{b} = \mathbf{c} = \mathbf{u}$, then

$$d(\mathbf{v}, \mathbf{u}) \leq d(\mathbf{u}, \mathbf{v}) + d(\mathbf{u}, \mathbf{u}) = d(\mathbf{u}, \mathbf{v}) \ . \tag{4.4}$$

  The conjunction of both inequalities forces the equality $d(\mathbf{u}, \mathbf{v}) = d(\mathbf{v}, \mathbf{u})$.

In the usual Cartesian vector space $\mathbb{R}^D$, the most-used distance functions are derived from the Minkowski norm. Actually, the pth-order Minkowski norm of point $\mathbf{a} = [a_1, \ldots, a_k, \ldots, a_D]^T$, also called the L_p norm and noted $\|\mathbf{a}\|_p$, is a simple function of the coordinates of $\mathbf{a}$:

$$\|\mathbf{a}\|_p = \sqrt[p]{\sum_{k=1}^{D} |a_k|^p} \ , \tag{4.5}$$

where $p \in \mathbb{N}_0$. A distance function that respects the above-mentioned axioms is obtained by measuring the norm of the difference between two points:

$$d(\mathbf{a}, \mathbf{b}) = \|\mathbf{a} - \mathbf{b}\|_p = \|\mathbf{b} - \mathbf{a}\|_p \ . \tag{4.6}$$

When using the Minkowski distance, some values of p are chosen preferentially because they lead to nice geometrical or mathematical properties:

- The maximum distance ($p = \infty$):

$$\|\mathbf{a} - \mathbf{b}\|_\infty = \max_{1 \leq k \leq D} |a_k - b_k| \ , \tag{4.7}$$

  also called the dominance distance, because when $p \to \infty$, all summed terms in Eq. (4.5) become negligible, except the largest one.
- The city-block distance ($p = 1$):

$$\|\mathbf{a} - \mathbf{b}\|_1 = \sum_{k=1}^{D} |a_k - b_k| \ , \tag{4.8}$$

  also called the Manhattan distance because from a geometrical point of view, the measurement of the distance resembles driving a taxi in an American city divided into regular rectangular blocks.

- The Euclidean distance ($p = 2$):

$$\|\mathbf{a} - \mathbf{b}\|_2 = \sqrt{\sum_{k=1}^{D} (a_k - b_k)^2} \ , \tag{4.9}$$

  proves to be the most natural and intuitive distance measure in the real world. The Euclidean distance also has particularly appealing mathematical properties (invariance with respect to rotations, etc.).

Among the three above-mentioned possibilities, the Euclidean distance is the most widely used one, not only because of its natural interpretation in the physical world, but also because of its simplicity. For example, the partial derivative along a component a_k of $\mathbf{a}$ is simply

$$\frac{\partial d(\mathbf{a}, \mathbf{b})}{\partial a_k} = \frac{a_k - b_k}{d(\mathbf{a}, \mathbf{b})} = -\frac{\partial d(\mathbf{a}, \mathbf{b})}{\partial b_k} \ , \tag{4.10}$$

or, written directly in a vector form, is

$$\frac{\partial d(\mathbf{a}, \mathbf{b})}{\partial \mathbf{a}} = \frac{\mathbf{a} - \mathbf{b}}{d(\mathbf{a}, \mathbf{b})} = -\frac{\partial d(\mathbf{a}, \mathbf{b})}{\partial \mathbf{b}} \ . \tag{4.11}$$

Another advantage of the Euclidean distance comes from the alternative definition of the Euclidean norm by means of the scalar product:

$$\|\mathbf{a}\|_2 = \sqrt{\langle \mathbf{a} \cdot \mathbf{a} \rangle} \ , \tag{4.12}$$

where the notation $\langle \mathbf{a} \cdot \mathbf{b} \rangle$ indicates the scalar product between vector $\mathbf{a}$ and $\mathbf{b}$. Formally, the scalar or dot product is defined as

$$\langle \mathbf{a} \cdot \mathbf{b} \rangle = \mathbf{a}^T \mathbf{b} = \sum_{k=1}^{D} a_k b_k \ . \tag{4.13}$$

Two important properties of the scalar product are:

- commutativity:

$$\langle \mathbf{a} \cdot \mathbf{b} \rangle = \langle \mathbf{b} \cdot \mathbf{a} \rangle \ . \tag{4.14}$$

- left and right distributivity:

$$\langle \mathbf{a} \cdot (\mathbf{b} + \mathbf{c}) \rangle = \langle \mathbf{a} \cdot \mathbf{b} \rangle + \langle \mathbf{a} \cdot \mathbf{c} \rangle \ , \tag{4.15}$$

$$\langle (\mathbf{a} + \mathbf{b}) \cdot \mathbf{c} \rangle = \langle \mathbf{a} \cdot \mathbf{c} \rangle + \langle \mathbf{b} \cdot \mathbf{c} \rangle \ . \tag{4.16}$$

Finally, the overview of the classical distance functions would not be complete without mentioning the Mahalanobis distance, a straight generalization of the Euclidean distance. The Mahalanobis norm is defined as

$$\|\mathbf{a}\|_{\text{Mahalanobis}} = \mathbf{a}^T \mathbf{M}^{-1} \mathbf{a} \ , \tag{4.17}$$

where $\mathbf{M}$ is often chosen as the covariance matrix $\mathbf{C}_{\mathbf{aa}} = E\{\mathbf{aa}^T\}$. Obviously, the Euclidean distance corresponds to the particular case where $\mathbf{M}$ is the identity matrix. Intuitively, the equicontours are circles for the Euclidean distance and ellipses for the Mahalanobis distance.

Most distance-preserving NLDR algorithms that are described in the forthcoming sections involve pairwise distances. Assuming that the finite set of indexed points, denoted as

$$\mathcal{Y} = \{\mathbf{y}(1), \ldots, \mathbf{y}(i), \ldots, \mathbf{y}(j), \ldots, \mathbf{y}(N)\} \ , \tag{4.18}$$

is available, then the distance between the two points $\mathbf{y}(i)$ and $\mathbf{y}(j)$, normally written as $d(\mathbf{y}(i), \mathbf{y}(j))$, can be shortened and noted as $d_{\mathbf{y}}(i, j)$.

4.2.2 Multidimensional scaling

The term *multidimensional scaling* (MDS) actually hides a family of methods rather than a single well-defined procedure. *Scaling* refers to methods that construct a configuration of points in a target metric space from information about interpoint distances, and MDS is scaling when the target space is Euclidean [43]. Historically, the first major steps were made by Young and Householder in 1938 [208] and then by Torgerson [182] in 1952, who proposed the purely Euclidean model of metric MDS. The next breakthrough was accomplished a few years later by Shepard in 1962 [171] and by Kruskal in 1964 [108], who elaborated methods for nonmetric MDS, focusing on rank information instead of interpoint distances.

Actually, MDS has been widely used and developed in human sciences like sociology, anthropology, economy, and also particularly in a subfield of psychology called psychometrics. In the latter domain, MDS is used as a tool for the geometrical representation of concepts, objects, or opinions. Classically, people are asked to give a quantitative separation between the concepts: for each concept, they have either to place all the others as points in a continuous interval or to rank them by order of similarity. The first approach is typical of metric MDS and the second of nonmetric MDS. In both cases, each object is characterized by either distances between or similarities to the others. Intuitively, the notion of distance, or dissimilarity, is very easy to understand: the dissimilarity is zero for identical objects and grows as they become increasingly different from each other. Conversely, similarity is high for nearly identical objects and decreases as differences appear. More formally, in the Euclidean case, a distance between two points $\mathbf{y}(i)$ and $\mathbf{y}(j)$ is related to the norm of their difference:

$$d(\mathbf{y}(i), \mathbf{y}(j)) = \sqrt{\langle (\mathbf{y}(i) - \mathbf{y}(j)) \cdot (\mathbf{y}(i) - \mathbf{y}(j)) \rangle} \ , \tag{4.19}$$

whereas an example of a similarity measure can be written using the inverse of the distance. As expected, the similarity then vanishes as the distance grows, and conversely the similarity tends to infinity for small distances.

When used in psychometrics, two different kinds of MDS can be distinguished. Indeed, when MDS has to process the results of a public opinion poll or survey, each individual surveyed provides similarities for all pairs of concepts. Consequently, the data are not stored in a matrix but in a three-dimensional tensor. The so-called three-way MDS is designed to analyze such a data set. However, it will be not studied hereafter, and the emphasis is put on two-way MDS, for which only one similarity value is attributed for each pair of objects.

The remainder of this section describes the original method, the oldest one, often called classical metric multidimensional scaling, which gave rise to many variants.

Embedding of the data set

Actually, classical metric MDS is not a true distance-preserving method. In its classical version, metric MDS preserves pairwise scalar products instead of pairwise distances (both are closely related, however, as will become clear farther ahead). Moreover, classical metric MDS cannot achieve dimensionality reduction in a nonlinear way either. Nevertheless, as it can be considered to be the antecedent of all nonlinear distance-preserving methods, its place in this chapter is fully justified.

Like PCA, metric MDS relies on a simple generative model. More precisely, only an orthogonal axis change separates the observed variables in $\mathbf{y}$ and the latent ones, stored in $\mathbf{x}$:

$$\mathbf{y} = \mathbf{W}\mathbf{x} \ , \tag{4.20}$$

where the components of $\mathbf{x}$ are independent or at least uncorrelated and $\mathbf{W}$ is a D-by-P matrix such that $\mathbf{W}^T\mathbf{W} = \mathbf{I}_P$. Both the observed and latent variables are assumed to be centered.

For a finite set of N points, written in matrix form as

$$\mathbf{Y} = [\ldots, \mathbf{y}(i), \ldots, \mathbf{y}(j), \ldots] \ , \tag{4.21}$$

a short-hand notation may be given for the scalar product between vectors $\mathbf{y}(i)$ and $\mathbf{y}(j)$:

$$s_{\mathbf{y}}(i,j) = s(\mathbf{y}(i), \mathbf{y}(j)) = \langle \mathbf{y}(i) \cdot \mathbf{y}(j) \rangle \ , \tag{4.22}$$

as has been done for distances. Then it can be written that

$$\begin{aligned} \mathbf{S} = [s_{\mathbf{y}}(i,j)]_{1 \le i,j \le N} &= \mathbf{Y}^T\mathbf{Y} && (4.23)\\ &= (\mathbf{W}\mathbf{X})^T(\mathbf{W}\mathbf{X}) && (4.24)\\ &= \mathbf{X}^T\mathbf{W}^T\mathbf{W}\mathbf{X} && (4.25)\\ &= \mathbf{X}^T\mathbf{X} \ . && (4.26) \end{aligned}$$

Usually, both $\mathbf{Y}$ and $\mathbf{X}$ are unknown; only the matrix of pairwise scalar products $\mathbf{S}$, called the Gram matrix, is given. As can be seen, the values of the latent variables can be found trivially by computing the eigenvalue decomposition (see Appendix A.2) of the Gram matrix $\mathbf{S}$:

$$\begin{aligned}\mathbf{S} &= \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T && (4.27)\\ &= (\mathbf{U}\boldsymbol{\Lambda}^{1/2})(\boldsymbol{\Lambda}^{1/2}\mathbf{U}^T) && (4.28)\\ &= (\boldsymbol{\Lambda}^{1/2}\mathbf{U}^T)^T(\boldsymbol{\Lambda}^{1/2}\mathbf{U}^T) \ , && (4.29)\end{aligned}$$

where $\mathbf{U}$ is an N-by-N orthonormal matrix and $\boldsymbol{\Lambda}$ is an N-by-N diagonal matrix containing the eigenvalues. (The reason why $\mathbf{U}$ is used instead of $\mathbf{V}$ like in Appendix A.2 will become clear farther ahead. Moreover, it is noteworthy that as $\mathbf{S}$ is the Gram matrix of the centered data, at most D eigenvalues are strictly positive while others are zero in $\boldsymbol{\Lambda}$.) If the eigenvalues are sorted in descending order, then the estimated P-dimensional latent variables can be computed as the product

$$\hat{\mathbf{X}} = \mathbf{I}_{P\times N}\boldsymbol{\Lambda}^{1/2}\mathbf{U}^T \ . \qquad (4.30)$$

Starting from this solution, the equivalence between metric MDS and PCA can easily be demonstrated.

Actually, metric MDS and PCA give the same solution. To demonstrate it, the data coordinates $\mathbf{Y}$ are assumed to be known— this is mandatory for PCA, but not for metric MDS —and centered. Moreover, the singular value decomposition of $\mathbf{Y}$ is written as $\mathbf{Y} = \mathbf{V}\boldsymbol{\Sigma}\mathbf{U}^T$ (see Appendix A.1). On one hand, PCA decomposes the covariance matrix, which is proportional to $\mathbf{Y}\mathbf{Y}^T$, into eigenvectors and eigenvalues:

$$\hat{\mathbf{C}}_{\mathbf{yy}} \propto \mathbf{Y}\mathbf{Y}^T = \mathbf{V}\boldsymbol{\Sigma}^T\mathbf{U}^T\mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T = \mathbf{V}\boldsymbol{\Sigma}\boldsymbol{\Sigma}^T\mathbf{V}^T = \mathbf{V}\boldsymbol{\Lambda}_{\text{PCA}}\mathbf{V}^T \ , \qquad (4.31)$$

where the division by N is intentionally omitted in the covariance, and $\boldsymbol{\Lambda}_{\text{PCA}} = \boldsymbol{\Sigma}\boldsymbol{\Sigma}^T$. The solution is $\hat{\mathbf{X}}_{\text{PCA}} = \mathbf{I}_{P\times D}\mathbf{V}^T\mathbf{Y}$ (see Eq. (2.19) in Subsection 2.4.2). On the other hand, metric MDS decomposes the Gram matrix into eigenvectors and eigenvalues:

$$\mathbf{S} = \mathbf{Y}^T\mathbf{Y} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^T\mathbf{V}\boldsymbol{\Sigma}\mathbf{U}^T = \mathbf{U}\boldsymbol{\Sigma}^T\boldsymbol{\Sigma}\mathbf{U}^T = \mathbf{U}\boldsymbol{\Lambda}_{\text{MDS}}\mathbf{U}^T \ , \qquad (4.32)$$

where $\boldsymbol{\Lambda}_{\text{MDS}} = \boldsymbol{\Sigma}^T\boldsymbol{\Sigma}$. The solution is $\hat{\mathbf{X}}_{\text{MDS}} = \mathbf{I}_{P\times N}\boldsymbol{\Lambda}_{\text{MDS}}^{1/2}\mathbf{U}^T$. By equating both solutions and by again using the singular value decomposition of $\mathbf{Y}$:

$$\begin{aligned}\hat{\mathbf{X}}_{\text{PCA}} &= \hat{\mathbf{X}}_{\text{MDS}} && (4.33)\\ \mathbf{I}_{P\times D}\mathbf{V}^T\mathbf{Y} &= \mathbf{I}_{P\times N}\boldsymbol{\Lambda}_{\text{MDS}}^{1/2}\mathbf{U}^T && (4.34)\\ \mathbf{I}_{P\times D}\mathbf{V}^T\mathbf{V}\boldsymbol{\Sigma}\mathbf{U}^T &= \mathbf{I}_{P\times N}(\boldsymbol{\Sigma}^T\boldsymbol{\Sigma})^{1/2}\mathbf{U}^T && (4.35)\\ \mathbf{I}_{P\times D}\boldsymbol{\Sigma}\mathbf{U}^T &= \mathbf{I}_{P\times D}\boldsymbol{\Sigma}\mathbf{U}^T \ . && (4.36)\end{aligned}$$

By the way, this proves that PCA and metric MDS minimize the same criterion. In the case of metric MDS, it can be rewritten as

$$E_{\mathrm{MDS}} = \sum_{i,j=1}^{N} \left(s_{\mathbf{y}}(i,j) - \langle \hat{\mathbf{x}}(i) \cdot \hat{\mathbf{x}}(j) \rangle\right)^2 \quad . \tag{4.37}$$

The equivalence between the two methods may be an advantage in some situations. For example, when data consist of distances or similarities, the absence of the coordinates does not prevent us from applying PCA: it suffices to replace PCA with metric MDS. On the other hand, when the coordinates are known, the equivalence is also very useful when the size of the data matrix $\mathbf{Y}$ becomes problematic. If data are not too high-dimensional but the number of points is huge, PCA spends fewer memory resources than MDS since the product $\mathbf{Y}\mathbf{Y}^T$ has a smaller size than $\mathbf{Y}^T\mathbf{Y}$. By contrast, MDS is better when the dimensionality is very high but the number of points rather low. An intermediate solution consists of using the SVD of $\mathbf{Y}$ in both cases.

Until now, it has been assumed that data are known by either coordinates (stored in $\mathbf{Y}$) or scalar products (stored in $\mathbf{S}$). What can be done when Euclidean distances are given instead of scalar products, which often happens?

In that case, distances have to be converted into scalar products before applying metric MDS. For this purpose, it is assumed that pairwise distances are squared and stored in an N-by-N matrix $\mathbf{D}$:

$$\mathbf{D} = [d_{\mathbf{y}}^2(i,j)]_{1 \leq i,j \leq N} \quad . \tag{4.38}$$

According to Section 4.2, the Euclidean distance can be defined as a scalar product:

$$\begin{aligned}
d_{\mathbf{y}}^2(i,j) &= \|\mathbf{y}(i) - \mathbf{y}(j)\|_2^2 && (4.39)\\
&= \langle \mathbf{y}(i) - \mathbf{y}(j) \cdot \mathbf{y}(i) - \mathbf{y}(j) \rangle && (4.40)\\
&= \langle \mathbf{y}(i) \cdot \mathbf{y}(i) \rangle - 2\langle \mathbf{y}(i) \cdot \mathbf{y}(j) \rangle + \langle \mathbf{y}(j) \cdot \mathbf{y}(j) \rangle && (4.41)\\
&= s_{\mathbf{y}}(i,i) - 2s_{\mathbf{y}}(i,j) + s_{\mathbf{y}}(j,j) \quad , && (4.42)
\end{aligned}$$

where $\mathbf{y}$ is assumed to be centered. Of course, this assumption does not influence the distances, which are invariant to translation:

$$\|\mathbf{y}(i) - \mathbf{y}(j)\|_2^2 = \|(\mathbf{y}(i) - \mathbf{z}) - (\mathbf{y}(j) - \mathbf{z})\|_2^2 \quad , \tag{4.43}$$

where $\mathbf{z}$ is the translation vector. According to Eq. (4.42), the scalar products can be computed as

$$s_{\mathbf{y}}(i,j) = -\frac{1}{2}(d_{\mathbf{y}}^2(i,j) - \langle \mathbf{y}(i) \cdot \mathbf{y}(i) \rangle - \langle \mathbf{y}(j) \cdot \mathbf{y}(j) \rangle) \quad . \tag{4.44}$$

Unfortunately, the coordinates $\mathbf{y}$ are usually unknown. Nevertheless, the two subtractions in the right-hand side of Eq. (4.44) can be achieved in an implicit way by an operation called “double centering” of $\mathbf{D}$. It simply consists of subtracting from each entry of $\mathbf{D}$ the mean of the corresponding row and the mean of the corresponding column, and adding back the mean of all entries. In matrix form, this can be written as

$$\mathbf{S} = -\frac{1}{2}(\mathbf{D} - \frac{1}{N}\mathbf{D}\mathbf{1}_N\mathbf{1}_N^T - \frac{1}{N}\mathbf{1}_N\mathbf{1}_N^T\mathbf{D} + \frac{1}{N^2}\mathbf{1}_N\mathbf{1}_N^T\mathbf{D}\mathbf{1}_N\mathbf{1}_N^T) \ . \qquad (4.45)$$

Using the properties of the scalar product, knowing that data are centered, and denoting by μ the mean operator, we find that the mean of the ith row of $\mathbf{D}$ is

$$\begin{aligned}\mu_j(d_{\mathbf{y}}^2(i,j)) &= \mu_j(\langle \mathbf{y}(i) - \mathbf{y}(j) \cdot \mathbf{y}(i) - \mathbf{y}(j)\rangle)\\ &= \mu_j(\langle \mathbf{y}(i) \cdot \mathbf{y}(i)\rangle - 2\langle \mathbf{y}(i) \cdot \mathbf{y}(j)\rangle + \langle \mathbf{y}(j) \cdot \mathbf{y}(j)\rangle)\\ &= \langle \mathbf{y}(i) \cdot \mathbf{y}(i)\rangle - 2\langle \mathbf{y}(i) \cdot \mu_j(\mathbf{y}(j))\rangle + \mu_j(\langle \mathbf{y}(j) \cdot \mathbf{y}(j)\rangle)\\ &= \langle \mathbf{y}(i) \cdot \mathbf{y}(i)\rangle - 2\langle \mathbf{y}(i) \cdot \mathbf{0}\rangle + \mu_j(\langle \mathbf{y}(j) \cdot \mathbf{y}(j)\rangle)\\ &= \langle \mathbf{y}(i) \cdot \mathbf{y}(i)\rangle + \mu_j(\langle \mathbf{y}(j) \cdot \mathbf{y}(j)\rangle) \ . \qquad (4.46)\end{aligned}$$

Similarly, by symmetry of $\mathbf{D}$, the mean of the jth column of $\mathbf{D}$ is

$$\mu_i(d_{\mathbf{y}}^2(i,j)) = \mu_i(\langle \mathbf{y}(i) \cdot \mathbf{y}(i)\rangle) + \langle \mathbf{y}(j) \cdot \mathbf{y}(j)\rangle \ . \qquad (4.47)$$

The mean of all entries of $\mathbf{D}^2$ is

$$\begin{aligned}\mu_{i,j}(d_{\mathbf{y}}^2(i,j)) &= \mu_{i,j}(\langle \mathbf{y}(i) - \mathbf{y}(j) \cdot \mathbf{y}(i) - \mathbf{y}(j)\rangle)\\ &= \mu_{i,j}(\langle \mathbf{y}(i) \cdot \mathbf{y}(i)\rangle) - 2\mu_{i,j}(\langle \mathbf{y}(i) \cdot \mathbf{y}(j)\rangle) + \mu_{i,j}(\langle \mathbf{y}(j) \cdot \mathbf{y}(j)\rangle)\\ &= \mu_i(\langle \mathbf{y}(i) \cdot \mathbf{y}(i)\rangle) - 2\mu_i(\langle \mathbf{y}(i) \cdot \mu_j(\mathbf{y}(j))\rangle) + \mu_j(\langle \mathbf{y}(j) \cdot \mathbf{y}(j)\rangle)\\ &= \mu_i(\langle \mathbf{y}(i) \cdot \mathbf{y}(i)\rangle) - 2\mu_i(\langle \mathbf{y}(i) \cdot \mathbf{0}\rangle) + \mu_j(\langle \mathbf{y}(j) \cdot \mathbf{y}(j)\rangle)\\ &= \mu_i(\langle \mathbf{y}(i) \cdot \mathbf{y}(i)\rangle) + \mu_j(\langle \mathbf{y}(j) \cdot \mathbf{y}(j)\rangle) \ . \qquad (4.48)\end{aligned}$$

Clearly, the two last unknown terms in Eq. (4.44) are equal to the sum of Eq. (4.46) and Eq. (4.47), minus Eq. (4.48):

$$s_{\mathbf{y}}(i,j) = -\frac{1}{2}(d_{\mathbf{y}}^2(i,j) - \mu_j(d_{\mathbf{y}}^2(i,j)) - \mu_i(d_{\mathbf{y}}^2(i,j)) + \mu_{i,j}(d_{\mathbf{y}}^2(i,j))) \ . \qquad (4.49)$$

The algorithm that achieves MDS is summarized in Fig. 4.1. In this algorithm, it is noteworthy that, due to symmetry, the third term in the right-hand side of Eq. (4.45) is the transpose of the second one, which is in turn a subfactor of the fourth one. Similarly, the product in the last step of the algorithm can be computed more efficiently by directly removing the unnecessary rows/columns of $\mathbf{U}$ and $\mathbf{\Lambda}$. The algorithm in Fig. 4.1 can easily be implemented in less than 10 lines in MATLAB®. A C++ implementation can also be downloaded from `http://www.ucl.ac.be/mlg/`. The only parameter of metric MDS is the embedding dimension P. Computing the pairwise distances requires $\mathcal{O}(N^2)$ memory entries and $\mathcal{O}(N^2 D)$ operations. Actually, time and space complexities of metric MDS are directly related to those of an EVD. Computing all eigenvalues and eigenvectors of an N-by-N nonsparse matrix typically demands at most $\mathcal{O}(N^3)$ operations, depending on the implementation.

1. If available data consist of vectors gathered in $\mathbf{Y}$, then center them, compute the pairwise scalar products $\mathbf{S} = \mathbf{Y}^T\mathbf{Y}$, and go to step 3.
2. If available data consist of pairwise Euclidean distances, transform them into scalar products:
   - Square the distances and build $\mathbf{D}$.
   - Perform the double centering of $\mathbf{D}$, according to Eq. (4.45); this yields $\mathbf{S}$.
3. Compute the eigenvalue decomposition $\mathbf{S} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T$.
4. A P-dimensional representation is obtained by computing the product $\hat{\mathbf{X}} = \mathbf{I}_{P\times N}\boldsymbol{\Lambda}^{1/2}\mathbf{U}^T$.

Fig. 4.1. Algorithm for classical metric multidimensional scaling.

Embedding of test set

If the data set is available as coordinates, then the equivalence between metric MDS and PCA can be used to easily embed a test set. In practice, this means that the principal components $\mathbf{v}_d$ are explicitly known, after either the SVD $\mathbf{Y} = \mathbf{V}\boldsymbol{\Sigma}\mathbf{U}^T$ or the EVD of the estimated covariance matrix $\hat{\mathbf{C}}_{\mathbf{yy}} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^T$. A point $\mathbf{y}$ of the test set is then embedded by computing the product:

$$\hat{\mathbf{x}} = \mathbf{I}_{P\times D}\mathbf{V}^T\mathbf{y} \ , \tag{4.50}$$

as for any point $\mathbf{y}(i)$ of the data set.

If the test set is given as scalar products, or if the scalar products are computed from the coordinates, then the knowledge of $\mathbf{V}$ is useless. Indeed, in that case, a test point $\mathbf{y}$ is written as the column vector

$$\begin{aligned} \mathbf{s} &= [\langle \mathbf{y}(i) \cdot \mathbf{y} \rangle]_{1\leq i\leq N} && (4.51) \\ &= \mathbf{Y}^T\mathbf{y} \ . && (4.52) \end{aligned}$$

Knowing that the SVD of $\mathbf{Y}$ is written as $\mathbf{Y} = \mathbf{V}\boldsymbol{\Sigma}\mathbf{U}^T$, it follows that

$$\mathbf{s} = \mathbf{U}\boldsymbol{\Sigma}^T\mathbf{V}^T\mathbf{y} \ . \tag{4.53}$$

Assuming that the test point $\mathbf{y}$ has been generated according to $\mathbf{y} = \mathbf{V}\mathbf{I}_{D\times P}\mathbf{x}$, then

$$\begin{aligned} \mathbf{s} &= \mathbf{U}\boldsymbol{\Sigma}^T\mathbf{V}^T\mathbf{V}\mathbf{I}_{D\times P}\mathbf{x} && (4.54) \\ &= \mathbf{U}\boldsymbol{\Sigma}^T\mathbf{I}_{D\times P}\mathbf{x} \ . && (4.55) \end{aligned}$$

where $\mathbf{V}$ disappears. Knowing the EVD $\mathbf{Y}^T\mathbf{Y} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T$, it further follows that

$$\mathbf{s} = \mathbf{U}\boldsymbol{\Lambda}^{1/2}\mathbf{I}_{N\times P}\mathbf{x} \ , \tag{4.56}$$

and, eventually,

$$\hat{\mathbf{x}} = \mathbf{I}_{P\times N}\mathbf{\Lambda}^{-1/2}\mathbf{U}^T\mathbf{s} \ , \tag{4.57}$$

which gives the desired P-dimensional coordinates. This corresponds to the Nyström formula [6, 16].

If the test set is given as distances, then a test point $\mathbf{y}$ is written as the column vector

$$\begin{aligned}\mathbf{D} &= [\langle \mathbf{y}(i) - \mathbf{y} \cdot \mathbf{y}(i) - \mathbf{y}\rangle]_{1\leq i\leq N} & (4.58)\\ &= [\langle \mathbf{y}(i) \cdot \mathbf{y}(i)\rangle - 2\langle \mathbf{y}(i) \cdot \mathbf{y}\rangle + \langle \mathbf{y} \cdot \mathbf{y}\rangle]_{1\leq i\leq N} & (4.59)\\ &= -2\mathbf{s} + [\langle \mathbf{y}(i) \cdot \mathbf{y}(i)\rangle + \langle \mathbf{y} \cdot \mathbf{y}\rangle]_{1\leq i\leq N} \ . & (4.60)\end{aligned}$$

Actually, a slightly modified version of the double centering can be applied in order to determine $\mathbf{s}$:

$$\mathbf{s} = -\frac{1}{2}(\mathbf{d} - \frac{1}{N}\mathbf{1}_N\mathbf{1}_N^T\mathbf{d} - \frac{1}{N}\mathbf{D}\mathbf{1}_N + \frac{1}{N^2}\mathbf{1}_N\mathbf{1}_N^T\mathbf{D}\mathbf{1}_N) \ , \tag{4.61}$$

where the second term is a column vector repeating the mean of $\mathbf{d}$, the third term is a column vector containing the mean of the rows of $\mathbf{D}$, and the fourth term is a column vector repeating the grand mean of $\mathbf{D}$. The above equation is only a discrete approximation and stems from a continuous formulation of double centering where $N \to \infty$ (see [16] for more details).

Example

Figure 4.2 shows the two-dimensional embeddings of the two data sets (Fig. 1.4) described in Section 1.5. Knowing that metric MDS projects data in a linear way, the results in Fig. 4.2 are not very surprising. Intuitively, they look like pictures of the two manifolds shot from aside (Swiss roll) and from above (open box), respectively.

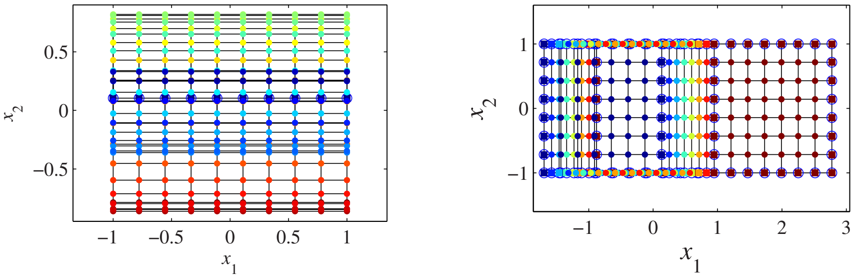


Fig. 4.2. Two-dimensional embeddings of the “Swiss roll” and “open box” data sets (Fig. 1.4), found by metric MDS.

In the case of the Swiss roll, such a result is not very useful, since all turns of the manifold are superposed. Similarly, for the open box, the presence of lateral faces can be guessed visually, but with difficulty: only the bottom face and the lid are clearly visible. These disappointing results can be explained theoretically by looking at the first couple of normalized eigenvalues

$$[\lambda_n]_{1 \leq n \leq N} = [0.46, 0.31, 0.22, 0.00, 0.00, 0.00, \ldots] \tag{4.62}$$

for the Swiss roll and

$$[\lambda_n]_{1 \leq n \leq N} = [0.58, 0.24, 0.18, 0.00, 0.00, 0.00, \ldots] \tag{4.63}$$

for the open box. In accordance with theory, only three eigenvalues are nonzero. Unfortunately, none of these three eigenvalues can be neglected compared to the others. This clearly means that metric MDS fails to detect that the two benchmark manifolds are two-manifolds. Any embedding with dimensionality less than three would cause an important information loss.

Classification

Exactly like PCA, metric MDS is an offline or batch method. The optimization method is exact and purely algebraical: the optimal solution is obtained in closed form. Metric MDS is also said to be a spectral method, since the core operation in its procedure is an EVD of a Gram matrix. The model is continuous and strictly linear. The mapping is implicit. Other characteristics of PCA are also kept: eigenvalues of the Gram matrix can be used to estimate the intrinsic dimensionality, and several embeddings can be built incrementally by adding or removing eigenvectors in the solution.

Advantages and drawbacks

Classical metric MDS possesses all the advantages and drawbacks of PCA: it is simple, robust, but strictly linear. By comparison with PCA, metric MDS is more flexible: it accepts coordinates as well as scalar products or Euclidean distances. On the other hand, running metric MDS requires more memory than PCA, in order to store the N-by-N Gram matrix (instead of the D-by-D covariance matrix). Another limitation is the generalization to new data points, which involves an approximate formula for the double-centering step.

Variants

Classical metric MDS has been generalized into metric MDS, for which pairwise distances, instead of scalar products, are explicitly preserved. The term *stress function* has been coined to denote the objective function of metric MDS, which can be written as

$$E_{\text{mMDS}} = \frac{1}{2} \sum_{i,j=1}^{N} w_{ij} (d_{\mathbf{y}}(i,j) - d_{\mathbf{x}}(i,j))^2 \ , \tag{4.64}$$

where $d_{\mathbf{y}}(i,j)$ and $d_{\mathbf{x}}(i,j)$ are the Euclidean distances in the high- and low-dimensional space, respectively. In practice, the nonnegative weights w_{ij} are often equal to one, except for missing data ($w_{ij} = 0$) or when one desires to focus on more reliably measured dissimilarities $d_{\mathbf{y}}(i,j)$. Although numerous variants of metric MDS exist, only one particular version, namely Sammon's nonlinear mapping, is studied in the next section. Several reasons explain this choice: Sammon's nonlinear mapping meets a wide success beyond the usual fields of application of MDS. Moreover, Sammon provided his NLDR method with a well-explained and efficient optimization technique.

In human science, the assumption that collected proximity values are distance measures might be too strong. Shepard [171] and Kruskal [108] addressed this issue and developed a method known as nonmetric multidimensional scaling. In nonmetric MDS, only the ordinal information (i.e., proximity ranks) is used for determining the spatial representation. A monotonic transformation of the proximities is calculated, yielding scaled proximities. Optimally scaled proximities are sometimes referred to as *disparities.* The problem of nonmetric MDS then consists of finding a spatial representation that minimizes the squared differences between the optimally scaled proximities and the distances between the points. In contrast to metric MDS, nonmetric MDS does not attempt to reproduce scalar products but explicitly optimizes a quantitative criterion that measures the preservation of the pairwise distances. Most variants of nonmetric MDS optimize the following stress function:

$$E_{\text{nMDS}} = \sqrt{\frac{\sum_{i,j=1}^{N} w_{ij} \left| f(\delta(i,j)) - d_{\mathbf{x}}(i,j) \right|^2}{c}} \ , \tag{4.65}$$

where

- $\delta(i,j)$ are the collected proximities;
- f is a monotonic transformation of the proximities, such that the assumption $f(\delta(i,j)) \approx d_{\mathbf{y}}(i,j)$ holds, where $d_{\mathbf{y}}(i,j)$ is the Euclidean distance between the unknown data points $\mathbf{y}(i)$ and $\mathbf{y}(j)$;
- $d_{\mathbf{x}}(i,j)$ is the Euclidean distance between the low-dimensional representations $\mathbf{x}(i)$ and $\mathbf{x}(j)$ of $\mathbf{y}(i)$ and $\mathbf{y}(j)$;
- c is a scale factor usually equal to $\sum_{i,j=1}^{N} w_{ij} d_{\mathbf{y}}(i,j)$;
- w_{ij} are nonnegative weights, with the same meaning and usage as for metric MDS.

More details can be found in the vast literature dedicated to nonmetric MDS; see, for instance, [41, 25] and references therein.

4.2.3 Sammon's nonlinear mapping

In 1969 Sammon [165] proposed a method to establish a mapping between a high-dimensional space and a lower-dimensional one. Actually, the word "mapping" may seem rather misleading. Indeed, Sammon's method does not exactly determine a continuous mapping between two Cartesian spaces: its real purpose is just to reduce the dimensionality of a finite set of data points. Sammon's method has met a considerable success, and still today much work is devoted to either its enhancement [148] or its applications. Due to its ubiquity in many fields of data analysis, several names or acronyms refer to Sammon's method: Sammon's nonlinear mapping, Sammon mapping, NLM (standing for nonlinear mapping), etc. The acronym NLM is preferred in this book.

Embedding of data set

Actually, NLM is closely related to metric MDS (see Subsection 4.2.2). No generative model of data is assumed: only a stress function is defined. Consequently, the low-dimensional representation can be totally different from the distribution of the true latent variables. More precisely, NLM minimizes the following stress function:

$$E_{\mathrm{NLM}} = \frac{1}{c} \sum_{\substack{i=1\\ i<j}}^{N} \frac{(d_{\mathbf{y}}(i,j) - d_{\mathbf{x}}(i,j))^2}{d_{\mathbf{y}}(i,j)} \quad , \tag{4.66}$$

where

- $d_{\mathbf{y}}(i,j)$ is a distance measure between the ith and jth points in the D-dimensional data space,
- $d_{\mathbf{x}}(i,j)$ is the Euclidean distance between the ith and jth points in the P-dimensional latent space.

The normalizing constant c is defined as

$$c = \sum_{\substack{i=1\\ i<j}}^{N} d_{\mathbf{y}}(i,j) \quad . \tag{4.67}$$

In the definition of E_{NLM}, no assumption is made about the distance function $d_{\mathbf{y}}(i,j)$ in the high-dimensional space. Classically, the Euclidean distance is chosen by default. Moreover, as the algorithm usually works in batch mode, all distances $d_{\mathbf{y}}(i,j)$ must be known in advance. On the contrary, the distances in the low-dimensional space are imposed to be Euclidean for the sake of simplicity in the following developments. Hence, $d_{\mathbf{x}}(i,j) = \|\mathbf{x}(i) - \mathbf{x}(j)\|_2$, where the points $\mathbf{x}(i)$ and $\mathbf{x}(j)$ are the low-dimensional representations of the data points $\mathbf{y}(i)$ and $\mathbf{y}(j)$.

Sammon's stress can be cast as an instance of metric MDS (see (4.64)), for which $w_{ij} = 1/d_{\mathbf{y}}(i,j)$. The intuitive meaning of the factor $1/d_{\mathbf{y}}(i,j)$, which is weighting the summed terms, is clear: it gives less importance to errors made on large distances. During the dimensionality reduction, a manifold should be unfolded in order to be mapped to a Cartesian vector space, which is flat, in contrast with the initial manifold, which can be curved. This means that long distances, between faraway points, cannot be preserved perfectly if the curvature of the manifold is high: they have to be stretched in order to "flatten" the manifold. On the other hand, small distances can be better preserved since on a local scale the curvature is negligible, or at least less important than on the global scale. Moreover, the preservation of short distances allows us to keep the local cohesion of the manifold. In summary, the weighting factor simply adjusts the importance to be given to each distance in Sammon's stress, according to its value: the preservation of long distances is less important than the preservation of shorter ones, and therefore the weighting factor is chosen to be inversely proportional to the distance.

Obviously, Sammon's stress E_{NLM} is never negative and vanishes in the ideal case where $d_{\mathbf{y}}(i,j) = d_{\mathbf{x}}(i,j)$ for all pairs $\{i,j\}$. The minimization of E_{NLM} is performed by determining appropriate coordinates for the low-dimensional representations $\mathbf{x}(i)$ of each observation $\mathbf{y}(i)$. Although E_{NLM} is a relatively simple continuous function, its minimization cannot be performed in closed form, in contrast with the error functions of PCA and classical metric MDS. Nevertheless, standard optimization techniques can be applied in order to find a solution in an iterative manner. Sammon's idea relies on a variant of Newton's method, called quasi-Newton optimization (see Appendix C.1). This method is a good tradeoff between the exact Newton method, which involves the Hessian matrix, and a gradient descent, which is less efficient. As Sammon's stress E_{NLM} depends on NP parameters, the Hessian would have been much too big! According to Eq. (C.12), the quasi-Newton update rule that iteratively determines the parameters $x_k(i)$ of E_{NLM} can be written as

$$x_k(i) \leftarrow x_k(i) - \alpha \frac{\frac{\partial E_{\text{NLM}}}{\partial x_k(i)}}{\left|\frac{\partial^2 E_{\text{NLM}}}{\partial x_k(i)^2}\right|} \quad , \tag{4.68}$$

where the absolute value is used to distinguish the minima from the maxima. Sammon [165] recommends setting α (called *magic factor* in his paper) between 0.3 and 0.4.

As $d_{\mathbf{x}}(i,j)$ is the Euclidean distance between vectors $\mathbf{x}(i)$ and $\mathbf{x}(j)$, it follows out from Eq. (4.9) that

$$d_{\mathbf{x}}(i,j) = \|\mathbf{x}(i) - \mathbf{x}(j)\|_2 = \sqrt{\sum_{k=1}^{P}(x_k(i) - x_k(j))^2} \quad . \tag{4.69}$$

Therefore, the first partial derivative of E_{NLM} is

$$\frac{\partial E_{\text{NLM}}}{\partial x_k(i)} = \frac{\partial E_{\text{NLM}}}{\partial d_{\mathbf{x}}(i,j)} \frac{\partial d_{\mathbf{x}}(i,j)}{\partial x_k(i)} \tag{4.70}$$

$$= \frac{-2}{c} \sum_{\substack{j=1 \\ j \neq i}}^{N} \frac{d_{\mathbf{y}}(i,j) - d_{\mathbf{x}}(i,j)}{d_{\mathbf{y}}(i,j)} \frac{\partial d_{\mathbf{x}}(i,j)}{\partial x_k(i)} \tag{4.71}$$

$$= \frac{-2}{c} \sum_{\substack{j=1 \\ j \neq i}}^{N} \frac{d_{\mathbf{y}}(i,j) - d_{\mathbf{x}}(i,j)}{d_{\mathbf{y}}(i,j)} \frac{(x_k(i) - x_k(j))}{d_{\mathbf{x}}(i,j)} \tag{4.72}$$

$$= \frac{-2}{c} \sum_{\substack{j=1 \\ j \neq i}}^{N} \frac{d_{\mathbf{y}}(i,j) - d_{\mathbf{x}}(i,j)}{d_{\mathbf{y}}(i,j)\, d_{\mathbf{x}}(i,j)} (x_k(i) - x_k(j)) \ . \tag{4.73}$$

After simplification, the second derivative of E_{NLM} is

$$\frac{\partial^2 E_{\text{NLM}}}{\partial x_k^2(i)} = \frac{-2}{c} \sum_{\substack{j=1 \\ j \neq i}}^{N} \left(\frac{d_{\mathbf{y}}(i,j) - d_{\mathbf{x}}(i,j)}{d_{\mathbf{y}}(i,j)\, d_{\mathbf{x}}(i,j)} - \frac{(x_k(i) - x_k(j))^2}{d_{\mathbf{x}}^3(i,j)} \right) \ . \tag{4.74}$$

The procedure detailed in Fig. 4.3 implements the above-mentioned ideas. A MATLAB® function that performs Sammon's nonlinear mapping can be found in the SOM toolbox, which is available at `http://www.cis.hut.fi/projects/somtoolbox/`. A C++ implementation can also be downloaded from `http://www.ucl.ac.be/mlg/`. In addition to the embedding dimensionality P, Sammon's NLM involves several parameters, essentially due to its iterative optimization scheme. These are the number of iterations and the magic factor α; also, it is noteworthy that initialization may play a part in the final result. Space complexity is $\mathcal{O}(N^2)$, corresponding to the amount of memory required to store the pairwise distances. Time complexity is $\mathcal{O}(N^2 P)$ per iteration.

1. Compute all pairwise distances $d_{\mathbf{y}}(i,j)$ in the D-dimensional data space.
2. Initialize the P-dimensional coordinates of all points $\mathbf{x}(i)$, either randomly or on the hyperplane spanned by the first P principal components of the data set (after PCA or MDS).
3. Compute the right-hand side of Eq. (4.68) for the coordinates of all points $\mathbf{x}(i)$.
4. Update the coordinates of all points $\mathbf{x}(i)$.
5. Return to step 3 until the value of the stress function no longer decreases.

Fig. 4.3. Algorithm implementing Sammon's nonlinear mapping.

Embedding of test set

Sammon's original method was published without regards to the embedding of test points. Test data available as distances or coordinates can be embedded by means of two techniques:

- The easiest solution consists of running a modified NLM for each test point, where all data points are taken into account but only the test point to be embedded is updated. Unfortunately, this procedure generally gives poor results, because NLM tries to find a *global* minimum of the stress function for each test point. Indeed, it is intuitively clear that a *local* projection would perform better: only a few data points around a given test point are needed to place it correctly in the low-dimensional space.
- The interpolation procedure of CCA (see Subsection 4.2.4) could also be used, precisely because it can perform a local projection thanks to its adjustable neighborhood width. But it does not behave much better than the preceding solution.

When points are available as coordinates only, another possibility exists. It involves using neural variants of NLM, like the SAMANN (discussed ahead), which naturally provides the ability to generalize the mapping to test points, at the expense of a more complex and perhaps less precise algorithm for the embedding of the data set.

Example

Figure 4.4 shows the two-dimensional embeddings of the two data sets (Fig. 1.4) described in Section 1.5. These results are obtained with the step size α set to 0.3. By comparison with metric MDS, NLM can embed in a nonlinear way. It is clearly visible that the two manifolds look “distorted” in

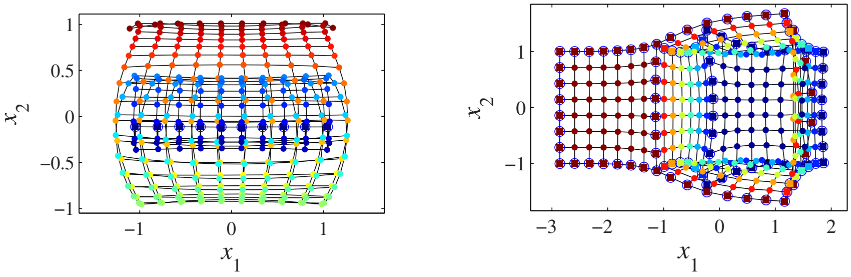


Fig. 4.4. Two-dimensional embeddings of the ‘Swiss roll’ and ‘open box’ data sets (Fig. 1.4), found by Sammon's NLM.

their two-dimensional embedding: some parts are stretched while others are compressed.

Unfortunately, the results of NLM remain disappointing for the two benchmark manifolds, particularly for the Swiss roll. Turns of the spiral are superposed, meaning that the mapping between the initial manifold and its two-dimensional embedding is not bijective. For the open box, NLM leads to a better result than metric MDS, but two faces are still superposed.

The shape of the embedded manifolds visually shows how NLM trades off the preservation of short against longer distances.

Classification

Sammon's NLM is a batch method. The model is nonlinear and discrete, producing an explicit mapping; no generative model of data is provided. No vector quantization was included in the original method described in Subsection 4.2.3. However, this useful preprocessing can easily be grafted on the method.

Sammon's NLM uses an approximate optimization procedure, which can possibly get stuck in a local minimum. The method does not include an estimator of the intrinsic dimensionality; the embedding dimension is actually fixed by the user. Incremental or layered embeddings are not possible: the method must be run separately for each specified dimensionality.

Advantages and drawbacks

By comparison with classical metric MDS, NLM can efficiently handle nonlinear manifolds, at least if they are not too heavily folded. Among other nonlinear variants of metric MDS, NLM remains relatively simple and elegant.

As a main drawback, NLM lacks the ability to generalize the mapping to new points.

In the same way as many other distance-preserving methods, NLM in its original version works with a complete matrix of distances, hence containing $\mathcal{O}(N^2)$ entries. This may be an obstacle when embedding very large data sets.

Another shortcoming of NLM is its optimization procedure, which may be slow and/or inefficient for some data sets. In particular, Sammon's stress function is not guaranteed to be concave; consequently the optimization process can get stuck in a local minimum.

Variants of the original NLM, briefly presented ahead, show various attempts to address the above-mentioned issues.

Variants

The SAMANN [134, 44], standing for Sammon artificial neural network, is a variant of the original NLM that optimizes the same stress function but achieves it in a completely different way. Instead of minimizing the stress function as such with a standard optimization technique, which leads to a discrete mapping of the data set, Mao and Jain propose establishing an implicit mapping between the initial space and the final space. They propose is a three-layer MLP with D inputs and P outputs, working in an unsupervised way with a modified back-propagation rule. More precisely, that rule takes into account the pairwise distances between the data points in a rather original way. Indeed, the SAMANN does not update the network parameters after the presentation of each input/output pair, as a usual supervised MLP does. Instead, the SAMANN updates its parameters after the presentation of each pair of *in*puts. The distance between the two corresponding outputs, compared to the distance measured between the two inputs, allows one to derive the necessary corrections for the network parameters in a totally unsupervised way. The main drawback of the SAMANN is that it requires scaling the data in order to cast the pairwise distances within the range of the (sigmoidal) outputs of an MLP. Moreover, despite its appealing elegance, the SAMANN is outperformed by another extension of NLM with an MLP. In that case [44], the original NLM is not replaced with an MLP: NLM is run as usual and if generalization to new points is needed afterwards, a classical supervised MLP learns the mapping previously obtained by NLM.

Fast versions of NLM, or versions using less memory space, are studied, for example, in [148]. These variants avoid using the whole distance matrix by using, for instance, triangulation methods. Another way of reducing the problem size consists of using vector quantization before running NLM.

From an algorithmic point of view, even when the original architecture of NLM is kept, some freedom remains in the way the stress function is optimized. It has often been reported that the second-order gradient method Sammon proposed has several weaknesses. The quasi-Newton optimization neglects off-diagonal entries of the Hessian matrix. Moreover, in order to distinguish minima from maxima, the remaining diagonal elements are taken in absolute values; this simple trick just changes the direction of the update in Eq. (4.68) while a modification of its norm would be required, too. Briefly put, the step size in the gradient descent of the original NLM is computed by an often ill-suited heuristics. Various other optimization techniques, showing better properties, may be used.

Finally, until now, very few variants of NLM have considered a change of metric to increase the ability of NLM to deal with heavily curved manifolds. Actually, as already mentioned, the Euclidean distance plays an essential role only in the embedding space, in order to derive a simple update rule from the stress function. In the data space, any distance function may suit, at least if it behaves more or less in the same way as the Euclidean distance, seeing that the stress function tries to equate them. A version of NLM using graph distances is described in Subsection 4.3.3.

4.2.4 Curvilinear component analysis

Demartines and Hérault proposed curvilinear component analysis (CCA) in 1995 [47], as an enhancement of Kohonen's self-organizing maps (see Subsection 5.2.1) when the latter method is used for nonlinear dimensionality reduction [48]. Nevertheless, and also despite its denomination being very close to that of PCA, CCA does not derive or even resemble either PCA or a SOM. Actually, CCA belongs to the class of distance-preserving methods and is more closely related to Sammon's NLM [48]. Common points between CCA and SOM must be sought elsewhere:

First, like an SOM, the CCA procedure includes a vector quantization step (see Appendix D). CCA was actually the first method to combine vector quantization with a nonlinear dimensionality reduction achieved by distance preservation. Historically, indeed, CCA was called VQP in [46, 45], which stands for vector quantization and projection.

Second, CCA and an SOM work with the same kind of optimization techniques, borrowed from the field of artificial neural networks. In the original VQP method decribed by Demartines in his PhD thesis, the algorithm performs simultanously the vector quantization and the nonlinear dimensionality reduction, exactly like an SOM.

In the following, however, the vector quantization is considered an optional preprocessing of the data. Actually, vector quantization can be applied to reduce the number of vectors in large databases, for DR method. For small databases or sparsely sampled manifolds, however, it is often better to skip vector quantization, in order to fully exploit the available information.

Embedding of data set

Like most distance-preserving methods, CCA minimizes a stress or error function, which is written as

$$E_{\mathrm{CCA}} = \frac{1}{2} \sum_{\substack{i=1\\j=1}}^{N} (d_{\mathbf{y}}(i,j) - d_{\mathbf{x}}(i,j))^2 F_\lambda(d_{\mathbf{x}}(i,j)) \qquad (4.75)$$

and closely resembles Sammon's stress function. As for the latter, no generative model of data is assumed. Just as usual, $d_{\mathbf{y}}(i,j)$ and $d_{\mathbf{x}}(i,j)$ are, respectively, the distances in the data space and the Euclidean distance in the latent space. There are two differences, however:

- No scaling factor stands in front of the sum; this factor is not very important, except for quantitative comparisons.
- The weighting $1/d_{\mathbf{y}}(i,j)$ is replaced with a more generic factor $F_\lambda(d_{\mathbf{x}}(i,j))$.

Of course, F_λ may not be any function. Like for the weighting of Sammon's NLM, the choice of F_λ is guided by the necessity to preserve short distances prior to longer ones. Because the global shape of the manifold has to be unfolded, long distances often have to be stretched, and their contribution in the stress should be low. On the other hand, the good preservation of short distances is easier (since the curvature of the manifold is often low on a local scale) but is also more important, in order to preserve the structure of the manifold. Consequently, F_λ is typically chosen as a monotically decreasing function of its argument. As CCA works on finite data sets, F_λ is also chosen bounded in order to prevent an abnormally short or null distance to dominate the other contributions in the stress function. This is especially critical because F_λ depends on the distances in the embedding space, which are varying and could temporarily be very small. Indeed, more important than the function F_λ in itself is the *argument* of F_λ. In contrast with Sammon's stress, the weighting does not depend on the constant distances measured in the data space but on the distances being optimized in the embedding space. When distances can be preserved, the hypothesis $d_{\mathbf{y}}(i,j) \approx d_{\mathbf{x}}(i,j)$ holds and CCA behaves more or less in the same way as NLM. If $d_{\mathbf{x}}(i,j) \ll d_{\mathbf{y}}(i,j)$ for some i and j, then the manifold is highly folded up and the contribution of this pair will increase in order to correct the flaw. But what happens if $d_{\mathbf{x}}(i,j) \gg d_{\mathbf{y}}(i,j)$? Then the contribution of this pair will decrease, meaning intuitively that CCA will allow some stretching not only for long distances but also for shorter ones.

Demartines and Hérault designed an optimization procedure specifically tailored for CCA. Like most optimization techniques, it is based on the derivative of E_{CCA}. Using the short-hand notations $d_{\mathbf{y}} = d_{\mathbf{y}}(i,j)$ and $d_{\mathbf{x}} = d_{\mathbf{x}}(i,j)$, the derivative can be written as

$$\begin{aligned}\frac{\partial E_{\text{CCA}}}{\partial x_k(i)} &= \frac{\partial E_{\text{CCA}}}{\partial d_{\mathbf{x}}} \frac{\partial d_{\mathbf{x}}}{\partial x_k(i)} \\ &= \sum_{j=1}^{N} (d_{\mathbf{y}} - d_{\mathbf{x}}) \big(2F_\lambda(d_{\mathbf{x}}) - (d_{\mathbf{y}} - d_{\mathbf{x}}) F'_\lambda(d_{\mathbf{x}})\big) \frac{x_k(j) - x_k(i)}{d_{\mathbf{x}}} \quad .\end{aligned}$$

(See Eqs. (4.10) and (4.11) for the derivative of $d_{\mathbf{x}}(i,j)$.) Alternatively, in vector form, this gives

$$\nabla_{\mathbf{x}(i)} E_{\text{CCA}} = \sum_{j=1}^{N} (d_{\mathbf{y}} - d_{\mathbf{x}}) \big(2F_\lambda(d_{\mathbf{x}}) - (d_{\mathbf{y}} - d_{\mathbf{x}}) F'_\lambda(d_{\mathbf{x}})\big) \frac{\mathbf{x}(j) - \mathbf{x}(i)}{d_{\mathbf{x}}} \quad , \tag{4.76}$$

where $\nabla_{\mathbf{x}(i)} E_{\text{CCA}}$ represents the gradient of E_{CCA} with respect to vector $\mathbf{x}(i)$. The minimization of E_{CCA} by a gradient descent gives the following update rule:

$$\mathbf{x}(i) \leftarrow \mathbf{x}(i) - \alpha \nabla_{\mathbf{x}(i)} E_{\text{CCA}} \quad , \tag{4.77}$$

where α is a positive learning rate scheduled according to the Robbins–Monro conditions [156]. Clearly, the modification brought to vector $\mathbf{x}(i)$ proves to be a sum of contributions coming from all other vectors $\mathbf{x}(j)$ $(j \neq i)$. Each contribution may be seen as the result of a repulsive or attractive force between vectors $\mathbf{x}(i)$ and $\mathbf{x}(j)$. Each contribution can also be written as a scalar coefficient $\beta(i,j)$ multiplying the unit vector $(\mathbf{x}(j)-\mathbf{x}(i))/d_{\mathbf{x}}(i,j)$. The scalar coefficient is

$$\beta(i,j) = (d_{\mathbf{y}} - d_{\mathbf{x}})\big(2F_\lambda(d_{\mathbf{x}}) - (d_{\mathbf{y}} - d_{\mathbf{x}})F'_\lambda(d_{\mathbf{x}})\big) = \beta(j,i) \ . \tag{4.78}$$

Provided that condition

$$2F_\lambda(d_{\mathbf{x}}) > (d_{\mathbf{y}} - d_{\mathbf{x}})F'_\lambda(d_{\mathbf{x}}) \tag{4.79}$$

holds, the coefficient is negative when the low-dimensional distance $d_{\mathbf{x}}(i,j)$ is shorter than the original high-dimensional distance $d_{\mathbf{y}}(i,j)$, and positive in the converse case. This means that when vector $\mathbf{x}(j)$ is too far from vector $\mathbf{x}(i)$, it is moved closer, whereas it is moved farther away in the converse case. Hence, the update rule just behaves as intuitively expected. However, it has a major drawback; the sum of all radial contributions to the update of a given vector $\mathbf{x}(i)$ leads to an averaging; as a consequence, the update is small and the convergence slows down. Intuitively, the update rule behaves like a very careful diplomat: when listening to numerous contradictory opinions at the same time, he makes no clear decision and the negotiations stagnate. As a consequence, the update rule easily gets stuck in a local minimum of E_{CCA}: the sum of all contributions vanishes, whereas the same terms taken individually are nonzero. This phenomenon is illustrated in Fig. 4.5.

A simple modification of the update rule allows the diplomat to show more dynamic behavior. The idea just consists of reorganizing the negotiations in such a way that every participant gives its opinion sequentially while the others stay quiet, and the diplomat immediately decides. For this purpose, E_{CCA} is decomposed in a sum of indiced terms:

$$E_{\mathrm{CCA}} = \sum_{i=1}^{N} E^i_{\mathrm{CCA}} \ , \tag{4.80}$$

where

$$E^i_{\mathrm{CCA}} = \frac{1}{2}\sum_{j=1}^{N}(d_{\mathbf{y}}(i,j) - d_{\mathbf{x}}(i,j))^2 F_\lambda(d_{\mathbf{x}}(i,j)) \ . \tag{4.81}$$

The separate optimization of each E^i_{CCA} leads to the following update rule:

$$\begin{aligned}\mathbf{x}(j) &\leftarrow \mathbf{x}(j) - \alpha\nabla_{\mathbf{x}(j)}E^i_{\mathrm{CCA}}\\ &\leftarrow \mathbf{x}(j) - \alpha\beta(i,j)\frac{\mathbf{x}(i)-\mathbf{x}(j)}{d_{\mathbf{x}}} \ .\end{aligned} \tag{4.82}$$

The modified procedure works by interlacing the gradient descent of each term E^i_{CCA}: it updates all $\mathbf{x}(j)$ for $E^{i=1}_{\mathrm{CCA}}$, then all $\mathbf{x}(j)$ for $E^{i=2}_{\mathrm{CCA}}$, and so on. Geometrically, the new procedure pins up a vector $\mathbf{x}(i)$ and moves all other $\mathbf{x}(j)$ radially, without regards to the cross contributions computed by the

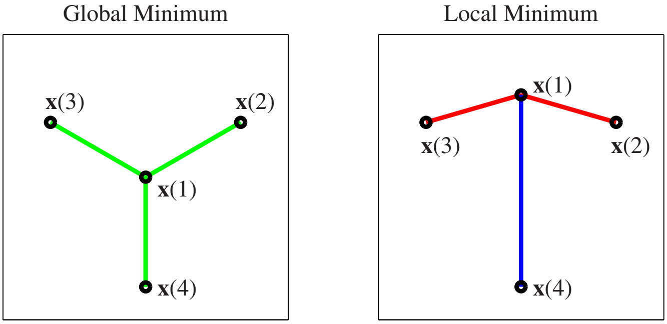


Fig. 4.5. Visual example of local minimum that can occur with distance preservation. The expected solution is shown on the left: the global minimum is attained when $\mathbf{x}(1)$ is exactly in the middle of the triangle formed by $\mathbf{x}(2)$, $\mathbf{x}(3)$, and $\mathbf{x}(4)$. A local minimum is shown on the right: $d_{\mathbf{x}}(1,2)$ and $d_{\mathbf{x}}(1,3)$ are too short, whereas $d_{\mathbf{x}}(1,4)$ is too long. In order to reach the global minimum, $\mathbf{x}(1)$ must be moved down, but this requires shortening $d_{\mathbf{x}}(1,2)$ and $d_{\mathbf{x}}(1,3)$ even further, at least temporarily. This is impossible for a classical gradient descent: the three contributions from $\mathbf{x}(2)$, $\mathbf{x}(3)$, and $\mathbf{x}(4)$ vanish after adding them, although they are not zero when taken individually. Actually, the points $\mathbf{x}(2)$, $\mathbf{x}(3)$ tend to move $\mathbf{x}(1)$ away, while $\mathbf{x}(4)$ pulls $\mathbf{x}(1)$ toward the center of the triangle. With a stochastic approach, the contributions are considered one after the other, in random order: for example, if the contribution of $\mathbf{x}(4)$ is taken into account first, then $\mathbf{x}(1)$ can escape from the local minimum.

first rule (classical gradient, Eq. (4.77)). Computationally, the new procedure performs many more updates than the first rule for (almost) the same number of operations: while the application of the first rule updates one vector, the new rule updates $N-1$ vectors. Moreover, the norm of the updates is larger with the new rule than with the first one. A more detailed study in [45] shows that the new rule minimizes the global error function E_{CCA}, not strictly like a normal gradient descent, but well on average.

Unless the embedding dimensionality is the same as the initial dimensionality, in which case the embedding is trivial, the presence and the choice of F_λ are very important. As already mentioned, the embedding of highly folded manifolds requires focusing on short distances. Longer distances have to be stretched in order to achieve the unfolding, and their contribution must be lowered in E_{CCA}. Therefore, F_λ is usefully chosen as a positive and decreasing function. For example, F_λ could be defined as the following:

$$F_\lambda(d_{\mathbf{x}}) = \exp\left(-\frac{d_{\mathbf{x}}}{\lambda}\right) \quad , \tag{4.83}$$

where λ controls the decrease. However, the condition (4.79) must hold, so that the rule behaves as expected, i.e., points that are too far away are brought closer to each other and points that too close are moved away. As F_λ is positive and decreasing, the condition (4.79) becomes

$$\left|\frac{F_\lambda(d_\mathbf{x})}{F'_\lambda(d_\mathbf{x})}\right| > \frac{1}{2}(d_\mathbf{x} - d_\mathbf{y}) \ , \tag{4.84}$$

and in the particular case of Eq. (4.83):

$$\lambda > \frac{1}{2}(d_\mathbf{x} - d_\mathbf{y}) \ , \tag{4.85}$$

which is difficult to ensure practically, because the distances in the embedding space may grow during the unfolding. Consequently, F_λ is preferably defined as

$$F_\lambda(d_\mathbf{x}) = H(\lambda - d_\mathbf{x}) \ , \tag{4.86}$$

where $H(u)$ is a step function defined as:

$$H(u) = \begin{cases} 0 & \text{if } u \leq 0 \\ 1 & \text{if } u > 0 \end{cases} \ . \tag{4.87}$$

Of course, the main interest of this simple function is its null derivative making the condition (4.79) trivially fulfilled. Unfortunately, as a counterpart, the behavior of the function becomes a little bit abrupt: distances larger than the value of λ are not taken into account at all. As a consequence, the choice of λ becomes critical.

A first way to circumvent the problem consists of using different values for λ during the convergence. Like in an SOM (see Subsection 5.2.1), the user can schedule the value of λ: high values, close to the maximal distance measured in the data, can be used during the first updates, whereas lower values finely tune the embedding when convergence is almost reached.

A second way to circumvent the critical choice of λ consists in neglecting the derivative of F_λ in the update rule. This can be seen as using a staircase function with infinitely small stairwidths.

In both cases, the hyperparameter λ may be interpreted as a neighborhood width, as in an SOM.

Figure 4.6 summarizes the procedure achieving CCA. A function written in MATLAB® that implements CCA can be found in the SOM toolbox, which is available at `http://www.cis.hut.fi/ projects/somtoolbox/`. A C++ implementation can also be downloaded from `http://www.ucl.ac.be/mlg/`. The main parameter of CCA is the embedding dimensionality. Like Sammon's mapping, CCA relies on an iterative optimization procedure that involves several parameters. The latter are the number of iterations and for each iteration the scheduled values of the learning rate α and neighborhood width λ. The choice of the weighting function can be considered, too. Finally, vector quantization also involves some parameters (number of prototypes, number of iterations, convergence parameters, etc.). Space complexity of CCA is $\mathcal{O}(M^2)$ as with most distance-preserving methods, whereas time complexity is $\mathcal{O}(M^2 P)$, where $M \leq N$ is the number of prototypes obtained after vector quantization.

1. Perform a vector quantization (see Appendix D) to reduce the size of the data set, if needed.
2. Compute all pairwise Euclidean distances $d_{\mathbf{y}}(i,j)$ in the D-dimensional data space.
3. Initialize the P-dimensional coordinates of all points $\mathbf{x}(i)$, either randomly or on the hyperplane spanned by the first principal components (after a PCA). Let q be equal to 1.
4. Give the learning rate α and the neighborhood width λ their scheduled value for epoch number q.
5. Select a point $\mathbf{x}(i)$, and update all other ones according to Eq. (4.82).
6. Return to step 5 until all points $\mathbf{x}(i)$ have been selected exactly once during the current epoch.
7. Increase q, and if convergence is not reached return to step 4.

Fig. 4.6. Algorithm implementing curvilinear component analysis.

Embedding of test set

Because it stems from the field of artificial neural networks, CCA has been provided with an interpolation procedure that can embed test points. Compared to the learning phase, the interpolation considers the prototypes (or the data points if the vector quantization was skipped) as fixed points. For each test point, the update rule (4.82) is applied, as in the learning phase, in order to move the embedded test point to the right position.

For simple manifolds, the interpolation works well and can even perform some basic extrapolation tasks [45, 48]. Unfortunately, data dimensionality that is too high or the presence of noise in the data set dramatically reduces the performance. Moreover, when the manifold is heavily crumpled, the tuning of the neighborhood width proves still more difficult than during the learning phase.

Example

Figure 4.7 shows the two-dimensional embeddings of the two data sets (Fig. 1.4) described in Section 1.5. As can be seen in the figure, CCA succeeds in embedding the two benchmark manifolds in a much more satisfying way than Sammon's NLM. The two-dimensional embedding of the Swiss roll is almost free of superpositions. To achieve this result, CCA tears the manifold in several pieces. However, this causes some neighborhood relationships between data points to be both arbitrarily broken and created. From the viewpoint of the underlying manifold, this also means that the mapping between the initial and final embeddings is bijective but discontinuous.

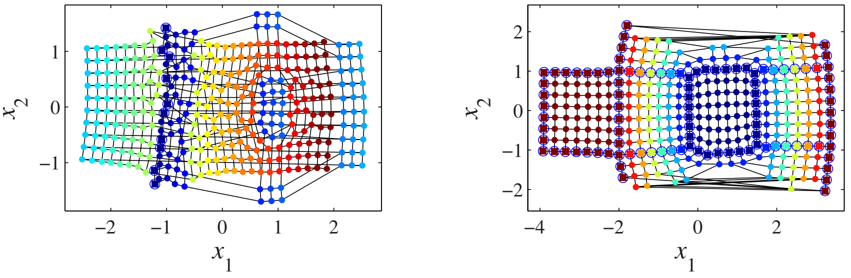


Fig. 4.7. Two-dimensional embeddings of the "Swiss roll" and "open box" data sets (Fig. 1.4), found by CCA.

Regarding the open box, the tearing strategy of CCA appears to be more convincing. All faces of the box are visible, and the mapping is both bijective and continuous. In this case, the weighted preservation of distances allows CCA to leave almost all neighborhood relationships unchanged, except where the two lateral faces are torn. Some short distances are shrunk near the bottom corners of the box.

Classification

Although VQP, the first version of CCA, was intended to be an online method, like usual versions of the SOM, that performed simultaneously vector quantization and dimensionality reduction, most current implementations of CCA are now simple batch procedures. Several reasons explain this choice. A clear separation between vector quantization and dimensionality reduction increases efficiency as well as modularity. Ready-to-use methods of vector quantization can advantageously preprocess the data. Moreover, these procedures may be skipped to ease the comparison with other DR methods, like Sammon's NLM, that are not provided with vector quantization.

In summary, CCA is an offline (batch) method with an optional vector quantization as preprocessing. The model is nonlinear and discrete; the mapping is explicit. The method works with an approximate optimization procedure. CCA does not include an estimator of the data dimensionality and cannot build embeddings incrementally. In other words, the intrinsic dimensionality has to be given as an external parameter and cannot be modified without running the method again.

Advantages and drawbacks

By comparison with Sammon's NLM, Demartines' CCA proves much more flexible, mainly because the user can choose and parameterize the weighting function F_λ in E_{CCA} at will. This allows one to limit the range of considered distances and focus on the preservation of distances on a given scale only. Moreover, the weighting function F_λ depends on the distances measured in the embedding space (and not in the data space like NLM); this allows tearing some regions of the manifold, as shown in the example. This is a better solution than crushing the manifold, like NLM does.

From a computational point of view, the stochastic optimization procedure of CCA works much better than the quasi-Newton rule of NLM. Convergence is generally faster.

On the other hand, the interpretation of CCA error criterion is difficult, since F_λ is changing when CCA is running, according to the schedule of λ. CCA can also get stuck in local minima, and sometimes convergence may fail when the learning rate α or the neighborhood width λ is not adequately scheduled. These are the unavoidable counterparts of the method's flexibility.

Variants

In 1998 Hérault [84] proposed an enhanced version of CCA including a model suited for noisy data. In the presence of noise, distances between the given noisy data points can be longer than the corresponding distances measured on the underlying manifold. This phenomenon, especially annoying for short distances, is illustrated in Fig. 4.8. Actually, the embedding of a distance-preserving method should not take into account the contribution of noise that is hidden in the distances. For that purpose, Hérault [84] decomposes the task of dimensionality reduction into two simultaneous subtasks. The first one is the global unfolding of the manifold, in such a way that it can be embedded in a lower-dimensional space. The second subtask aims at "projecting" locally the noisy data points onto the underlying manifold.

As unfolding generally stretches the distances, the condition to perform unfolding could be $d_{\mathbf{y}} \leq d_{\mathbf{x}}$. And since noise elimination tends to shrink the polluted distances, the corresponding condition is $d_{\mathbf{y}} \geq d_{\mathbf{x}}$.

For the global unfolding, Hérault keeps the former definition of E^i_{CCA}, which is written as

$$E^i_{\text{CCA-u}} = \frac{1}{2}\sum_{j=1}^{N}(d_{\mathbf{y}} - d_{\mathbf{x}})^2 F_\lambda(d_{\mathbf{x}}) \ . \tag{4.88}$$

On the other hand, for the local projection, the error function becomes

$$E^i_{\text{CCA-p}} = \frac{1}{2}\sum_{j=1}^{N}(d^2_{\mathbf{y}} - d^2_{\mathbf{x}})^2 F_\lambda(d_{\mathbf{x}}) \ , \tag{4.89}$$

where the quadratic term resembles the one used in Torgerson's classical metric MDS (see Subsection 4.2.2). The local projection thus behaves like a local PCA (or MDS) and projects the noisy points onto the underlying manifold (more or less) orthogonally.

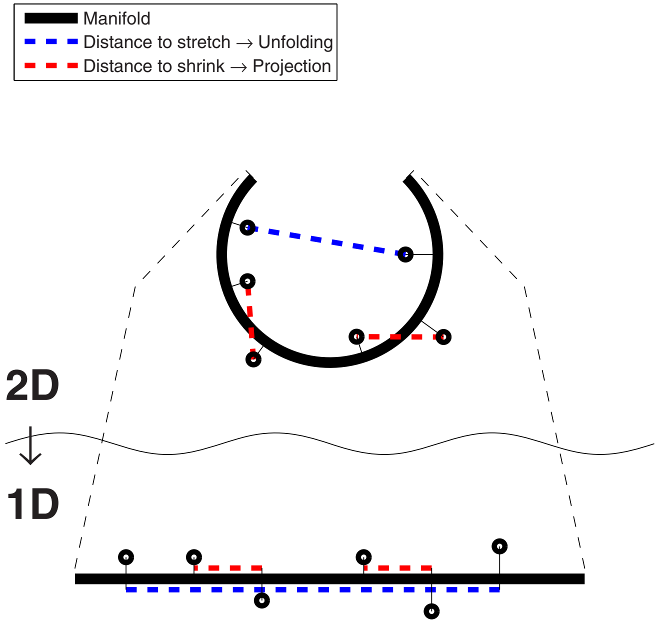

Manifold
Distance to stretch → Unfolding
Distance to shrink → Projection
2D
1D

Fig. 4.8. The C curve is a one-manifold embedded in a two-dimensional space. Assuming, in the theoretical case, that infinitely small distances are perfectly preserved, the dimensionality reduction then consists of unrolling the C curve in order to make it straight. In practical cases, only a discrete set of distances is available. Some distances between pairs of noisy points (bold black circles) are shown. Long distances often need to be stretched, because the curvature of the manifold gives them a smaller value than expected. On the other hand, shorter distances are corrupted by noise (noise is negligible for long distances); this makes them longer than expected.

Assuming F_λ has a null derivative, the gradients of $E^i_{\text{CCA-u}}$ and $E^i_{\text{CCA-p}}$ are:

$$\begin{aligned} \nabla_{\mathbf{x}(j)} E^i_{\text{CCA-u}} &= \quad 2(d_{\mathbf{y}} - d_{\mathbf{x}}) F_\lambda(d_{\mathbf{x}}) \frac{\mathbf{x}(j) - \mathbf{x}(i)}{d_{\mathbf{x}}} \quad , & (4.90) \\ \nabla_{\mathbf{x}(j)} E^i_{\text{CCA-p}} &= -4(d^2_{\mathbf{y}} - d^2_{\mathbf{x}}) F_\lambda(d_{\mathbf{x}}) (\mathbf{x}(j) - \mathbf{x}(i)) \quad . & (4.91) \end{aligned}$$

In order to obtain the same error function in both cases around $d_{\mathbf{x}} = d_{\mathbf{y}}$, Hérault equates the second derivatives of $E^i_{\text{CCA-u}}$ and $E^i_{\text{CCA-p}}$ and scales $E^i_{\text{CCA-p}}$ accordingly:

$$E_{\text{CCA}} = \sum_{i=1}^{N} \begin{cases} E^i_{\text{CCA-u}} & \text{if } d_{\mathbf{x}} > d_{\mathbf{y}} \\ \frac{1}{4 d_{\mathbf{y}}(i,j)^2} E^i_{\text{CCA-p}} & \text{if } d_{\mathbf{x}} < d_{\mathbf{y}} \end{cases} \quad . \tag{4.92}$$

Another variant of CCA is also introduced in [186, 188, 187]. In that version, the main change occurs in the weighting function F, which is a balanced compound of two functions. The first one depends on the distance in the embedding space, as for regular CCA, whereas the second has for argument the distance in the data space, as for Sammon's NLM:

$$F_\lambda(d_{\mathbf{x}}, d_{\mathbf{y}}) = (1-\rho)H(\lambda - d_{\mathbf{x}}) + \rho H(\lambda - d_{\mathbf{y}}) \ . \tag{4.93}$$

The additional parameter ρ allows one to tune the balance between both terms. With ρ close to one, local neighborhoods are generally well preserved, but discontinuities can possibly appear (manifold tears). In contrast, with ρ close to zero, the global manifold shape is better preserved, often at the price of some errors in local neighborhoods.

Finally, a method similar to CCA is described in [59]. Actually, this method is more closely related to VQP [46, 45] than to CCA as it is described in [48]. This method relies on the neural gas [135] for the vector quantization and is thus able to work online (i.e., with a time-varying data set).

4.3 Graph distances

Briefly put, graph distances attempt to overcome some shortcomings of spatial metrics like the Euclidean distance. The next subsection introduces both the geodesic and graph distances, explains how they relate to each other, and motivates their use in the context of dimensionality reduction. The subsequent subsections describe three methods of nonlinear dimensionality reduction that use graph distances.

4.3.1 Geodesic distance and graph distance

In order to reduce the dimensionality of highly folded manifolds, algorithms like NLM and CCA extend the purely linear MDS mainly by changing the optimization procedure. Instead of an exact solution computed algebraically, these algorithms use more sophisticated optimization procedures. This gives more freedom in the definition of the error function, for example, by allowing the user to differently weight short and long distances.

Fundamentally, however, the problem of unfolding may be solved from the opposite side. The goal of NLM and CCA is formalized by complex error functions that preserve short distance and allow the stretching of longer ones. But is this not an awkward remedy to the fact that traditional spatial metrics like the Euclidean one are not adapted to distance preservation? Would it be possible, just by changing the metric used to measure the pairwise distances, either to keep the simplicity of metric MDS or to attain better performances? These two directions are explored respectively by Isomap (Subsection 4.3.2) and curvilinear distance analysis (Subsection 4.3.4).

The idea of these two methods results from simple geometrical considerations, as illustrated in Fig. 4.9. That figure shows a curved line, i.e., a

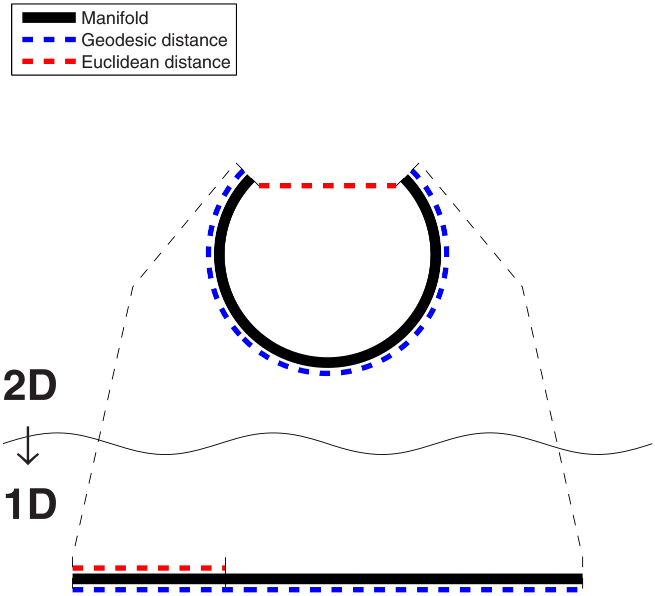


Fig. 4.9. The C curve is a one-manifold embedded in a two-dimensional space. Intuitively, it is expected that the reduction to one dimension unrolls the curve and makes it straight. With this idea in mind, the Euclidean distance cannot be preserved easily, except for very small distances, on so small a scale that the manifold is nearly linear. For larger distances, difficulties arise. The best example consists of measuring the Euclidean distance between the two endpoints of the curve. In the two-dimensional space, this distance is short because the manifold is folded on itself. In the one-dimensional embedding space, the same length proves much smaller than the new distance measured between the endpoints of the unrolled curve. In contrast with the Euclidean distance, the geodesic distance is measured along the manifold. As a consequence, it does not depend as much as the Euclidean metric on a particular embedding of the manifold. In the case of the C curve, the geodesic distance remains the same in both two- and one-dimensional spaces.

one-dimensional manifold embedded in a two-dimensional space. In order to reduce the dimensionality to 1, it is intuitively expected that the curve has to be unrolled. Assuming that very short Euclidean distances are preserved, this means, as a counterpart, that Euclidean longer distances are considerably stretched. For example, the distance between the two endpoints of the C curve in Fig. 4.9 would be multiplied by more than three! Intuitively, such an issue could be addressed by measuring the distance *along the manifold* and not *through the embedding space*, like the Euclidean distance does. With such a metric, the distance depends less on the particular embedding of the manifold. In simpler words, the curvature of the manifold does not modify (or hardly modifies) the value of the distance. The distance along a manifold is usually called the *geodesic distance*, by analogy with curves drawn on the surface of Earth. The geodesic distance can also be interpreted as a railroad distance: trains are forced to follow the track (the manifold). On the other hand, the Euclidean distances may follow straight shortcuts, as a plane flies regardless of roads and tracks.

Formally, the geodesic distance is rather complicated to compute starting with the analytical expression of a manifold. For example, in the case of a one-dimensional manifold $\mathcal{M}$, which depends on a single latent variable x, like the C curve, parametric equations can be written as

$$\mathbb{R} \to \mathcal{M} \subset \mathbb{R}^D : x \mapsto \mathbf{m}(x) = [m_1(x), \ldots, m_D(x)]^T \tag{4.94}$$

and may help to compute the manifold distance as an arc length. Indeed, the arc length l from point $\mathbf{y}(i) = \mathbf{m}(x(i))$ to point $\mathbf{y}(j) = \mathbf{m}(x(j))$ is computed as the integral

$$l = \int_{\mathbf{y}(i)}^{\mathbf{y}(j)} dl = \int_{\mathbf{y}(i)}^{\mathbf{y}(j)} \sqrt{\sum_{k=1}^{D} dm_i^2} = \int_{x(i)}^{x(j)} \|\mathbf{J}_x \mathbf{m}(x)\| dx \ , \tag{4.95}$$

where $\mathbf{J}_x\mathbf{m}(x)$ designates the Jacobian matrix of $\mathbf{m}$ with respect to the parameter x. As x is scalar, the Jacobian matrix reduces to a column vector.

Unfortunately, the situation gets worse for multidimensional manifolds, which involve more than one parameter:

$$\mathbf{m} : \mathbb{R}^P \to \mathcal{M} \subset \mathbb{R}^D, \mathbf{x} = [x_1, \ldots, x_P]^T \mapsto \mathbf{y} = \mathbf{m}(\mathbf{x}) = [m_1(\mathbf{x}), \ldots, m_D(\mathbf{x})]^T \ . \tag{4.96}$$

In this case, several different paths may go from point $\mathbf{y}(i)$ to point $\mathbf{y}(j)$. Each of these paths is in fact a one-dimensional submanifold $\mathcal{P}$ of $\mathcal{M}$ with parametric equations

$$\mathbf{p} : \mathbb{R} \to \mathcal{P} \subset \mathcal{M} \subset \mathbb{R}^P, \quad z \mapsto \mathbf{x} = \mathbf{p}(z) = [p_1(z), \ldots, p_P(z)]^T \ . \tag{4.97}$$

The integral then has to be minimized over all possible paths that connect the starting and ending points:

$$l = \min_{\mathbf{p}(z)} \int_{z(i)}^{z(j)} \|\mathbf{J}_z \mathbf{m}(\mathbf{p}(z))\| dz \ . \tag{4.98}$$

In practice, such a minimization is intractable since it is a functional minimization. Anyway, the parametric equations of $\mathcal{M}$ (and $\mathcal{P}$) are unknown; only some (noisy) points of $\mathcal{M}$ are available.

Although this lack of analytical information is usually considered as a drawback, this situation simplifies the problem of the arc length minimization because it becomes discrete. Consequently, the problem has to be reformulated. Instead of minimizing an arc length between two points on a manifold, the task becomes the following: minimize the length of a path (i.e., a broken line) from point $\mathbf{y}(i)$ to point $\mathbf{y}(j)$ and passing through a certain number of other points $\mathbf{y}(k), \mathbf{y}(l), \ldots$ of the manifold $\mathcal{M}$. Obviously, the Euclidean distance is a trivial solution to this problem: the path directly connects $\mathbf{y}(i)$ to $\mathbf{y}(j)$. However, the idea is that the path should be constrained to follow the underlying manifold, at least approximately. Therefore, a path cannot jump from one point of the manifold to any other one. Restrictions have to be set. Intuitively, they are quite simple to establish. In order to obtain a good approximation of the true arc length, a fine discretization of the manifold is needed. Thus, only the smallest jumps will be permitted. In practice, several simple rules can achieve this goal. A first example is the K-rule, which allows jumping from one point to the K closest other ones, K being some constant. A second example is the ϵ-rule, which allows jumping from one point to all other ones lying inside a ball with a predetermined radius ϵ. See Appendix E for more details about these rules.

Formally, the set of data points associated with the set of allowed jumps constitutes a graph in the mathematical sense of the term. More precisely, a *graph* is a pair $G = (V_N, E)$, where V_N is a finite set of N *vertices* v_i (intuitively: the data points) and E is a set of *edges* (intuitively: the allowed jumps). The edges are encoded as pairs of vertices. If edges are ordered pairs, then the graph is said to be *directed*; otherwise it is *undirected*. In the present case, the graph is *vertex-labeled*, i.e., there is a one-to-one correspondence between the N vertices and the N data points $\mathbf{y}(i)$: $\text{label}(v_i) = \mathbf{y}(i)$. In order to compute the length of paths, the graph has to be vertex-labeled as well as *edge-labeled*, meaning in the present case that edges are given a length. As the length is a numerical attribute, the graph is said to be (edge-)weighted. If the edge length is its Euclidean length, then the graph is said to be Euclidean and

$$\text{label}\left((v_i, v_j)\right) = \|\text{label}(v_i) - \text{label}(v_j)\|_2 = \|\mathbf{y}(i) - \mathbf{y}(j)\|_2 \ . \tag{4.99}$$

A path π in a graph G is an ordered subset of vertices $[v_i, v_j, v_k, \ldots]$ such that the edges $(v_i, v_j), (v_j, v_k), \ldots$ belong to E. In a Euclidean graph, it is sometimes easier to write the same path π by the short-hand notation $\pi = [\mathbf{y}(i), \mathbf{y}(j), \mathbf{y}(k), \ldots]$. The length of π is defined as the sum of the lengths of its constituting edges:

$$\text{length}(\pi) = \text{label}\left((v_i, v_j)\right) + \text{label}\left((v_j, v_k)\right) + \dots \tag{4.100}$$

In order to be considered as a distance, the path length must show the properties of nonnegativity, symmetry, and triangular inequality (see Subsection 4.2.1). Non-negativity is trivially ensured since the path length is a sum of Euclidean distances that already show this property. Symmetry is gained when the graph is undirected. In the case of the K-rule, this means in practice that if point $\mathbf{y}(j)$ belongs to the K closest ones from point $\mathbf{y}(i)$, then the edge from $\mathbf{y}(i)$ to $\mathbf{y}(j)$ as well as the edge from $\mathbf{y}(j)$ to $\mathbf{y}(i)$ are added to E, even if point $\mathbf{y}(i)$ does not belong to the K closest ones from $\mathbf{y}(j)$. The triangular inequality requires that, given points $\mathbf{y}(i)$ and $\mathbf{y}(j)$, the length computed between them must follow the shortest path. If this condition is satisfied, then

$$\text{length}([\mathbf{y}(i), \dots, \mathbf{y}(j)]) \leq \text{length}([\mathbf{y}(i), \dots, \mathbf{y}(k)]) + \text{length}([\mathbf{y}(k), \dots, \mathbf{y}(j)]) \tag{4.101}$$

holds. Otherwise, the concatenated path $[\mathbf{y}(i) \dots, \mathbf{y}(k), \dots, \mathbf{y}(j)]$ would be shorter than length$([\mathbf{y}(i) \dots, \mathbf{y}(j)])$, what is impossible by construction.

At this point, the only remaining problem is how to compute the shortest paths in a weighted graph? Dijkstra [53] designed an algorithm for this purpose. Actually, Dijkstra's procedure solves the *single-source shortest path* problem (SSSP). In other words, it computes the shortest paths between one predetermined point (the source) and all other ones. The *all-pairs shortest path* problem (APSP) is easily solved by repeatedly running Dijkstra's procedure for each possible source [210]. The only requirement of the procedure if the nonnegativity of the edge labels. The result of the procedure is the length of the shortest paths from the source to all other vertices. A representation of the shortest paths can be computed as a direct byproduct. In graph theory, the lengths of the shortest paths are traditionally called *graph distances* and noted

$$\delta(\mathbf{y}(i), \mathbf{y}(j)) = \delta(v_i, v_j) = \min_{\pi} \text{length}(\pi) \ , \tag{4.102}$$

where $\pi = [v_i, \dots, v_j]$.

Dijkstra's algorithm works as follows. The set of vertices V is divided into two subsets such that $V_N = V_{\text{done}} \cup V_{\text{sort}}$ and $\emptyset = V_{\text{done}} \cap V_{\text{sort}}$ where

- V_{done} contains the vertices for which the shortest path is known;
- V_{sort} contains the vertices for which the shortest path is either not known at all or not completely known.

If v_i is the source vertex, then the initial state of the algorithm is $V_{\text{done}} = \emptyset$, $V_{\text{sort}} = V_N$, $\delta(v_i, v_i) = 0$, and $\delta(v_i, v_j) = \infty$ for all $j \neq i$. After the initialization, the algorithm iteratively looks for the vertex v_j with the shortest $\delta(v_i, v_j)$ in V_{sort}, removes it from V_{sort}, and puts it in V_{done}. The distance $\delta(v_i, v_j)$ is then definitely known. Next, all vertices v_k connected to v_j, i.e., such that $(v_j, v_k) \in E$, are analyzed: if $\delta(v_i, v_j) + \text{label}((v_j, v_k)) < \delta(v_i, v_k)$, then the value of $\delta(v_i, v_k)$ is changed to $\delta(v_i, v_j) + \text{label}((v_j, v_k))$ and the candidate shortest path from v_i to v_k becomes $[v_i, \dots, v_j, v_k]$, where $[v_i, \dots, v_j]$ is the shortest path from v_i to v_j. The algorithm stops when $V_{\text{sort}} = \emptyset$. If the graph is not connected, i.e., if there are one or several pairs of vertices that cannot be connected by any path, then some $\delta(v_i, v_j)$ keep an infinite value.

Intuitively, the correctness of the algorithm is demonstrated by proving that vertex v_j in V_{sort} with shortest path $\pi_1 = [v_i, \ldots, v_j]$ of length $\delta(v_i, v_j)$ can be admitted in V_{done}. This is trivially true for v_i just after the initialization. In the general case, it may be assumed that an unexplored path going from the source v_i to vertex v_j is shorter than the path found by Dijkstra's algorithm. Then this hypothetical "shorter-than-shortest" path may be written as $\pi_2 = [v_i, \ldots, v_t, v_u, \ldots, v_j]$, wherein $\pi_3 = [v_i, \ldots, v_t]$ is the maximal (always nonempty) subpath belonging to V_{done} and $\pi_4 = [v_u, \ldots, v_j]$ is the unexplored part of π_2. Consequently, v_u still lies in V_{sort} and $\delta(v_i, v_u)$ has been given its value when vertex v_t was removed from V_{sort}. Thus, the inequalities

$$\text{length}(\pi_3) < \text{length}(\pi_2) < \text{length}(\pi_1) \tag{4.103}$$

hold but contradict the fact that the first path π_1 was chosen as the one with the shortest $\delta(v_i, v_j)$ in V_{sort}.

At this point, it remains to prove that the graph distance approximates the true geodesic distance in an appropriate way. Visually, it seems to be true for the C curve, as illustrated in Fig. 4.10. Formal demonstrations are provided in [19], but they are rather complicated and not reproduced here. The intuitive idea consists of relating the natural Riemannian structure of a smooth manifold $\mathcal{M}$ to the graph distance computed between points of $\mathcal{M}$. Bounds are more easily computed for graphs constructed with ϵ-balls than with K-ary neighborhoods. Unfortunately, these bounds rely on assumptions that are hard to meet with real data sets, especially for K-ary neighborhoods. Moreover, the bounds are computed in the ideal case where no noise pollutes the data.

4.3.2 Isomap

Isomap [179, 180] is the simplest NLDR method that uses the graph distance as an approximation of the geodesic distance. Actually, the version of Isomap described in [180] is closely related to Torgerson's classical metric MDS (see Subsection 4.2.2). The only difference between the two methods is the metric used to measure the pairwise distances: Isomap uses graph distances instead of Euclidean ones in the algebraical procedure of metric MDS. Just by introducing the graph distance, the purely linear metric MDS becomes a nonlinear method. Nevertheless, it is important to remind that the nonlinear capabilities of Isomap are *exclusively* brought by the graph distance. By comparison, methods like Sammon's NLM (Subsection 4.2.3) and Hérault's CCA (Subsection 4.2.4) are built on inherently nonlinear models of data, independently of the chosen metric. However, Isomap keeps the advantage of reducing the dimensionality with a simple, fast, and direct algebraical manipulation. On

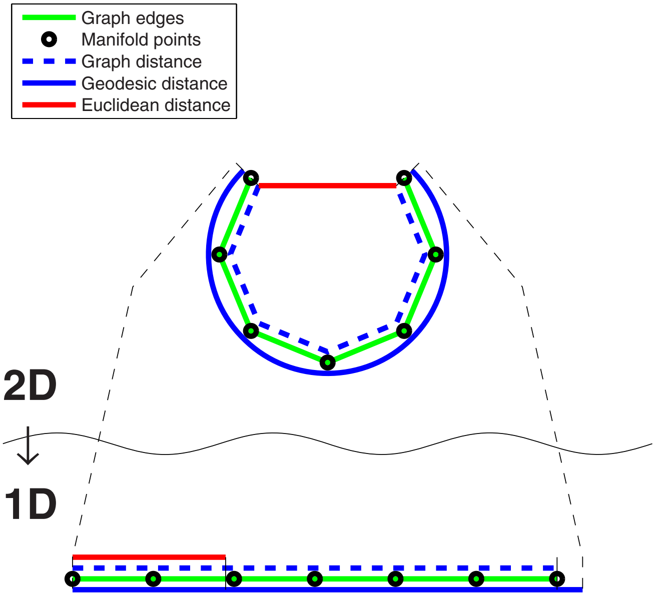


Fig. 4.10. The same C curve as in Fig. 4.9. In this case, the manifold is not available: only some points are known. In order to approximate the geodesic distance, vertices are associated with the points and a graph is built. The graph distance can be measured by summing the edges of the graph along the shortest path between both ends of the curve. That shortest path can be computed by Dijkstra's algorithm. If the number of points is large enough, the graph distance gives a good approximation of the true geodesic distance.

the other hand, Sammon's NLM and Hérault's CCA, which use specific optimization procedures like gradient descent, are much slower.

Although Isomap greatly improves metric MDS, it inherits one of its major shortcomings: a very rigid model. Indeed, the model of metric MDS is restricted to the projection onto a hyperplane. Moreover, the analytical developments behind metric MDS rely on the particular form of the Euclidean distance. In other words, this means that the matrix of pairwise distances $\mathbf{D}$ handled by metric MDS must ideally contain Euclidean distances measured between points lying on a hyperplane. Hence, if the distances in $\mathbf{D}$ are not Euclidean, it is implicitly assumed that the replacement metric yields distances that are equal to Euclidean distances measured in some transformed hyperplane. Otherwise, the conditions stated in the metric MDS model are no longer fulfilled.

In the case of Isomap, the Euclidean distances are replaced with the graph distances when computing the matrix $\mathbf{D}$. For a theoretical analysis, however, it is easier to assume that the graph distances perfectly approximate the true geodesic distances. Therefore, data agree with the model of Isomap if the pairwise geodesic distances computed between points of the P-manifold to be embedded can be mapped to pairwise Euclidean distances measured in a P-dimensional Euclidean space. A manifold that fulfills that condition is called a *developable P-manifold* [111, 31]. (The terminology "Euclidean manifold" in [120, 115] seems to be misleading and encompasses a broader class of manifolds.) In this book, a more restrictive definition is proposed: a P-manifold is developable if and only if a diffeomorphism between the P-manifold and a convex subset of the P-dimensional Euclidean space exists in such a way that geodesic distances are mapped to Euclidean distances by the identity mapping. For the sake of simplicity, it is useful to assume that a developable manifold has parametric equations of the form $\mathbf{y} = \mathbf{m}(\mathbf{x})$ and that the isometry $\delta(\mathbf{y}(i), \mathbf{y}(j)) = \|\mathbf{x}(i) - \mathbf{x}(j)\|_2$ holds. This assumption is justified by the fact that a developable manifold can always be parameterized in that form and, additionally, that the purpose of Isomap is precisely to retrieve the latent vector $\mathbf{x}$ starting from the knowledge of $\mathbf{y}$.

Developable manifolds have nice properties. For example, a straightforward consequence of the isometry between geodesic distances in a developable P-manifold and Euclidean distances in $\mathbb{R}^P$ is the following: in a developable manifold, the Pythagorean theorem is valid for geodesic distances. In practice this means that the geodesic distance between points $\mathbf{y}(i) = \mathbf{m}(\mathbf{x}(i))$ and $\mathbf{y}(j) = \mathbf{m}(\mathbf{x}(j))$ can be computed as follows:

$$\begin{aligned}
\delta^2(\mathbf{y}(i), \mathbf{y}(j)) &= \delta^2(\mathbf{m}(\mathbf{x}(i)), \mathbf{m}(\mathbf{x}(j))) \\
&= \|\mathbf{x}(i) - \mathbf{x}(j)\|_2^2 \\
&= \sum_{p=1}^{P} (x_p(i) - x_p(j))^2 \\
&= \sum_{p=1}^{P} \|\mathbf{x}(i) - [x_1(i), \ldots, x_p(j), \ldots, x_P(i)]^T\|_2^2 \\
&= \sum_{p=1}^{P} \delta^2(\mathbf{m}(\mathbf{x}(i)), \mathbf{m}([x_1(i), \ldots, x_p(j), \ldots, x_P(i)]^T)) \ .
\end{aligned}$$

Therefore, in a developable manifold, geodesic distances can be computed componentwise, either in the latent Euclidean space of the manifold space or in the D-dimensional embedding space.

Another very nice property of developable manifolds is that the computation of geodesic distances does not require a minimization as for other more generic manifolds. Indeed, it is easy to see that

$$\begin{aligned}\delta(\mathbf{y}(i),\mathbf{y}(j)) &= \|\mathbf{x}(i)-\mathbf{x}(j)\|_2\\ &= \|\mathbf{x}(i)-(\mathbf{x}(i)+\alpha(\mathbf{x}(j)-\mathbf{x}(i)))\|_2\\ &\qquad+\|(\mathbf{x}(i)+\alpha(\mathbf{x}(j)-\mathbf{x}(i)))-\mathbf{x}(j)\|_2\\ &= \delta(\mathbf{m}(\mathbf{x}(i)),\mathbf{m}(\mathbf{x}(i)+\alpha(\mathbf{x}(j)-\mathbf{x}(i))))\\ &\qquad+\delta(\mathbf{m}(\mathbf{x}(i)+\alpha(\mathbf{x}(j)-\mathbf{x}(i))),\mathbf{m}(\mathbf{x}(j)))\ ,\end{aligned} \tag{4.104}$$

where α is a real number between 0 and 1. These equalities simply demonstrate that the shortest geodesic between $\mathbf{y}(i)$ and $\mathbf{y}(j)$ is the image by $\mathbf{m}$ of the line segment going from $\mathbf{x}(i)$ to $\mathbf{x}(j)$: all points $\mathbf{m}(\mathbf{x}(i)+\alpha(\mathbf{x}(j)-\mathbf{x}(i)))$ on that segment must also lie on the shortest path. Therefore, in the case of a developable manifold, the path $\mathbf{p}(z)$ in Eq. (4.97) can be written as

$$\mathbf{p}:[0,1]\subset\mathbb{R}\to\mathbb{R}^P, z\mapsto\mathbf{x}=\mathbf{p}(z)=\mathbf{x}(i)+z(\mathbf{x}(j)-\mathbf{x}(i))\ , \tag{4.105}$$

and the minimization in Eq. (4.98) may be dropped.

With the last property in mind (Eq. (4.104)) and knowing that $\mathbf{p}(z)=\mathbf{x}(i)+z(\mathbf{x}(j)-\mathbf{x}(i))$, we find that

$$\begin{aligned}\|\mathbf{x}(i)-\mathbf{x}(j)\|_2 &= \delta(\mathbf{y}(i),\mathbf{y}(j))\\ \int_0^1\|\mathbf{J}_z\mathbf{p}(z)\|_2dz &= \int_0^1\|\mathbf{J}_z\mathbf{m}(\mathbf{p}(z))\|_2dz\\ &= \int_0^1\|\mathbf{J}_{\mathbf{p}(z)}\mathbf{m}(\mathbf{p}(z))\mathbf{J}_z\mathbf{p}(z)\|_2dz\ .\end{aligned} \tag{4.106}$$

Therefore, the equality $\|\mathbf{J}_z\mathbf{p}(z)\|_2=\|\mathbf{J}_{\mathbf{p}(z)}\mathbf{m}(\mathbf{p}(z))\mathbf{J}_z\mathbf{p}(z)\|_2$ holds. This means that the Jacobian of a developable manifold must be a D-by-P matrix whose columns are orthogonal vectors with unit norm (a similar reasoning is developed in [209]). Otherwise, the norm of $\mathbf{J}_z\mathbf{p}(z)$ cannot be preserved. More precisely, the Jacobian matrix can be written in a generic way as

$$\mathbf{J}_\mathbf{x}\mathbf{m}(\mathbf{x})=\mathbf{Q}\mathbf{V}(\mathbf{x})\ , \tag{4.107}$$

where $\mathbf{Q}$ is a constant orthonormal matrix (a rotation matrix in the D-dimensional space) and $\mathbf{V}(\mathbf{x})$ a D-by-P matrix with unit-norm columns and only one nonzero entry per row. The last requirement ensures that the columns of $\mathbf{V}(\mathbf{x})$ are always orthogonal, independently of the value of $\mathbf{x}$.

Because of the particular form of its Jacobian matrix, a developable P-manifold embedded in a D-dimensional space can always be written with the following “canonical” parametric equations:

$$\mathbf{y}=\mathbf{Q}\,\mathbf{f}(\mathbf{x})=\mathbf{Q}\begin{bmatrix}f_1(x_{1\le p\le P})\\ \vdots\\ f_d(x_{1\le p\le P})\\ \vdots\\ f_D(x_{1\le p\le P})\end{bmatrix}\ , \tag{4.108}$$

where $\mathbf{Q}$ is the same as above, $\mathbf{J}_{\mathbf{x}}\mathbf{f}(\mathbf{x}) = \mathbf{V}(\mathbf{x})$, and $f_1, \ldots, f_D$ are constant, linear, or nonlinear continuous functions from $\mathbb{R}$ to $\mathbb{R}$, with $x_{1 \leq p \leq P}$ standing for one of the latent variables [120]. Hence, if $\mathbf{Q}$ is omitted, a manifold is developable when the parametric equation of each coordinate in the D-dimensional space depends on at most a single latent variable x_k.

Of course, the above development states the conditions that are necessary from a generative point of view. This means that (i) a manifold whose parametric equations fulfill those conditions is developable and (ii) the latent variable can be retrieved. However, within the framework of nonlinear dimensionality reduction, generative models are seldom used. In practice, the main question consists of determining when a given manifold is developable, without regards to the actual parametric equations that generate it. In other words, the actual latent variables do not matter, but it must be checked that parametric equations fulfilling the above conditions *could* exist. If the answer is yes, then data respect the model of Isomap and the "canonical" latent variables could be retrieved. From this non-generative point of view, some conditions may be relaxed in Eq. (4.108): the columns of $\mathbf{Q}$ no longer need to be orthonormal and the sum

$$\sum_{p,d=1}^{P,D} (\mathbf{J}_{\mathbf{x}}\mathbf{f}(\mathbf{x}))^2_{d,p} = \sum_{p,d=1}^{P,D} v^2_{d,p}(\mathbf{x}) = \sum_{p,d=1}^{P,D} \left(\frac{\partial f_d(x_{1 \leq p \leq P})}{\partial x_p} \right)^2$$

may be different from P. Only the conditions on the functions $f_d(x_{1 \leq p \leq P})$ remain important, in order to obtain a Jacobian matrix with orthogonal columns. This milder set of conditions allows one to recover the latent variables up to a scaling.

Visually, in a three-dimensional space, a manifold is developable if it looks like a curved sheet of paper. Yet simple manifolds like (a piece of) a hollow sphere are *not* developable. Moreover, no holes may exist on the sheet (due to the convexity of the latent space); an example that does not fulfill this condition is studied in Subsection 6.1.1.

Embedding of data set

Isomap follows the same procedure as metric MDS (Subsection 4.2.2); the only difference is the metric in the data space, which is the graph distance. In order to compute the latter, data must be available as coordinates stored in matrix $\mathbf{Y}$ as usual. Figure 4.11 shows a simple procedure that implements Isomap. It is noteworthy that matrix $\mathbf{S}$ is not guaranteed to be positive semidefinite after double centering [78], whereas this property holds in the case of classical metric MDS. This comes from the fact that graph distances merely approximate the true geodesic distances. Nevertheless, if the approximation is good (see more details in [19]), none or only a few eigenvalues of low magnitude should be negative after double centering. Notice, however, that care must be taken if the eigenvalues are used for estimating the intrinsic dimensionality, especially

1. Build a graph with either the K-rule or the ϵ-rule.
2. Weight the graph by labeling each edge with its Euclidean length.
3. Compute all pairwise graph distances with Dijkstra's algorithm, square them, and store them in matrix $\mathbf{D}$.
4. Convert the matrix of distances $\mathbf{D}$ into a Gram matrix $\mathbf{S}$ by double centering.
5. Once the Gram matrix is known, compute its spectral decomposition $\mathbf{S} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$.
6. A P-dimensional representation of $\mathbf{Y}$ is obtained by computing the product $\hat{\mathbf{X}} = \mathbf{I}_{P\times N}\mathbf{\Lambda}^{1/2}\mathbf{U}^T$.

Fig. 4.11. Isomap algorithm.

if the goal is to normalize them in the same way as in Subsection 2.4.3. In that case, it is better to compute "residual variances", as advised in [180]:

$$\sigma_P^2 = 1 - r_i^2 j(d_{\hat{\mathbf{x}}}(i,j), \delta_{\mathbf{y}}(i,j)) \ , \tag{4.109}$$

where $r_i^2 j$ denotes the correlation coefficient over indices i and j. When plotting the evolution of σ_P^2 w.r.t. P, the ideal dimensionality is the abscissa of the "curve elbow", i.e., the lowest value of P such that σ_P^2 is close enough to zero and does not significantly decrease anymore. Another way to estimate the right embedding dimensionality consists in computing the MDS objective function w.r.t. P:

$$E_{\text{MDS}} = \sum_{i,j=1}^{N} (s_{\mathbf{y}}(i,j) - \langle \hat{\mathbf{x}}(i) \cdot \hat{\mathbf{x}}(j) \rangle)^2 \ , \tag{4.110}$$

where $s_{\mathbf{y}}(i,j)$ is computed from the matrix of graph distances after double centering and $\hat{\mathbf{x}}(i)$ is the ith column of $\hat{\mathbf{X}} = \mathbf{I}_{P\times N}\mathbf{\Lambda}^{1/2}\mathbf{U}^T$. Here also an elbow indicates the right dimensionality.

A MATLAB® package containing the above procedure is available at `http://isomap.stanford.edu/`. A C++ implementation can also be downloaded from `http://www.ucl.ac.be/mlg/`. The parameters of Isomap are the embedding dimensionality P and either the number of neighbors K or the radius ϵ, depending on the rule chosen to build the data graph. Space complexity of Isomap is $\mathcal{O}(N^2)$, reflecting the amount of memory required to store the pairwise geodesic distances. Time complexity is mainly determined by the computation of the graph distances; using Dijkstra's algorithm, this leads to $\mathcal{O}(N^2 \log N)$. The EVD decomposition in the MDS-like step is generally fast when using dedicated libraries or MATLAB®.

Embedding of test set

Like Sammon's NLM, Isomap is not provided with an interpolation procedure for test points. From the point of view of statistical data analysis, the generalization to new points is generally useless: data are gathered at once and processed offline.

Nevertheless, it is not difficult to adapt the interpolation procedure of metric MDS described in Subsection 4.2.2. For this purpose, each test point $\mathbf{y}$ is temporarily "grafted" on the graph, by connecting it to the K closest data points or to all data points lying in an ϵ-ball centered on the test point. Then the geodesic distances from the test point to all data points can be computed by running Dijkstra's algorithm with the test point as source vertex; the obtained distances are stored in the column vector $\delta = [\delta(\mathbf{y}, \mathbf{y}(i))]_{1 \leq i \leq N}$. A more efficient way to obtain δ consists of taking advantage of the graph distances that have already been computed in the learning phase between the neighbors of the test point $\mathbf{y}$ and all other data points [113]. More precisely, it can be written that

$$\delta(\mathbf{y}, \mathbf{y}(i)) = \min_{j \in \mathcal{N}(\mathbf{y})} d(\mathbf{y}, \mathbf{y}(j)) + \delta(\mathbf{y}(j), \mathbf{y}(i)) \ , \tag{4.111}$$

where the set $\mathcal{N}(\mathbf{y})$ contains the indices of the neighbors of $\mathbf{y}$.

Next, the distances in δ are transformed into scalar products in the column vector $\mathbf{s}$ by double centering. Finally, the low-dimensional representation of $\mathbf{y}$ is computed as the product $\hat{\mathbf{x}} = \mathbf{I}_{P \times N} \mathbf{\Lambda}^{-1/2} \mathbf{U}^T \mathbf{s}$. More details can be found in [16].

Example

Figure 4.12 shows the two-dimensional embeddings of the two data sets (Fig. 1.4) described in Section 1.5. The use of geodesic distances instead of

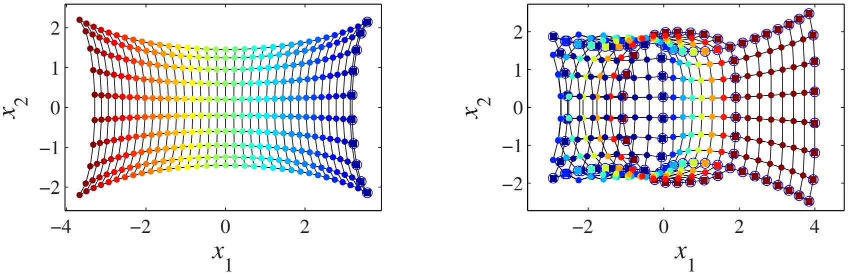


Fig. 4.12. Two-dimensional embeddings of the "Swiss roll" and "open box" data sets (Fig. 1.4), found by Isomap.

Euclidean ones allows Isomap to perform much better than metric MDS (see Fig. 4.2). In the case of the two benchmark manifolds, the graphs used for approximating the geodesic distances are built with the ϵ-rule; the value of ϵ is somewhat larger than the distance between the three-dimensional points of the open box data set shown in Fig. 1.4. This yields the regular graphs displayed in the figure. As a direct consequence, the graph distances are computed in the same way as city-block distances (see Subsection 4.2.1), i.e., by summing the lengths of perpendicular segments.

With this method, the Swiss roll is almost perfectly unrolled (corners of the rolled-up rectangle seem to be stretched outward). This result is not surprising since the Swiss roll is a developable manifold. The first six eigenvalues confirm that two dimensions suffice to embed the Swiss roll with Isomap:

$$[\lambda_n]_{1\leq n\leq N} = [1560.1, 494.7, 112.2, 70.1, 43.5, 42.4, \ldots] \ . \tag{4.112}$$

The first two eigenvalues clearly dominate the others. By the way, it must be remarked that in contrast to classical metric MDS, the eigenvalues λ_n with $n > D$ do not vanish completely (the last ones can even be negative, as mentioned above). This phenomenon is due to the fact that graph distances only approximate the true geodesic ones.

On the other hand, the open box is not a developable manifold and Isomap does not embed it in a satisfying way. The first six eigenvalues found by Isomap are

$$[\lambda_n]_{1\leq n\leq N} = [1099.6, 613.1, 413.0, 95.8, 71.4, 66.8, \ldots] \ . \tag{4.113}$$

As can be seen, the first three eigenvalues dominate the others, just as they do with metric MDS. Hence, like MDS, Isomap does not succeed in detecting that the intrinsic dimensionality of the box is two. Visually, two faces of the open box are still superposed: neighborhood relationships between data points on these faces are not correctly rendered.

Classification

Isomap is an offline batch method, working with an exact algebraical optimization. As Isomap operates like metric MDS, by decomposing a Gram matrix into eigenvalues and eigenvectors, it is often qualified to be a spectral method [31, 12]. In the literature, Isomap is described without any preprocessing of data such as vector quantization (see, however, [120]). Instead, a variant of Isomap (see Subsection 4.3.2 ahead) works by selecting a random subset of the available data points, which are called anchors or landmarks.

Isomap relies on a nonlinear model. Actually, if the Euclidean distance can be considered as a "linear" metric, then the ability of Isomap to embed nonlinear manifolds results only from the use of the graph distance; other parts of the method, such as the underlying model of the optimization procedure, stem from classical metric MDS and remain purely linear. Hence, the data model of Isomap is hybrid: the approximation of geodesic distances by graph distances is discrete whereas the subsequent MDS-like step can be considered to be continuous.

Since Isomap shares the same model type and optimization procedure as PCA and metric MDS, it also inherits its integrated estimation of the intrinsic dimensionality. The latter can be evaluated just by looking at the eigenvalues of $\mathbf{D}$. (In [179, 180] and in the Stanford implementation of Isomap, eigenvalues are transformed into so-called residual variances, which are defined as one minus the correlation coefficient between the geodesic distances in the initial space and the Euclidean distances in a P-dimensional final space.)

Just as for PCA and MDS, embeddings in spaces of increasing dimensionality can be obtained at once, in an incremental way.

Advantages and drawbacks

By comparison with PCA and metric MDS, Isomap is much more powerful. Whereas the generative model of PCA and metric MDS is designed for linear submanifolds only, Isomap can handle a much wider class of manifolds: the developable manifolds, which may be nonlinear. For such manifolds, Isomap reduces the dimensionality without any loss.

Unfortunately, the class of developable manifolds is far from including all possible manifolds. When Isomap is used with a nondevelopable manifold, it suffers from the same limitations as PCA or metric MDS applied to a nonlinear manifold.

From a computational point of view, Isomap shares the same advantages and drawbacks as PCA and metric MDS. It is simple, works with simple algebraic operations, and is guaranteed to find the global optimum of its error function in closed form.

In summary, Isomap extends metric MDS in a very elegant way. However, the data model of Isomap, which relies on developable manifolds, still remains too rigid. Indeed, when the manifold to be embedded is not developable, Isomap yields disappointing results. In this case, the guarantee of determining a global optimum does not really matter, since actually the model and its associated error function are not appropriate anymore.

Another problem encountered when running Isomap is the practical computation of the geodesic distances. The approximations given by the graph distances may be very rough, and their quality depends on both the data (number of points, noise) and the method parameters (K or ϵ in the graph-building rules). Badly chosen values for the latter parameters may totally jeopardize the quality of the dimensionality reduction, as will be illustrated in Section 6.1.

Variants

When the data set becomes too large, Isomap authors [180] advise running the method on a subset of the available data points. Instead of performing a vector quantization like for CCA, they simply select points at random in the available data. Doing this presents some drawbacks that appear particularly critical, especially because Isomap uses the graph distance (see examples in Section 6.1).

A slightly different version of Isomap also exists (see [180]) and can be seen as a compromise between the normal version of Isomap and the economical version described just above. Instead of performing Isomap on a randomly chosen subset of data, Isomap is run with all points but only on a subset of all distances. For this purpose, a subset of the data points is chosen and only the geodesic distances from these points to all other ones are computed. These particular points are called *anchors* or *landmarks*. In summary, the normal or "full" version of Isomap uses the N available points and works with an N-by-N distance matrix. The economical or "light" version of Isomap uses a subset of $M < N$ points, yielding a M-by-M distance matrix. The intermediate version uses $M < N$ anchors and works with a rectangular M-by-N distance matrix. Obviously, an adapted MDS procedure is required to find the embedding (see [180] for more details) in the last version.

An online version of Isomap is detailed in [113]. Procedures to update the neighborhood graph and the corresponding graph distances when points are removed from or added to the data set are given, along with an online (but approximate) update rule of the embedding.

Finally, it is noteworthy that the first version of Isomap [179] is quite different from the current one. It relies on data resampling (graph vertices are a subset of the whole data set) and the graph is built with a rule inspired from topology-representing networks [136] (see also Appendix E). Next, graph distances are compute with Floyd's algorithm (instead of Dijkstra's one, which is more efficient), and the embedding is obtained with nonmetric MDS instead of classical metric MDS. Hence this previous version is much closer to geodesic NLM than the current one is.

4.3.3 Geodesic NLM

Sammon's nonlinear mapping (NLM) is mostly used with the Euclidean distance, in the data space as well as in the embedding space. Nevertheless, as mentioned in Subsection 4.2.3, nothing forbids the user to choose another metric, at least in the data space. Indeed, in the embedding space, the simple and differentiable formula of the Euclidean distance helps to deduce a not-too-complicated update rule for the optimization of the stress function. So, why not create a variant of NLM that uses the graph distance in the data space? Isomap (see Subsection 4.3.2) and CDA (see Subsection 4.3.4) follow the same idea by modifying, respectively, the metric MDS (see Subsection 4.2.2) and CCA (see Subsection 4.2.4). Strangely enough, very few references to such variant of NLM can be found in the literature (see, however, [117, 150, 58]). NLM using geodesic distances is here named GNLM (geodesic NLM), according to [117, 121, 58], although the method described in [58] is more related to CDA.

Embedding of data set

The embedding of the data set simply follows the procedure indicated in Subsection 4.2.3. The only difference regards the distance in the data space, which is the graph distance introduced in Subsection 4.3.1. Hence, Sammon's stress can be rewritten as

$$E_{\text{GNLM}} = \frac{1}{c} \sum_{\substack{i=1 \\ i<j}}^{N} \frac{(\delta_{\mathbf{y}}(i,j) - d_{\mathbf{x}}(i,j))^2}{\delta_{\mathbf{y}}(i,j)} \quad , \tag{4.114}$$

where

- $\delta_{\mathbf{y}}(i,j)$ is the graph distance between the ith and jth points in the D-dimensional data space,
- $d_{\mathbf{x}}(i,j)$ is the Euclidean distance between the ith and jth points in the P-dimensional latent space.
- the normalizing constant c is defined as

$$c = \sum_{\substack{i=1 \\ i<j}}^{N} \delta_{\mathbf{y}}(i,j) \quad . \tag{4.115}$$

In order to compute the graph distance, a graph structure must be attached to the initial data set. This can be done by using the same simple rules as in Isomap: each data point $\mathbf{y}(i)$ may be connected either to its K closest neighbors or to all points lying inside an ϵ-ball centered on $\mathbf{y}(i)$. Other ways to build the graph are described in Appendix E.

According to Eq. (C.12), the quasi-Newton update rule that iteratively determines the parameters $x_k(i)$ of E_{NLM} can be written as

$$x_k(i) \leftarrow x_k(i) - \alpha \frac{\frac{\partial E_{\text{GNLM}}}{\partial x_k(i)}}{\left| \frac{\partial^2 E_{\text{GNLM}}}{\partial x_k(i)^2} \right|} \quad , \tag{4.116}$$

where the absolute value is used for distinguishing the minima from the maxima. As for the classical NLM, the step size alpha is usually set between 0.3 and 0.4.

No MATLAB® package is available for the geodesic NLM. However, it is possible to build one quite easily using part of the Isomap archive (`http://isomap.stanford.edu/`), combined with Sammon's NLM provided in the SOM toolbox (`http://www.cis.hut.fi/projects/somtoolbox/`). Functions and libraries taken from Isomap compute the pairwise graph distances, wheres the mapping is achieved by the NLM function from the SOM toolbox. A

C++ implementation of geodesic NLM can also be downloaded from `http://www.ucl.ac.be/mlg/`. Using graph distances in Sammon's NLM requires tuning additional parameters, namely those related to the construction of the data graph before applying Dijkstra's algorithm. These parameters are, for instance, the number of neighbors K (for K-ary neighborhoods) or the radius ϵ (for ϵ-ball neighborhoods). Space complexity of GNLM remains the same as for Euclidean NLM. On the other hand, the time complexity must take into acount the computation of all pairwise graph distances ($\mathcal{O}(N^2 \log N)$). Depending on the data size and the number of iterations, this additional step may eventually become the most time-consuming one.

Embedding of test set

As with the original NLM using Euclidean distance, no generalization of the mapping is possible with GNLM as such. Propositions mentioned in Subsection 4.2.3 may work. However, the use of the geodesic distance brings an additional difficulty, as it does for Isomap. Indeed, the test points must be "grafted" on the graph connecting the data points, in order to compute their geodesic distances to all data points. Moreover, it must be recalled that graph distances only approximate the true geodesic distances. For first-order neighbors, the graph distances equal the Euclidean distances: on a small scale, this is a good approximation. On the other hand, the graph distances to second-order neighbors are computed as a sum of two Euclidean distances; depending on the angle formed by the two segments, the graph distance can overestimate the true geodesic distance. First- and second-order neighbors are exactly the points that are essential to get a good embedding of the test points.

Example

Figure 4.13 shows the two-dimensional embeddings of the two data sets (Fig. 1.4) described in Section 1.5. These results are obtained with step size α set to 0.3. Geodesic NLM performs much better than its Euclidean version (see Fig. 4.4). The Swiss roll is perfectly unrolled, and only two faces of the open box are superposed. The fact that geodesic distances are longer than the corresponding Euclidean ones explains this improvement: their weight in Sammon's stress is lower and the method focuses on preserving short distances.

GNLM also provides better results than Isomap (see Subsection 4.3.2), although the graph distances have been computed in exactly the same way. This is due to the fact that GNLM can embed in a nonlinear way independently from the distance used. In practice, this means that GNLM is expected to perform better than Isomap when the manifold to embed is not developable.

Classification

The classification of GNLM exactly follows that of NLM in Subsection 4.2.3.

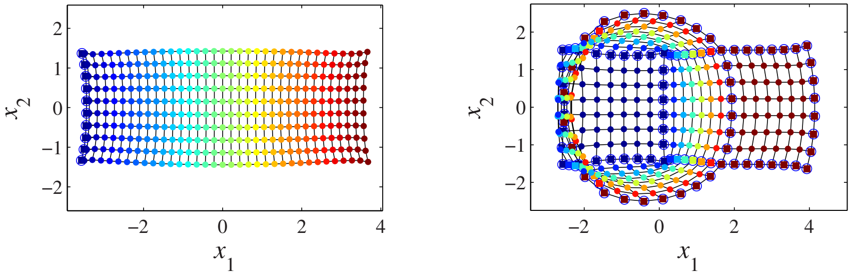


Fig. 4.13. Two-dimensional embeddings of the “Swiss roll” and “open box” data sets (Fig. 1.4), found by GNLM.

Advantages and drawbacks

For a large part, the advantages and drawbacks of GNLM stay more or less the same as for NLM. However, the use of the geodesic distance gives GNLM a better ability to deal with heavily curved manifolds. If GNLM is given a developable P-manifold, then it can reduce the dimensionality from D to P with $E_{\mathrm{GNLM}} \approx 0$ (the stress never vanishes completely since the graph distances are only approximations). In the same situation, the original NLM could yield a much larger value of E_{NLM} and provide disappointing results.

As other methods using the geodesic distance, GNLM may suffer from badly approximated graph distances. Additionally, the graph construction necessary to get them requires adjusting one or several additional parameters.

Variants

Enhancements regarding the optimization procedure of NLM are still applicable to GNLM. See Subsection 4.2.3 for more details.

4.3.4 Curvilinear distance analysis

Curvilinear distance analysis [116, 117] (CDA in short) is a version of CCA (see Subsection 4.2.4) that makes use, among other differences, of the graph distance, in the same way as Isomap extends Torgerson’s classical metric MDS.

Despite their common use of the graph distance and their almost simultaneous publication, CDA and Isomap have been developed independently, in very different contexts. Whereas Isomap aims at extending the application of metric MDS to developable manifolds, keeping its algebraical simplicity, CDA generalizes CCA and provides it with a metric that reinforces its resemblance to an SOM. Considering CDA as a neural network, the prototypes after vector quantization being the neurons, the replacement of the Euclidean distance with the graph distance may be seen as the addition of lateral connections between the neurons. Visually, these lateral synapses are the weighted edges of the graph, very similar to the lattice of an SOM, except that it is not regular.

Even if this may seem counterintuitive, the use of the graph distance actually *removes* lateral connections. In the original CCA, the use of the Euclidean distance may be seen as a degenerate graph distance where all connections are allowed by default. From this point of view, the graph distance, by capturing information about the manifold shape, removes misleading shortcuts that the Euclidean distance could follow. Thus, the graph is less dependent on the manifold embedding than the Euclidean distance.

Because the graph distance is a sum of Euclidean distances, and by virtue of the triangular inequality, graph distances are always longer than (or equal to) the corresponding Euclidean distances. This property elegantly addresses the main issue encountered with distance-preserving DR methods: because manifolds embedded in high-dimensional spaces may be folded on themselves, spatial distances like the Euclidean one are shorter than the corresponding distances measured along the manifold. This issue is usually circumvented by giving an increasing weight to very short distances. In this context, the graph distance appears as a less blind solution: instead of "forgetting" the badly estimated distance, the graph distance enhances the estimation. To some extent, the graph distance "guesses" the value of the distance in the embedding space before embedding. When the manifold to be embedded is exactly Euclidean (see Subsection 4.3.2), the guess is perfect and the error made on long distances remains negligible. In this case, the function F_λ that is weighting the distances in E_{CCA} becomes almost useless.

Nevertheless, the last statement is valid only for developable manifolds. This is a bit too restrictive and for other manifolds, the weighting of distances remains totally useful. It is noteworthy, however, that the use of the graph distance makes F_λ much easier to parameterize.

For a heavily crumpled nondevelopable manifold, the choice of the neighborhood width λ in CCA appears very critical. Too large, it forces CCA to take into account long distances that have to be distorted anyway; the error criterion E_{CCA} no longer corresponds to what the user expects and its minimization no longer makes sense. Too small, the neighborhood width impeaches CCA to access sufficient knowledge about the manifold and the convergence becomes very slow.

On the other hand, in CDA, the use of the graph distance yields a better estimate of the distance after embedding. Distances are longer, and so may λ be longer, too.

Embedding of data set

Formally, the error criterion of CDA is

$$E_{\text{CCA}} = \sum_{i=1}^{N} \begin{cases} E^i_{\text{CCA-u}} & \text{if } d_{\mathbf{x}} > \delta_{\mathbf{y}} \\ \frac{1}{4\delta^2_{i,j}} E^i_{\text{CCA-p}} & \text{if } d_{\mathbf{x}} < \delta_{\mathbf{y}} \end{cases} , \tag{4.117}$$

exactly as for CCA, except for $\delta_{\mathbf{y}}$, which is here the graph distance instead of the Euclidean one.

Assuming F_λ has a null derivative, the gradients of $E^i_{\text{CCA-u}}$ and $E^i_{\text{CCA-p}}$ are

$$\nabla_{\mathbf{x}(j)} E^i_{\text{CCA-u}} = \quad 2(\delta_{\mathbf{y}} - d_{\mathbf{x}}) F_\lambda(d_{\mathbf{x}}) \frac{\mathbf{x}(j) - \mathbf{x}(i)}{d_{\mathbf{x}}} \ , \tag{4.118}$$

$$\nabla_{\mathbf{x}(j)} E^i_{\text{CCA-p}} = -4(\delta^2_{\mathbf{y}} - d^2_{\mathbf{x}}) F_\lambda(d_{\mathbf{x}})(\mathbf{x}(j) - \mathbf{x}(i)) \ , \tag{4.119}$$

leading to the update rule:

$$\mathbf{x}(j) \leftarrow \mathbf{x}(j) + \alpha \begin{cases} \quad \nabla_{\mathbf{x}(j)} E^i_{\text{CCA-u}} & \text{if } d_{\mathbf{x}}(i,j) > \delta_{\mathbf{y}} \\ \frac{1}{4\delta^2_{i,j}} \quad \nabla_{\mathbf{x}(j)} E^i_{\text{CCA-p}} & \text{if } d_{\mathbf{x}}(i,j) < \delta_{\mathbf{y}} \end{cases} . \tag{4.120}$$

Besides the use of the graph distance, another difference distinguishes CCA from CDA. The latter is indeed implemented with a slightly different handling of the weighting function $F_\lambda(d_{\mathbf{x}}(i,j))$. In the original publications about CCA [46, 45, 48], the authors advise using a step function (see Eqs. (4.86) and (4.87)), which is written as

$$F_\lambda(d_{\mathbf{x}}(i,j)) = \begin{cases} 0 & \text{if } \lambda < d_{\mathbf{x}}(i,j) \\ 1 & \text{if } \lambda \geq d_{\mathbf{x}}(i,j) \end{cases} \tag{4.121}$$

From a geometrical point of view, this function centers an open λ-ball around each point $\mathbf{x}(i)$: the function equals one for all points $\mathbf{x}(j)$ inside the ball, and zero otherwise. During the convergence of CCA, the so-called *neighborhood width* λ, namely the radius of the ball, is usually decreasing according a schedule established by the user. But the important point to remark is that the neighborhood width has a unique and common value for all points $\mathbf{x}(i)$. This means that depending on the local distribution of $\mathbf{x}$, the balls will include different numbers of points. In sparse regions, even not so small values of λ could yield empty balls for some points, which are then no longer updated.

This problematic situation motivates the replacement of the neighborhood width λ with a *neighborhood proportion* π, with $0 \leq \pi \leq 1$. The idea consists of giving each point $\mathbf{x}(i)$ an individual neighborhood width $\lambda(i)$ such that the corresponding ball centered on $\mathbf{x}(i)$ contains exactly $\lfloor \pi N \rfloor$ points. This can be achieved easily and exactly by computing the $\lfloor \pi N \rfloor$ closest neighbors of $\mathbf{x}(i)$. However, as mentioned in Appendix F.2, this procedure is computationally demanding and would considerably slow down CCA.

Instead of computing exactly the $\lfloor \pi N \rfloor$ closest neighbors of $\mathbf{x}(i)$, it is thus cheaper to approximate the radius $\mathbf{x}(i)$ of the corresponding ball. Assuming that $\pi \approx 1$ when CDA starts, all the $\lambda(i)$ could be initialized as follows:

$$\lambda(i) \leftarrow \max_j d_{\mathbf{x}}(i,j) \ . \tag{4.122}$$

Next, when CDA is running, each time the point $\mathbf{x}(i)$ is selected, all other points $\mathbf{x}(j)$ lying inside the $\lambda(i)$-ball are updated radially. The number $N(i)$ of updated points gives the real proportion of neighbors, defined as $\pi(i) = N(i)/N$. The real proportion $\pi(i)$, once compared with the desired proportion, helps to adjust $\lambda(i)$. For example, this can be done with the simple update rule

$$\lambda(i) \leftarrow \lambda(i) \sqrt[P]{\frac{\pi}{\pi(i)}} \ , \tag{4.123}$$

which gives $\lambda(i)$ its new value when point $\mathbf{x}(i)$ will be selected again by CDA. In practice, the behavior of the update rule for $\lambda(i)$ may be assessed when CDA is running, by displaying the desired proportion π versus the effective average one $\mu_i(\pi(i))$. Typically, as π is continually decreasing, $\mu_i(\pi(i))$ is always a bit higher than desired. Experimentally, it has been shown that the handling of $F_\lambda(d_\mathbf{x}(i,j))$ in CDA deals with outliers in a rather robust way and avoids some useless tearings that are sometimes observed when using the original CCA with a neighborhood width that is too small.

It is noteworthy that the use of an individual neighborhood width does not complicate the parameter setting of CDA, since all neighborhood widths are guided by a single proportion π.

Gathering all above-mentioned ideas leads to the procedure given in Fig. 4.14. No MATLAB® package is available for CDA. However, it is pos-

1. Perform a vector quantization (see Appendix D) to reduce the size of the data set, if needed.
2. Build a graph with a suitable rule (see Appendix E).
3. Compute all pairwise graph distances $\delta(\mathbf{y}(i), \mathbf{y}(j))$ in the D-dimensional data space.
4. Initialize the P-dimensional coordinates $\mathbf{x}(i)$ of all points, either randomly or on the hyperplane spanned by the first principal components (after a PCA).
5. Initialize the individual neighborhood widths $\lambda(i)$ according to Eq. (4.122). Let q be equal to one.
6. Give the learning rate α and the neighborhood proportion π their scheduled value for epoch number q.
7. Select a point $\mathbf{x}(i)$ in the data set and update all other ones according to Eq. (4.120).
8. Update $\lambda(i)$ according to Eq. (4.123).
9. Return to step 7 until all points have been selected exactly once during the current epoch.
10. Increase q, and return to step 6 if convergence is not reached.

Fig. 4.14. Algorithm implementing curvilinear distances analysis.

sible to build one quite easily using part of the Isomap archive (`http://isomap.stanford.edu/`), combined with CCA provided in the SOM toolbox (`http://www.cis.hut.fi/projects/somtoolbox/`). Functions and libraries taken from Isomap compute the pairwise graph distances, wheres the mapping is achieved by the CCA function from the SOM toolbox. A C++ implementation of CDA can be downloaded from `http://www.ucl.ac.be/mlg/`. Like the geodesic version of Sammon's NLM, CDA involves additional parameters related to the construction of a data graph before applying Dijkstra's algorithm to compute graph distances. These parameters are, for instance, the number of neighbors K or the neighborhood radius ϵ. Space complexity of CDA remains unchanged compared to CCA, whereas time complexity must take into account the application of Dijkstra's algorithm for each graph vertex ($\mathcal{O}(N^2 \log N)$) before starting the iterative core procedure of CCA/CDA.

Embedding of test set

A linear piecewise interpolation can work efficiently if the data set is not too noisy. The interpolation procedure described in [45] for CCA also works for CDA, at least if the neighborhood width is set to a small value; this ensures that Euclidean and graph distances hardly differ on that local scale.

Example

Figure 4.15 shows the two-dimensional embeddings of the two data sets (Fig. 1.4) described in Section 1.5. The graph used to approximate the geodesic distance and shown in the figure is the same as for Isomap and GNLM (ϵ-rule). The main difference between the results of CCA (see Fig. 4.7) and CDA regards the Swiss roll. The use of geodesic distances enables CDA to unroll the Swiss roll and to embed it without superpositions and unnecessary tears.

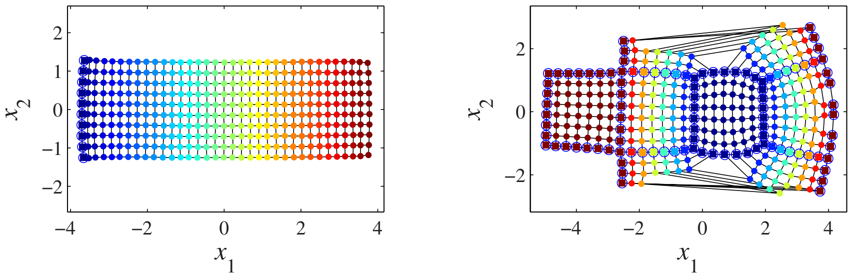


Fig. 4.15. Two-dimensional embeddings of the 'Swiss roll' and 'open box' data sets (Fig. 1.4), found by CDA.

Neighborhood relationships between nearby data points are respected everywhere, except along the tears in the open box. Considering the underlying manifolds, CDA establishes a bijective mapping between their initial and final embeddings.

In the case of the open box, the ability of CCA to deal with a nonlinear manifold is already so good that the use of the graph distance does not improve the result. More difficult examples can be found in Chapter 6.

Classification

Exactly like CCA, CDA is an offline (batch) method, with an optional vector quantization as preprocessing. The data model is nonlinear and discrete; the mapping is explicit. The method works with an approximate optimization procedure. CDA does not include any estimator of the intrinsic dimensionality of data and cannot build embeddings incrementally.

Advantages and drawbacks

CDA keeps all the advantages of CCA. In addition, the parameter tuning is easier for CDA than for CCA. As already mentioned, this is due to the use of the graph distance and to the different handling of the weighting function. Consequently, fewer trials and errors are needed before getting good results. Moreover, CDA converges faster than CCA for heavily crumpled manifolds that are (nearly) developable. In that case, the minimization of E_{CDA} becomes trivial, whereas CCA may encounter difficulties as for any other nonlinear manifold.

Variants

A variant of CDA is described in [58]. Although this method is named geodesic nonlinear mapping, it is more closely related to CCA/CDA than to Sammon's NLM. Actually, this method uses the neural gas [135], a topology-representing network (TRN [136]), instead of a more classical vector quantizer like K-means. This choice enables the method to build the data graph in parallel with the quantization. Next, graph distances are computed in the TRN. The subsequent embedding procedure is driven with an objective function simmilar to the one of CCA/CDA, except that F is an exponentially decaying function of the neighborhood rank, exactly as in the neural gas. The resulting method closely resembles Demartines and Hérault's VQP [46, 45], though it remains a batch procedure.

4.4 Other distances

The previous two sections deal with natural distance measures, in the sense that mainly geometrical arguments motivate the use of both spatial and graph distances. This section introduces NLDR methods that rely on less intuitive ideas. The first one is kernel PCA, which is closely related to metric MDS and other spectral methods. In this case, the methods directly stem from mathematical considerations about kernel functions. These functions can transform a matrix of pairwise distances in such a way that the result can still be interpreted as distances and processed using a spectral decomposition as in metric MDS. Unfortunately, in spite of its elegance, kernel PCA performs rather poorly in practice.

The second method, semi-definite embedding, relies on the same theory but succeeds in combining it with a clear geometrical intuition. This different point of view leads to a much more efficient method.

4.4.1 Kernel PCA

The name of kernel PCA [167] (KPCA) is quite misleading, since its approach relates it more closely to classical metric MDS than to PCA [203]. Beyond the equivalence between PCA and classical metric MDS, maybe this choice can be justified by the fact that PCA is more widely known in the field where KPCA has been developed.

Whereas the changes brought by Isomap to metric MDS were motivated by geometrical consideration, KPCA extends the algebraical properties of MDS to nonlinear manifolds, without regards to their geometrical meaning. The inclusion of KPCA in this chapter about dimensionality reduction by distance-preserving methods is only justified by its resemblance to Isomap: they both generalize metric MDS to nonlinear manifolds in similar ways, although the underlying ideas completely differ.

Embedding of data set

The first idea of KPCA consists of reformulating the PCA into its metric MDS equivalent, or dual form. If, as usual, the centered data points $\mathbf{y}(i)$ are stored in matrix $\mathbf{Y}$, then PCA works with the sample covariance matrix $\hat{\mathbf{C}}_{\mathbf{yy}}$, proportional to $\mathbf{Y}\mathbf{Y}^T$. On the contrary, KPCA works as metric MDS, i.e., with the matrix of pairwise scalar products $\mathbf{S} = \mathbf{Y}^T\mathbf{Y}$.

The second idea of KPCA is to "linearize" the underlying manifold $\mathcal{M}$. For this purpose, KPCA uses a mapping $\phi : \mathcal{M} \subset \mathbb{R}^D \to \mathbb{R}^Q, \mathbf{y} \mapsto \mathbf{z} = \phi(\mathbf{y})$, where Q may be any dimension, possibly higher than D or even infinite. Actually, the exact analytical expression of the mapping ϕ is useless, as will become clear below. As a unique hypothesis, KPCA assumes that the mapping ϕ is such that the mapped data span a linear subspace of the Q-dimensional space, with $Q > D$. Interestingly, KPCA thus starts by increasing the data dimensionality!

Once the mapping ϕ has been chosen, pairwise scalar products are computed for the mapped data and stored in the N-by-N matrix $\mathbf{\Phi}$:

$$\begin{aligned}\boldsymbol{\Phi} &= [\langle \phi(\mathbf{y}(i)) \cdot \phi(\mathbf{y}(j)) \rangle]_{1 \le i,j \le N} && (4.124)\\ &= [\langle \mathbf{z}(i) \cdot \mathbf{z}(j) \rangle]_{1 \le i,j \le N} && (4.125)\\ &= [s_{\mathbf{z}}(i,j)]_{1 \le i,j \le N} \ , && (4.126)\end{aligned}$$

where the shortened notation $s_{\mathbf{z}}(i,j)$ stands for the scalar product between the mapped points $\mathbf{y}(i)$ and $\mathbf{y}(j)$.

Next, according to the metric MDS procedure, the symmetric matrix $\boldsymbol{\Phi}$ has to be decomposed in eigenvalues and eigenvectors. However, this operation will not yield the expected result unless $\boldsymbol{\Phi}$ is positive semidefinite, i.e., when the mapped data $\mathbf{z}(i)$ are centered. Of course, it is difficult to center $\mathbf{z}$ because the mapping ϕ is unknown. Fortunately, however, centering can be achieved in an implicit way by performing the double centering on $\boldsymbol{\Phi}$.

Assuming that $\mathbf{z}$ is already centered but $\mathbf{z}'$ is not, it can be written in the general case

$$\begin{aligned}s_{\mathbf{z}'}(i,j) &= \langle \mathbf{z}(i) + \mathbf{c} \cdot \mathbf{z}(j) + \mathbf{c} \rangle\\ &= \langle \mathbf{z}(i) \cdot \mathbf{z}(j) \rangle + \langle \mathbf{z}(i) \cdot \mathbf{c} \rangle + \langle \mathbf{c} \cdot \mathbf{z}(j) \rangle + \langle \mathbf{c} \cdot \mathbf{c} \rangle\\ &= s_{\mathbf{z}}(i,j) + \langle \mathbf{z}(i) \cdot \mathbf{c} \rangle + \langle \mathbf{c} \cdot \mathbf{z}(j) \rangle + \langle \mathbf{c} \cdot \mathbf{c} \rangle \ , && (4.127)\end{aligned}$$

where $\mathbf{c}$ is some unknown constant as, for instance, the mean that has previously been subtracted from $\mathbf{z}'$ to get $\mathbf{z}$. Then, denoting μ_i the mean operator with respect to index i, the mean of the jth column of $\mathit{\Phi}$ is

$$\begin{aligned}\mu_i(s_{\mathbf{z}}(i,j)) &= \mu_i(\langle \mathbf{z}(i) \cdot \mathbf{z}(j) \rangle + \langle \mathbf{z}(i) \cdot \mathbf{c} \rangle + \langle \mathbf{c} \cdot \mathbf{z}(j) \rangle + \langle \mathbf{c} \cdot \mathbf{c} \rangle)\\ &= \langle \mu_i(\mathbf{z}(i)) \cdot \mathbf{z}(j) \rangle + \langle \mu_i(\mathbf{z}(i)) \cdot \mathbf{c} \rangle + \langle \mathbf{c} \cdot \mathbf{z}(j) \rangle + \langle \mathbf{c} \cdot \mathbf{c} \rangle\\ &= \langle \mathbf{0} \cdot \mathbf{z}(j) \rangle + \langle \mathbf{0} \cdot \mathbf{c} \rangle + \langle \mathbf{c} \cdot \mathbf{z}(j) \rangle + \langle \mathbf{c} \cdot \mathbf{c} \rangle\\ &= \langle \mathbf{c} \cdot \mathbf{z}(j) \rangle + \langle \mathbf{c} \cdot \mathbf{c} \rangle \ . && (4.128)\end{aligned}$$

Similarly, by symmetry, the mean of the ith row of $\mathit{\Phi}$ is

$$\mu_j(s_{\mathbf{z}}(i,j)) = \langle \mathbf{z}(i) \cdot \mathbf{c} \rangle + \langle \mathbf{c} \cdot \mathbf{c} \rangle \ . \qquad (4.129)$$

The mean of all entries of $\mathit{\Phi}$ is

$$\begin{aligned}\mu_{i,j}(s_{\mathbf{z}}(i,j)) &= \mu_{i,j}(\langle \mathbf{z}(i) \cdot \mathbf{z}(j) \rangle + \langle \mathbf{z}(i) \cdot \mathbf{c} \rangle + \langle \mathbf{c} \cdot \mathbf{z}(j) \rangle + \langle \mathbf{c} \cdot \mathbf{c} \rangle)\\ &= \langle \mu_i(\mathbf{z}(i)) \cdot \mu_j(\mathbf{z}(j)) \rangle + \langle \mu_i(\mathbf{z}(i)) \cdot \mathbf{c} \rangle + \langle \mathbf{c} \cdot \mu_j(\mathbf{z}(j)) \rangle + \langle \mathbf{c} \cdot \mathbf{c} \rangle\\ &= \langle \mathbf{0} \cdot \mathbf{0} \rangle + \langle \mathbf{0} \cdot \mathbf{c} \rangle + \langle \mathbf{c} \cdot \mathbf{0} \rangle + \langle \mathbf{c} \cdot \mathbf{c} \rangle\\ &= \langle \mathbf{c} \cdot \mathbf{c} \rangle \ . && (4.130)\end{aligned}$$

It is easily seen that unknown terms in the right-hand side of Eq. (4.127) can be obtained as the sum of Eqs. (4.128) and (4.129) minus Eq. (4.130). Hence,

$$\begin{aligned}s_{\mathbf{z}}(i,j) &= s_{\mathbf{z}'}(i,j) - \langle \mathbf{z}(i) \cdot \mathbf{c} \rangle - \langle \mathbf{c} \cdot \mathbf{z}(j) \rangle - \langle \mathbf{c} \cdot \mathbf{c} \rangle && (4.131)\\ &= s_{\mathbf{z}'}(i,j) - \mu_i(s_{\mathbf{z}}(i,j)) - \mu_j(s_{\mathbf{z}}(i,j)) + \mu_{i,j}(s_{\mathbf{z}}(i,j)) \ . && (4.132)\end{aligned}$$

Once the double centering has been performed, $\mathbf{\Phi}$ can be decomposed into its eigenvectors and eigenvalues:

$$\mathbf{\Phi} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T \quad . \tag{4.133}$$

As for metric MDS, the P eigenvectors (columns of U) associated with the P largest eigenvalues give the coordinates of all data points along the P principal components in the Q-dimensional space:

$$\hat{\mathbf{X}} = \mathbf{I}_{P\times N}\mathbf{\Lambda}^{1/2}\mathbf{U}^T \quad . \tag{4.134}$$

The final result is thus coordinates on a hyperplane in $\mathbb{R}^Q$. It is noteworthy that in contrast with metric MDS the maximal number of strictly positive eigenvalues is not bounded by $\min(N, D)$. Instead, the bound for KPCA is $\min(N, Q)$, which often equals N since actually Q may be very high, depending on the mapping ϕ.

At this point, it must be remarked that the mapping ϕ is used solely in scalar products. Therefore, ϕ may stay unknown if a kernel function κ directly gives the value of the scalar product starting from $\mathbf{y}(i)$ and $\mathbf{y}(j)$:

$$\kappa : \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R},\ (\mathbf{y}(i), \mathbf{y}(j)) \mapsto \kappa(\mathbf{y}(i), \mathbf{y}(j)) = \langle \phi(\mathbf{y}(i)) \cdot \phi(\mathbf{y}(j)) \rangle \quad . \tag{4.135}$$

Obviously, κ may not be any function: the reformulation as a scalar product implies satisfying some conditions that have been extensively studied in [167]. More precisely, Mercer's theorem of functional analysis (see, e.g., [39]) states that

- if κ is a continuous kernel of a positive integral operator $\mathcal{K}$, written as

$$\mathcal{K} : L_2 \to L_2,\ f \mapsto \mathcal{K}f \quad , \tag{4.136}$$

with

$$(\mathcal{K}f)(\mathbf{v}) = \int \kappa(\mathbf{u}, \mathbf{v}) f(\mathbf{v}) d\mathbf{v} \quad , \tag{4.137}$$

- if $\mathcal{K}$ is positive definite, i.e.,

$$\int f(\mathbf{u})\kappa(\mathbf{u}, \mathbf{v}) f(\mathbf{v}) d\mathbf{u} d\mathbf{v} > 0 \ \text{ if } f \neq 0 \quad , \tag{4.138}$$

then κ can be expanded into a series

$$\kappa(\mathbf{u}, \mathbf{v}) = \sum_{q=1}^{\infty} \lambda_q \phi_q(\mathbf{u}) \phi_q(\mathbf{v}) \tag{4.139}$$

with positive coefficients λ_q (the eigenvalues) and orthogonal functions (the eigenfunctions, instead of eigenvectors)

$$\langle \phi_{q_1} \cdot \phi_{q_2} \rangle = \begin{cases} 0 & \text{if } q_1 \neq q_2 \\ 1 & \text{if } q_1 = q_2 \end{cases} \quad . \tag{4.140}$$

Using Eq. (4.139), it is easy to see that

$$\phi(\mathbf{y}) = \sum_{q=1}^{\infty} \sqrt{\lambda_q} \phi_q(\mathbf{y}) \tag{4.141}$$

is a mapping function into a space where κ acts as the Euclidean scalar product, i.e.,

$$\langle \phi(\mathbf{u}) \cdot \phi(\mathbf{v}) \rangle = \kappa(\mathbf{u}, \mathbf{v}) \ . \tag{4.142}$$

In practice, simple kernels that fulfill Mercer's conditions are, for example,

- polynomial kernels [27]: $\kappa(\mathbf{u}, \mathbf{v}) = (\langle \mathbf{u} \cdot \mathbf{v} \rangle + 1)^p$, where p is some integer;
- radial basis function like Gaussian kernels: $\kappa(\mathbf{u}, \mathbf{v}) = \exp\left(-\frac{\|\mathbf{u}-\mathbf{v}\|^2}{2\sigma^2}\right)$;
- kernels looking like the MLP activation function: $\kappa(\mathbf{u}, \mathbf{v}) = \tanh(\langle \mathbf{u} \cdot \mathbf{v} \rangle + b)$.

The choice of a specific kernel is quite arbitrary and mainly motivated by the hope that the induced mapping ϕ linearizes the manifold to be embedded. If this goal is reached, then PCA applied to the mapped data set should efficiently reveal the nonlinear principal components of the data set.

The "kernel trick" described above plays a key role in a large family of methods called support vector machines [27, 37, 42] (SVMs). This family gathers methods dedicated to numerous applications like regression, function approximation, classification, etc.

Finally, Fig. 4.16 shows how to implement KPCA. No general-purpose MATLAB® function is available on the Internet. However, a simple toy example can be downloaded from `http://www.kernel-machines.org/code/kpca_toy.m`; straightforward adaptations of this script can transform it in a more generic function. Another implementation is available in the SPIDER software package (`http://www.kyb.mpg.de/bs/people/spider/main.`

1. Compute either the matrix $\mathbf{S}$ (scalar products) or the matrix $\mathbf{D}$ (squared Euclidean distances), depending on the chosen kernel.
2. Compute the matrix of kernel values Φ.
3. Double-center Φ:
   - Compute the mean of the rows, the mean of the columns, and the grand mean.
   - Subtract from each entry the mean of the corresponding row and the mean of the corresponding column, and add back the grand mean.
4. Decompose the double-centered Φ into eigenvalues and eigenvectors.
5. A P-dimensional representation of $\mathbf{Y}$ is obtained by computing the product $\hat{\mathbf{X}} = \mathbf{I}_{P \times N} \mathbf{\Lambda}^{1/2} \mathbf{U}^T$.

Fig. 4.16. Kernel PCA algorithm.

html). Otherwise, it is straightforward to write a simple function from scratch or to derive it from MDS using the above pseudo-code. Parameters of KPCA are the embedding dimensionality along with all kernel-related parameters. Last but not least, the choice of a precise kernel proves to be a tedious problem. Assuming that the kernel is an easy-to-compute function, time complexity of KPCA is almost the same as for metric MDS (i.e., $\mathcal{O}(N^3)$ or less, depending on the efficiency of the EVD implementation). Space complexity remains unchanged too.

Embedding of test set

The particular version of the double centering described in Subsection 4.2.2 for MDS also works for KPCA. It is easily adapted to the use of kernel functions.

Example

Figure 4.17 shows the two-dimensional embeddings of the two data sets (Fig. 1.4) described in Section 1.5. Visually, it is clear that KPCA has transformed the two manifolds in a nonlinear way. Unfortunately, the embeddings are rather disappointing: the Swiss roll remains rolled up, and faces of the open box are superposed. Once again, it must be emphasized that KPCA is not motivated by geometrical arguments. It aims at embedding the manifold into a space where an MDS-based projection would be more successful than in the initial space. However, no guarantee is provided that this goal can be reached; one can only hope that the kernel is well chosen and well parameterized. For example, the results in Fig. 4.17 are computed with Gaussian kernels: tuning the width of this kernel proves to be tedious. In case of a bad choice of the kernel, KPCA may increase the embedding dimensionality of the manifold instead of reducing it! This is what exactly happens in Fig. 4.17: with a kernel width of 0.75, the first six eigenvalues are

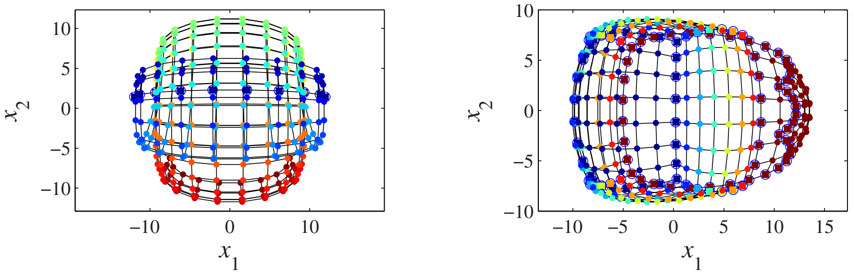


Fig. 4.17. Two-dimensional embeddings of the “Swiss roll” and “open box” data sets (Fig. 1.4), found by KPCA.

$$[\lambda_n]_{1\leq n\leq N} = [0.1517, 0.1193, 0.0948, 0.0435, 0.0416, 0.0345, \ldots] \ , \qquad (4.143)$$

for the Swiss roll, and

$$[\lambda_n]_{1\leq n\leq N} = [0.1694, 0.1084, 0.0890, 0.0544, 0.0280, 0.0183, \ldots] \ , \qquad (4.144)$$

for the open box with a kernel width of 2.5. In other methods using an EVD, like metric MDS and Isomap, the variance remains concentrated within the first three eigenvalues, whereas KPCA spreads it out in most cases. In order to concentrate the variance within a minimal number of eigenvalues, the width of the Gaussian kernel may be increased, but then the benefit of using a kernel is lost and KPCA tends to yield the same result as metric MDS: a linear projection.

Classification

Like metric MDS and Isomap, KPCA is a batch method working offline with simple algebraical operations. As with other spectral methods, KPCA can build projections incrementally, by discarding or keeping eigenvectors. The model of the method is nonlinear thanks to the kernel function that maps the data in an implicit way. The model is also continuous, as for PCA, and the mapping is implicit.

Advantages and drawbacks

By construction, KPCA shares many advantages and drawbacks with PCA and metric MDS. In contrast with these methods, however, KPCA can deal with nonlinear manifolds. And actually, the theory hidden behind KPCA is a beautiful and powerful work of art.

However, KPCA is not used much in dimensionality reduction. The reasons are that the method is not motivated by geometrical arguments and the geometrical interpretation of the various kernels remains difficult. For example, in the case of Gaussian kernels, which is the most widely used, the Euclidean distance is transformed in such a way that long distances yield smaller values than short distances. This is exactly the inverse of what is intuitively expected: in Isomap and CDA, the use of the graph distance was precisely intended to stretch distances!

The main difficulty in KPCA, as highlighted in the example, is the choice of an appropriate kernel along with the right values for its parameters.

4.4.2 Semidefinite embedding

As shown in the previous section, generalizing PCA using the kernel trick as in KPCA proves to be an appealing idea. However, and although theoretical conditions determine the domain of admissible kernel functions, what the theorems do not say is how to choose the best-suited kernel function in some particular case. This shortcoming could disappear if one could learn the best kernel from the data. Semidefinite embedding [196, 198, 195] (SDE), also known as maximum variance unfolding [197] (MVU), follows this approach.

Embedding of data set

Like metric MDS, Isomap and KPCA, SDE implements distance preservation by means of a spectral decomposition. Given a set of N observations, all these methods build the low-dimensional embedding by juxtaposing the dominating eigenvectors of an N-by-N matrix. In metric MDS, pairwise distances are used as they are, and then converted into scalar products by double centering. In Isomap, traditional Euclidean distance are simply replaced with graph distances. In KPCA, Euclidean distances are nonlinearly transformed using a kernel function. In the case of SDE, the transformation of the distances is a bit more complicated than for the former methods. Results of those methods depend on the specific transformation applied to the pairwise distances. Actually, this transformation is arbitrarily chosen by the user: the type of distance (Euclidean/graph) or kernel (Gaussian, etc.) is fixed beforehand. The idea behind SDE is to determine this transformation in a purely data-driven way. For this purpose, distances are constrained to be preserved locally only. Nonlocal distances are free to change and are optimized in such a way that a suitable embedding can be found. The only remaining constraint is that the properties of the corresponding Gram matrix of scalar products are kept (symmetry, positive semidefiniteness), so that metric MDS remains applicable. This relaxation of strict distance preservation into a milder condition of local isometry enables SDE to embed manifolds in a nonlinear way.

In practice, the constraint of local isometry can be applied to smooth manifolds only. However, in the case of a finite data set, a similar constraint can be stated. To this end, SDE first determines the K nearest neighbors of each data point, builds the corresponding graph, and imposes the preservation of angles and distances for all K-ary neighborhoods:

$$\langle (\mathbf{x}_i - \mathbf{x}_j) \cdot (\mathbf{x}_i - \mathbf{x}_k) \rangle = \langle (\mathbf{y}_i - \mathbf{y}_j) \cdot (\mathbf{y}_i - \mathbf{y}_k) \rangle \ . \tag{4.145}$$

This constraint of local isometry reduces to the preservation of pairwise distances only in an "enriched" graph. The latter is obtained by creating a fully connected clique of size $K+1$ out of every data point $\mathbf{y}_i$ and its K nearest neighbors. In other words, after building the undirected graph of K-ary neighborhoods, additional edges are created between the neighbors of each datum, if they do not already exist. If $\mathbf{A} = [a(ij)]_{1 \leq i,j \leq N}$ denotes the N-by-N adjacency matrix of this graph, then the local isometry constraint can be expressed as

$$\|\mathbf{x}_i - \mathbf{x}_j\|_2^2 = \|\mathbf{y}_i - \mathbf{y}_j\|_2^2 \text{ if } A_{ij} = 1 \ . \tag{4.146}$$

As in the case of metric MDS, it can be shown that such an isometry constraint determines the embedding only up to a translation. If the embedding is also constrained to be centered:

$$\sum_{i=1}^{N} \mathbf{x}_i = \mathbf{0} \ , \tag{4.147}$$

then this undeterminacy is avoided.

Subject to the constraint of local isometry, SDE tries to find an embedding of the data set that unfolds the underlying manifold. To illustrate this idea, the Swiss roll (see Section 1.5) is once again very useful. The Swiss roll can be obtained by rolling up a flat rectangle in a three-dimensional space, subject to the same constraint of local isometry. This flat rectangle is also the best two-dimensional embedding of the Swiss roll. As a matter of fact, pairwise Euclidean distances between faraway points (e.g., the corners of the rolled-up rectangle) depend on the embedding. In the three-dimensional space, these distances are shorter than their counterparts in the two-dimensional embedding. Therefore, maximizing long distances while maintaining the shortest ones (i.e., those between neighbors) should be a way to flatten or unfold the Swiss roll. This idea translates into the following objective function:

$$\phi = \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} d_{\mathbf{x}}^2(i,j) \ , \tag{4.148}$$

which should be maximized subject to the above-mentioned constraint of local isometry and where $d_{\mathbf{x}}^2(i,j) = \|\mathbf{x}_i - \mathbf{x}_j\|_2^2$. It is not very difficult to prove that the objective function is bounded, meaning that distances cannot grow infinitely. To this end, graph distances (see Section 4.3) can be used. By construction, graph distances measure the length of the path connecting two data points as the sum of edge lengths. As all edges are subject to the local isometry constraint, it results that

$$\phi \leq \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \delta_{\mathbf{y}}^2(i,j) \ , \tag{4.149}$$

where $\delta_{\mathbf{y}}(i,j)$ is the graph distance between data points $\mathbf{y}_i$ and $\mathbf{y}_j$.

The optimization as stated above is not convex, as it involves maximizing a quadratic form (Eq. (4.148)) subject to quadratic equality constraints (Eq. (4.146)). Fortunately, the formulation of the problem can be simplified by using dot products instead of squared distances. Doing so not only makes the optimization convex but also casts the problem within the framework of classical metric MDS (see Subsection 4.2.2). If $\mathbf{D} = [d_{\mathbf{y}}^2(i,j)]_{1 \leq i,j \leq N}$ denotes the square matrix of squared Euclidean distances and if $\mathbf{S} = [s_{\mathbf{y}}(i,j)]_{1 \leq i,j \leq N}$ with entries $s_{\mathbf{y}}(i,j) = \langle \mathbf{y}_i \cdot \mathbf{y}_j \rangle$ denotes the corresponding matrix of dot products, then the relation

$$d_{\mathbf{y}}^2(i,j) = s_{\mathbf{y}}(i,i) - 2 s_{\mathbf{y}}(i,j) + s_{\mathbf{y}}(j,j) \tag{4.150}$$

holds (see Subsection 4.2.2). If $\mathbf{K} = [s_{\mathbf{x}}(i,j)]_{1 \leq i,j \leq N}$ denotes the symmetric matrix of dot products $\langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle$ in the low-dimensional embedding space, then the constraint in Eq. (4.146) becomes

$$s_{\mathbf{x}}(i,j) = s_{\mathbf{y}}(i,j) \text{ if } a(i,j) = 1 \ . \tag{4.151}$$

Likewise, the centering constraint can be transformed as follows:

$$\mathbf{0} = \sum_{i=1}^{N} \mathbf{x}_i \tag{4.152}$$

$$0 = \langle \mathbf{0} \cdot \mathbf{x}_j \rangle = \sum_{i=1}^{N} \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle \tag{4.153}$$

$$0 = \sum_{j=1}^{N} \langle \mathbf{0} \cdot \mathbf{x}_j \rangle = \sum_{i=1}^{N} \sum_{j=1}^{N} \langle \mathbf{x}_i \cdot \mathbf{x}_j \rangle = \sum_{i=1}^{N} \sum_{j=1}^{N} s_{\mathbf{x}}(i,j) \ . \tag{4.154}$$

Finally, the objective function can also be expressed in terms of dot products using Eqs. (4.150) and (4.154):

$$\begin{aligned} \phi &= \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} d_{\mathbf{x}}^2(i,j) & (4.155) \\ &= \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} s_{\mathbf{x}}(i,i) - 2 s_{\mathbf{x}}(i,j) + s_{\mathbf{x}}(j,j) & (4.156) \\ &= \sum_{i=1}^{N} s_{\mathbf{x}}(i,i) & (4.157) \\ &= \operatorname{tr}(\mathbf{K}) \ , & (4.158) \end{aligned}$$

where $\operatorname{tr}(\mathbf{K})$ denotes the trace of $\mathbf{K}$. At this stage, all constraints are linear with respect to the entries of $\mathbf{K}$ and the optimization problem can be reformulated. The goal of SDE consists of maximizing the trace of some N-by-N matrix $\mathbf{K}$ subject to the following constraints:

- The matrix $\mathbf{K}$ is symmetric and positive semidefinite.
- The sum of all entries of $\mathbf{K}$ is zero (Eq. (4.154)).
- For nonzero entries of the adjacency matrix, the quality $s_{\mathbf{x}}(i,j) = s_{\mathbf{y}}(i,j)$ must hold.

The first two constraints allows us to cast SDE within the framework of classical metric MDS. Compared to the latter, SDE enforces only the preservation of dot products between neighbors in the graph; all other dot products are free to change.

In practice, the optimization over the set of symmetric and positive semidefinite matrices is an instance of semidefinite programming (SDP; see, e.g., [184, 112] and references therein): the domain is the cone of positive semidefinite matrices interesected with hyperplanes (representing the equality constraints) and the objective function is a linear function of the matrix entries. The optimization problem has some useful properties:

- Its objective function is bounded above by Eq. (4.149).
- It is also convex, thus preventing the existence of spurious local maxima.
- The problem is feasible, because **S** is a trivial solution that satisfies all constraints.

Details on SDP are beyond the scope of this book and can be found in the literature. Several SDP toolboxes in C++ or MATLAB® can be found on the Internet. Once the optimal matrix **K** is determined, low-dimensional embedding is obtained by decomposing **K** into eigenvalues and eigenvectors, exactly as in classical metric MDS. If the EVD is written as $\mathbf{K} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T$, then the low-dimensional sample coordinates are computed as

$$\hat{\mathbf{X}} = \mathbf{I}_{P\times N}\boldsymbol{\Lambda}^{1/2}\mathbf{U}^T \quad . \tag{4.159}$$

The SDE algorithm is summarized in Fig. 4.18. A MATLAB® version can

1. If data consist of pairwise distances, then skip step 2 and go directly to step 3.
2. If data consist of vectors, then compute all squared pairwise distances in matrix **D**.
3. Determine the K nearest neighbors of each datum.
4. Build an undirected graph that comprises the complete clique of each K-ary neighborhood.
5. Perform the double centering of **D** in order to obtain **S** (see MDS).
6. Maximize the trace of some N-by-N matrix **K** subject to the following constraints:
   - The matrix **K** is symmetric and positive semidefinite.
   - The sum of all entries of **K** is zero (Eq. (4.154)).
   - For nonzero entries of the graph adjacency matrix, the quality $s_{\mathbf{x}}(i,j) = s_{\mathbf{y}}(i,j)$ must hold.
7. Perform classical metric MDS on the optimized matrix **K** to obtain low-dimensional coordinates:
   - Compute the EVD of **K**: $\mathbf{K} = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T$.
   - The P-dimensional embedding is then given by $\hat{\mathbf{X}} = \mathbf{I}_{P\times N}\boldsymbol{\Lambda}^{1/2}\mathbf{U}^T$.

Fig. 4.18. Procedure achieving semidefinite embedding.

be found at `http://www.seas.upenn.edu/~kilianw/sde/`. The use of this package requires installing at least one additional package, which performs the semidefinite programming step of SDE. This step consumes a large amount of computational resources, regarding memory space as well as running time; so it is advised to run it on a high-end machine. Parameters of SDE are the embedding dimensionality, the number of neighbors K, the type of constraints (equality or inequality), along with all parameters and options involved by the semidefinite programming solver (number of iterations, etc.). See the help of the MATLAB® package for a detailed list of all parameters and options.

Embedding of test set

The Nyström formula that is referred to in [6, 16] (see also Subsection 4.2.2) cannot be used in this case, because the kernel function applied to the Gram matrix and learned from data in the SDP stage remains unknown in closed form.

Example

Figure 4.19 shows the two-dimensional embeddings of the two data sets (Fig. 1.4) described in Section 1.5. These results are obtained with number of neighbors K equal to five (the graph shown in the figure does not correspond to the actual graph built by SDE). Slack variables are off for the Swiss roll (isometry is required) and on for open box (distances may shrink). As can be seen, the Swiss roll is well unfolded: local distances from one point to its neighbors are almost perfectly preserved. For the open box, the embedding is not so satisfying: although distances are allowed to shrink, SDE fails to find a satisfying embedding and faces are superposed. For this manifold, the SDP solver displays a failure message: no embedding with fewer than three dimensions can be built without violating the constraints.

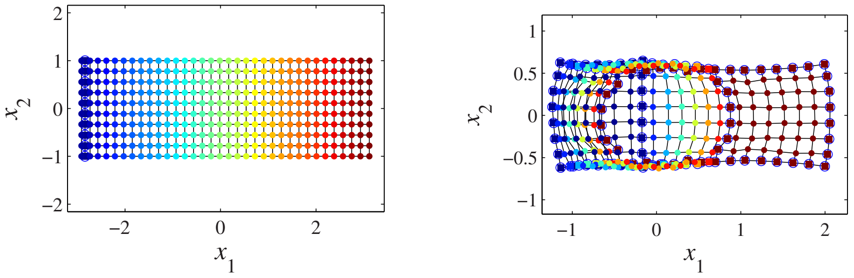


Fig. 4.19. Two-dimensional embeddings of the “Swiss roll” and “open box” data sets (Fig. 1.4), found by SDE.

Classification

SDE is a batch method that processes data offline. It belongs to the family of spectral methods, like metric MDS, Isomap, and KPCA. Exactly as those methods, SDE is provided with an estimator of the data dimensionality and can build embeddings in an incremental way.

In [196, 198, 195], SDE is introduced without vector quantization. As SDE is quite computationally demanding, this preprocessing proves to be very useful in practice.

The mapping resulting from SDE is explicit, and the associated data model is discrete.

Advantages and drawbacks

The constraint of local isometry proposed in SDE is milder than strict isometry, as required in metric MDS and derived methods. As a result, SDE compares favorably to methods based on weighted distance preservation.

SDE can also be seen as a kind of super-Isomap that remedies some shortcomings of the graph distance. Results of Isomap tightly depend on the estimation quality of the geodesic distances: if the data set is sparse, the latter will be poorly approximated by graph distances. Leaving aside any graph construction problem (shortcuts), graph distances are likely to be overestimated in this case, because graph paths are zigzagging. Similarly, Isomap also fails to embed correctly nonconvex manifolds (e.g., manifolds with holes): in this case, some graph paths are longer than necessary because they need to go round the holes. (Actually, both problems are closely related, since a sparse sample from a convex manifold and a dense sample from a manifold with many holes may look quite similar.)

On the down side, SDE proves to be dramatically slow. This comes from the complexity of the SDP step. Finally, SDE cannot process test points.

Variants

A variant of SDE that uses landmark points has been developed and is referred to in [195]. Instead of using all pairwise distances between data points, a rectangular matrix is used instead, which consists of distances from data points to a few landmarks (typically a subset of data). The main goal is to reduce the computational cost of SDE while approximating reasonably well the normal version. A similar principle is used to develop a landmark-based version of Isomap.

Other manifold or graph embedding methods based on semidefinite programming are also developed in [125, 30].

5

Topology Preservation

Overview. This chapter reviews methods that reduce the dimensionality by preserving the topology of data rather than their pairwise distances. Topology preservation appears more powerful than but also more complex to implement than distance preservation. The described methods are separated into two classes according to the kind of topology they use. The simplest methods rely on a predefined topology whereas more recent methods prefer a topology built according to the data set to be re-embedded.

5.1 State of the art

As demonstrated in the previous chapter, nonlinear dimensionality reduction can be achieved by distance preservation. Numerous methods use distances, which are intuitively simple to understand and very easy to compute. Unfortunately, the principle of distance preservation also has a major drawback. Indeed, the appealing quantitative nature of a distance function also makes it very constraining. Characterizing a manifold with distances turns out to support and bolt it with rigid steel beams. In many cases, though, the embedding of a manifold requires some flexibility: some subregions must be locally stretched or shrunk in order to embed them in a lower-dimensional space.

As stated in Section 1.4, the important point about a manifold is its topology, i.e., the neighborhood relationships between subregions of the manifold. More precisely, a manifold can be entirely characterized by giving relative or comparative proximities: a first region is close to a second one but far from a third one. To some extent, distances give too much information: the exact measure of a distance depends not exclusively on the manifold itself but also, for much too large a part, on a given embedding of the manifold. Actually, comparative information between distances, like inequalities or ranks, suffices to characterize a manifold, for any embedding.

Another argument pleads for neighborhood relations: they are considered exclusively *inside* the manifold. On the other hand, most distance functions make no distinction between the manifold and the surrounding empty space. Most often the distance between two points is computed along a straight line and does not take the manifold into account. A first attempt to avoid those annoying "shortcuts" caused by spatial distances comes with the graph distance, studied in Section 4.3.

From a practical point of view, the use of topology raises a serious difficulty. Indeed, how do we characterize the topology of a manifold, knowing that only a few points are available? Without a good description of the topology, its preservation in a different space is impossible! This chapter attempts to answer the above question by studying several existing methods. Briefly put, all these methods translate the qualitative concept of topology into a set of numerical descriptors. Obviously, care must be taken in order for that those descriptors to remain as independent as possible from the initial embedding of the manifold. Otherwise, the same drawback as in the case of distance reappears.

The next two sections review some of the best-known methods that reduce the dimensionality using the principle of topology preservation; they are called topology-preserving methods in short and are classified according to the type of topology they use in the embedding space. Actually, as most of these methods work with a discrete mapping model (see Subsection 2.5.5), the topology is often defined in a discrete way, too. More precisely, such a discrete representation of the topology is usually called a *lattice*. It can be a set of points regularly spaced on a plane or, more formally, a graph (see Section 4.3). The latter is the most generic and flexible mathematical object to discretize a topology: points are associated with graph vertices and their proximity is symbolized by a (weighted) graph edge. Relationships between faraway points may be explicit (an edge with a heavy weight in a weighted graph) or implicit (absence of an edge). In the latter case, the relationships may still be deduced, for example, by graph distances (see Subsection 4.3). A large family of neural networks, called topology-representing networks [136] (TRNs), is specifically devoted to the discrete representation of topologies. Unfortunately, most of those networks do not easily provide a low-dimensional representation of the topology. Hence, other methods must be sought, or TRNs must be adapted.

Section 5.2 deals with methods relying on a predefined lattice, i.e., the lattice or graph is fixed in advance and cannot change after the dimensionality reduction has begun. Kohonen self-organizing map is the best-known example in this category.

Section 5.3 introduces methods working with a *data-driven* lattice, meaning that the shape of the lattice can be modified or is entirely built while the methods are running. If a method reduces the dimensionality without vector quantization (see Subsection 2.5.9), the shape is the sole property of the lattice that can be tuned. On the other hand, when vector quantization is used, even the number of points (vertices) in the lattice (graph) can be adjusted by data.

Additional examples and comparisons of the described methods can be found in Chapter 6.

5.2 Predefined lattice

This section describes two methods, namely the self-organizing map and the generative topographic mapping, that reduce the dimensionality using a predefined lattice. Because the lattice is imposed in advance, the applicability of these methods seems very limited. Indeed, the lattice proposed by most implementations seldom differs from a rectangular or hexagonal grid made of regularly spaced points. As can easily be guessed, very few manifolds fit such a simple shape in practice. Hence, the lattice must often be heavily deformed in order to fit the data cloud; sometimes, it is even folded on itself. However, considering the lattice as the domain of the manifold parameters may help to accept such a restrictive hypothesis.

It is noteworthy, however, that methods working with a predefined lattice allow the easy visualization of neighborhood relationships between labeled data. This is usually realized by displaying the labels in a standardized rectangle or hexagon: each point of the lattice is then represented by its own average or dominant label.

5.2.1 Self-Organizing Maps

Along with the multi-layer perceptron (MLP), the self-organizing map is perhaps the most widely known method in the field of artificial neural networks.

The story began with von der Malsburg's pioneering work [191] in 1973. His project aimed at modeling the stochastic patterns of eye dominance and the orientation preference in the visual cortex. After this rather biologically motivated introduction of self-organization, little interest was devoted to the continuation of von der Malsburg's work until the 1980s. At that time, the field of artificial neural network was booming again: it was a second birth for this interdisciplinary field, after a long and quiet period due to Minski and Papert's flaming book [138]. This boom already announced the future discovery of the back-propagation technique for the multi-layer perceptron [201, 161, 88, 160]. But in the early 1980s, all the attention was focused on Kohonen's work. He simplified von der Malsburg's ideas, implemented them with a clear and relatively fast algorithm, and introduced them in the field of artificial neural networks: the so-called (Kohonen's) self-organizing map (SOM or KSOM) was born.

Huge success of the SOMs in numerous applications of data analysis quickly followed and probably stems from the appealing elegance of the SOMs.

The task they perform is indeed very intuitive and easy to understand, although the mathematical translation of these ideas into a compactly written error function appears quite difficult or even impossible in the general case. More precisely, SOMs simultaneously perform the combination of two concurrent subtasks: vector quantization (see Appendix D) and topographic representation (i.e., dimensionality reduction). Thus, this "magic mix" is used not only for pure vector quantization, but also in other domains where self-organization plays a key part. For example, SOMs can also be used to some extent for nonlinear blind source separation [147, 85] as for nonlinear dimensionality reduction. This polyvalence explains the ubiquity of SOMs in numerous applications in several fields of data analysis like data visualization, time series prediction [123, 122], and so forth.

Embedding of data set

Within the framework of dimensionality reduction, an SOM can be interpreted intuitively as a kind of nonlinear but discrete PCA. In the latter method, a hyperplane is fitted at best inside the data cloud, and points are encoded as coordinates on that hyperplane. If the data cloud is curved, the hyperplane should be curved as well. One way to put this idea in practice consists of replacing the hyperplane with a discrete (and bounded) representation. For example, a grid or lattice defined by some points perfectly can play this role (see Fig. 5.1). If the segments connecting the grid points are elastic and articulated around the points, the fitting inside the data cloud becomes easy, at least intuitively. Roughly, it is like covering an object with an elastic fishing net. This intuitive idea underlies the way an SOM works. Unfortunately, things become difficult from an algorithmic point of view. How do we encode the fishing net? And how do we fit it inside the data cloud?

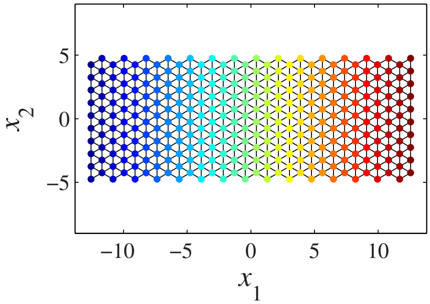


Fig. 5.1. Two-dimensional grid or lattice that can be used by a SOM.

Considering an SOM as a special case of a vector quantization method may help to answer the question. As explained in Appendix D, vector quantization aims at replacing a set of points with a smaller set of representative points called prototypes. Visually, “representative” means that the cloud of prototypes resembles the initial data cloud. In other words, the prototypes are fitted inside the data cloud. Unfortunately, classical vector quantization methods do not take into account a grid or lattice: prototypes move independently from each other.

An SOM circumvents this obstacle by moving several prototypes together, according to their location in the lattice. Assuming the prototypes are fitted iteratively in the data cloud, each time a prototype is moved, its neighbors in the lattice are moved too, in the same direction. Doing so allows us to keep the cohesion of the grid: neighboring prototypes in the grid will be located close to each other in the data cloud.

More formally, an SOM consists of

- A set $\mathcal{C}$ containing the prototypes for the vector quantization; these prototypes are D-dimensional points $\mathbf{c}(r)$, where D is the dimensionality of the data space.
- A function $d_{\mathbf{g}}(r,s)$ giving the distance between a pair of prototypes in the lattice; this distance function implicitly determines the neighborhood relationships between the prototypes.

In the context of dimensionality reduction, it is clear that the lattice plays the role of the embedding space. Hence, $d_{\mathbf{g}}(r,s)$ may not be any function. Very often $d_{\mathbf{g}}(r,s)$ is defined as $d(\mathbf{g}(r),\mathbf{g}(s))$, where $\mathbf{g}(r),\mathbf{g}(s)\in\mathcal{G}\subset\mathbb{R}^P$, and P is the dimensionality of the embedding space. In other words, prototypes have coordinates in the initial space as well as in the final space. Weirdly enough, coordinates $\mathbf{g}(r)$ in the embedding or latent space are known before running the SOM since they are fixed in advance, either explicitly or implicitly by defining $d_{\mathbf{g}}(r,s)$. On the other hand, the corresponding coordinates $\mathbf{c}(r)$ in the data space are unknown; it is the SOM's duty to determine them. Once the coordinates $\mathbf{c}(r)$ are computed, the embedding of a data point $\mathbf{y}(i)$ is given as the P-dimensional coordinates associated with the nearest prototype in the data space, i.e.,

$$\hat{\mathbf{x}}(i)=\mathbf{g}(r)\qquad\text{with}\qquad r=\arg\min_s d(\mathbf{y}(i),\mathbf{c}(s))\ , \tag{5.1}$$

where d is a distance function in the data space, usually the Euclidean distance.

The coordinates $\mathbf{c}(r)$ can be determined iteratively, by following more or less the scheme of a Robbins–Monro procedure. Briefly put, the SOM runs through the data set $\mathbf{Y}$ several times; each pass through the data set is called an *epochs*. During each epoch, the following operations are achieved for each datum $\mathbf{y}(i)$:

1. Determine the index r of the closest prototype of $\mathbf{y}(i)$, i.e.

$$r=\arg\min_s d(\mathbf{y}(i),\mathbf{c}(r))\ , \tag{5.2}$$

where d is typically the Euclidean distance.

2. Update the D-dimensional coordinates $\mathbf{c}(s)$ of all prototypes according to

$$\mathbf{c}(s) \leftarrow \mathbf{c}(s) + \alpha \nu_\lambda(r, s)(\mathbf{y}(i) - \mathbf{c}(s)) \ , \tag{5.3}$$

where the learning rate α, obeying $0 \leq \alpha \leq 1$, plays the same role as the step size in a Robbins–Monro procedure. Usually, α slowly decreases as epochs go by.

In the update rule (5.3), $\nu_\lambda(r, s)$ is called the neighborhood function and can be defined in several ways.

In the early publications about SOMs [191, 104], $\nu_\lambda(r, s)$ was defined to be the so-called 'Bubble' function:

$$\nu_\lambda(r, s) = \begin{cases} 0 & \text{if } d_\mathbf{g}(r, s) > \lambda \\ 1 & \text{if } d_\mathbf{g}(r, s) \leq \lambda \end{cases} \ , \tag{5.4}$$

where λ is the neighborhood (or Bubble) width. As the step size α does, the neighborhood width usually decreases slowly after each epoch.

In [155, 154], $\nu_\lambda(r, s)$ is defined as

$$\nu_\lambda(r, s) = \exp\left(-\frac{d_\mathbf{g}^2(r, s)}{2\lambda^2}\right) \ , \tag{5.5}$$

which looks like a Gaussian function (see Appendix B) where the neighborhood width λ replaces the standard deviation.

It is noteworthy that if $\nu_\lambda(r, s)$ is defined as

$$\nu_\lambda(r, s) = \begin{cases} 0 & \text{if } r \neq s \\ 1 & \text{if } r = s \end{cases} \ , \tag{5.6}$$

then the SOM does not take the lattice into account and becomes equivalent to a simple competitive learning procedure (see Appendix D).

In all above definitions, if $d_\mathbf{g}(r, s)$ is implicitly given by $d_\mathbf{g}(\mathbf{g}(r), \mathbf{g}(s))$, then $d_\mathbf{g}$ may be any distance function introduced in Section 4.2. In Eq. (5.5), the Euclidean distance (L_2) is used most of the time. In Eq. (5.4), on the other hand, L_2 as well as L_1 or L_∞ are often used.

Moreover, in most implementations, the points $\mathbf{g}(r)$ are regularly spaced on a plane. This forces the embedding space to be two-dimensional. Two neighborhood shapes are then possible: square (eight neighbors) or hexagonal (six neighbors) as in Fig. 5.1. In the first (resp., second) case, all neighbors are equidistant for L_∞ (resp., L_2). For higher-dimensional lattices, (hyper-)cubic neighborhoods are the most widely used. The global shape of the lattice is often a rectangle or a hexagon (or a parallelepiped in higher dimensions). Figure 5.2 shows a typical implementation of a SOM. The reference software package for SOMs is the SOM toolbox, available at `http://www.cis.hut.fi/projects/somtoolbox/`. This a complete MATLAB® toolbox, including

1. Define the lattice by assigning the low-dimensional coordinates $\mathbf{g}(r)$ of the prototypes in the embedding space.
2. Initialize the coordinates $\mathbf{c}(r)$ of the prototypes in the data space.
3. Give α and λ their scheduled values for epoch q.
4. For all points $\mathbf{y}(i)$ in the data set, compute r as in Eq. (5.2) and update all prototypes according to Eq. (5.3).
5. Return to step 3 until convergence is reached (i.e., updates of the prototypes become negligible).

Fig. 5.2. Batch version of Kohonen's self-organizing map.

not only the SOM algorithm but also other mapping functions (NLM, CCA) and visualization functions. A C++ implementation can also be downloaded from `http://www.ucl.ac.be/mlg/`. The main parameters of an SOM are the lattice shape (width, height, additional dimensions if useful), neighborhood shape (square or hexagon), the neighborhood function ν_λ, and the learning schedules (for learning rate α and neighborhood width λ). Space complexity is negligible and varies according to implementation choices. Time complexity is $\mathcal{O}(ND|\mathcal{C}|)$ per iteration.

Eventually, it is noteworthy that SOMs are motivated by biological and empirical arguments. Neither a generative model of data nor an objective function is defined, except in very particular cases [38]. More information about mathematical aspects of SOMs can be found in [62] and references therein.

Embedding of test set

Within the framework of vector quantization, any point $\mathbf{y}$ is encoded (not projected; see Appendix D) by giving the index r of the closest prototype $\mathbf{c}(r)$, which is computed in the same way as in Eq. (5.2). In this context, the embedding of $\mathbf{y}$ can be defined as $\hat{\mathbf{x}} = \mathbf{g}(r)$. Obviously, this gives only a very rough estimate of the low-dimensional coordinates, since at most $|\mathcal{C}|$ values (the number of prototypes) can be output. More precise interpolation procedures are described in [75].

Example

The two benchmark data sets introduced in Section 1.5 can easily be processed using an SOM. In order to quantize the 350 and 316 three-dimensional points contained in the data sets (see Fig. 1.4), the same 30-by-10 rectangular lattice as in Fig. 5.1 is defined. The neighborhood shape is hexagonal. With the neighborhood function defined as in Eq. (5.5), the SOM computes the embeddings shown in Fig. 5.3. By construction, the embedding computed by

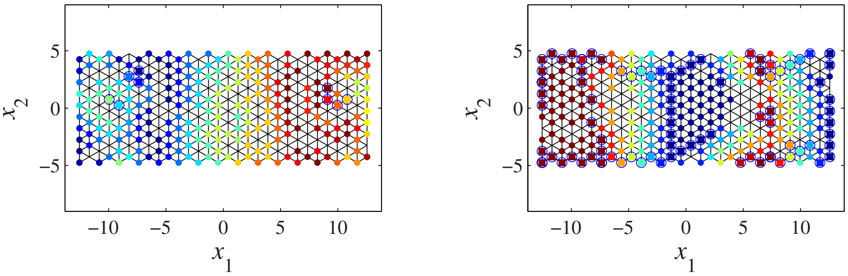


Fig. 5.3. Two-dimensional embeddings of the "Swiss roll" and "open box" data sets (Fig. 1.4), found by an SOM. The shape of the embedding is identical to the predefined lattice shown in Fig. 5.1. Only the color patches differ: the specific color and shape of each data point in Fig. 1.4 have been assigned to their closest prototypes.

an SOM *is* the predefined lattice, regardless of what the manifold to embed looks like. Indeed, the 30-by-10 rectangle of Fig. 5.1 is left unchanged.

Does it mean that the embedding conveys no information? Actually, as already mentioned, an SOM is often used to visualize labeled data. In the case of the open box, each point in the data set has been given a color (dark blue for the bottom, dark red for the top) and a shape (small circles for the faces and larger square for the edges and corners). Prototypes of the SOM can inherit these visual labels: each point gives its label to its closest prototype. As can be seen, the bottom face occupies the center of the rectangular lattice.

This interpretation is confirmed by looking at Fig. 5.4. The lattice "packs"

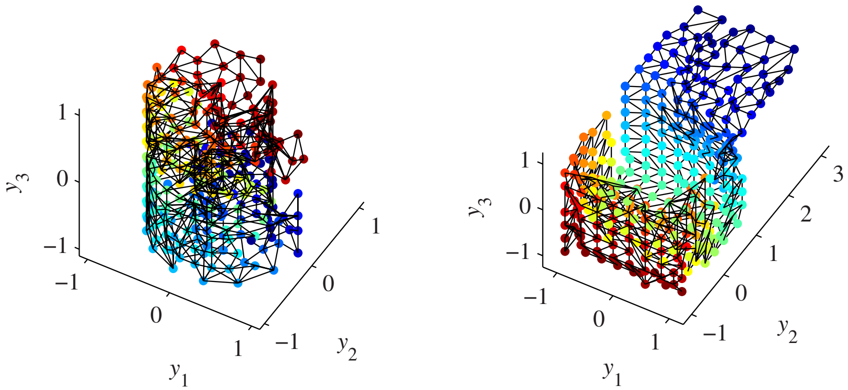


Fig. 5.4. Three-dimensional views showing how the lattice of an SOM curves in order to fit in the two data sets of Fig. 1.3. Colors indicate the left-right position of each point in the lattice, as in Fig. 5.1.

the box. Cracks, however, are visible on two lateral faces. They explain why two lateral faces are torn in Fig. 5.3 and cause the loss of some topological relationships. In addition, some points of the lattice have no color spot. This means that although the lattice includes fewer points than the data set, some points of the lattice never play the role of closest prototype.

Finally, it must be remarked that the axis ranges differ totally in Figs. 5.3 and 5.4. This intuitively demonstrates that only the topology has been preserved: the SOM has not taken distances into account.

Classification

An SOM is mainly a method of vector quantization. This means that vector quantization is mandatory in an SOM.

Regarding dimensionality reduction, an SOM models data in a nonlinear and discrete way, by representing it with a deformed lattice. The mapping is explicit (and defined only for the prototypes).

Most of the time, SOMs are implemented by offline algorithms, looking like a Robbins–Monro procedure. Online versions can easily be derived. A so-called ‘batch’ version of the SOM also exists [105]: instead of updating prototypes one by one, they are all moved simultaneously at the end of each epoch, as in a standard gradient descent.

Advantages and drawbacks

The wide success of SOMs can be explained by the following advantages. The method is very simple from an algorithmic point of view, and its underlying idea, once understood, is intuitively appealing. SOMs are quite robust and perform very well in many situations, such as the visualization of labeled data.

Nevertheless, SOMs have some well-known drawbacks, especially when they are used for dimensionality reduction. Most implementations handle one- or two-dimensional lattices only. Vector quantization is mandatory, meaning that a SOM does not really embed the data points: low-dimensional coordinates are computed for the prototypes only. Moreover, the shape of the embedding is identical to the lattice, which is, in turn, defined in advance, arbitrarily. This means that an SOM cannot capture the shape of the data cloud in the low-dimensional embedding.

From a computational point of view, it is very difficult to assess the convergence of an SOM, since no explicit objective function or error criterion is optimized. Actually, it has been proved that such a criterion cannot be defined, except in some very particular cases [57]. In addition, the parameter setting of an SOM appears as a tedious task, especially for the neighborhood width λ.

Variants

In the literature, many variants have been adapted to specific applications and tried with good success. Further details can be found in [105, 154] and references therein. To some extent, SOMs can also be related to principal curves [79, 80], or at least to their variants working with polygonal curve approximations [7].

It is noteworthy that several attempts have been made to give SOMs a data-driven lattice (see Section 5.3 although the methods described therein are quite different from the SOMs). In particular, Fritzke developed the growing cell structure [69] (GCS) and the growing grid [70] (GG), whereas Bauer and Villmann designed the growing SOM [11] (GSOM). Actually, these methods are incremental vector quantization techniques, i.e., the number of prototypes may increase automatically depending on the problem's complexity. Some variants of principal curves, using polygonal curve approximations, develop the same strategy in the 1D case [7].

The GCS is actually a TRN (see Section 5.1) able to build any two-dimensional lattice (neither the number nor the position of the lattice points is fixed in advance), but as most other TRNs, the GCS does not really perform a dimensionality reduction: an additional procedure [69] (*embedding method*) is needed in order to embed the lattice in a plane. Isotop (see Subsection 5.3.3), and particularly its third step, may be seen as an efficient way to embed the result of a TRN like the GCS in low-dimensional space.

The GSOM and GG, on the contrary, depart less from the original SOM and conserve a rectangular grid, making a two- or higher-dimensional representation straightforward, as for a traditional SOM. The difference with an SOM holds here in the possibility to add rows and columns in the lattice.

Thus, visibly, many variants of the SOMs stay very close to the original method. Specifically, very few models use anything other than the two-dimensional rectangular grid, with regularly placed points. However, it is possible to define lattices with any number of points, any shape, and even any distribution. A lattice could indeed be defined by randomly drawing points in any distribution. The predefined neighborhood relationships between these points can be made explicit by computing distances between them (either spatial or graph distance; see Sections 4.2 and 4.3). Thus, if any information about the manifold shape or distribution is known in advance, it can be exploited by fitting the lattice shape or distribution accordingly. Additionally, with today's computing power, the lattice can contain a huge number of points, in order to refine the discretization or to better approximate a specific distribution. As the only shortcoming, the use of such a randomly drawn lattice makes the visualization less straightforward.

The idea of giving a prior distribution to the lattice is also followed by the generative topographic mapping (GTM), which is introduced in Subsection 5.2.2. In the case of GTM, however, the choice of a prior distribution is intended to exploit statistical concepts and stochastic optimization methods rather than to broaden the choice of the grid shape.

5.2.2 Generative Topographic Mapping

The generative topographic mapping (GTM) has been put forward by Bishop, Svensén, and Williams [23, 24, 176] as a principled alternative to the SOM. Actually, GTM is a specific *density network* based on *generative modeling*, as indicated by its name. Although the term "generative model" has already been used, for example in Subsection 2.4.1 where the model of PCA is described, here it has a stronger meaning.

In generative modeling[1], all variables in the problem are assigned a probability distribution to which the Bayesian machinery is applied. For instance, density networks [129] are a form of Bayesian learning that try to model data in terms of latent variables [60]. Bayesian neural networks learn differently from other, more traditional, neural networks like an SOM. Actually, Bayesian learning defines a more general framework than traditional (frequentist) learning and encompasses it. Assuming that the data set $\mathbf{Y} = [\mathbf{y}(i)]_{1 \leq i \leq N}$ has to be modeled using parameters stored in vector $\mathbf{w}$, the likelihood function $L(\mathbf{w})$ is defined as the probability of the data set given the parameters

$$L(\mathbf{w}) = p(\mathbf{Y}|\mathbf{w}) \ . \tag{5.7}$$

Then traditional and Bayesian learning can be understood as follows [142]:

- **Traditional (frequentist) learning.** In an SOM, or in any other classical neural network like an MLP, no distribution over the model parameters $\mathbf{w}$ is assumed. The aim is then to determine the optimal value $\mathbf{w}_{\text{opt}}$, which is often found as a maximum likelihood estimator:

  $$\mathbf{w}_{\text{opt}} = \arg\max_{\mathbf{w}} \left(\ln L(\mathbf{w}) + R(\mathbf{w})\right) \ , \tag{5.8}$$

  where $R(\mathbf{w})$ is an optional regularization term, such as the usual quadratic regularizer:

  $$R(\mathbf{w}) = \alpha \frac{1}{2} \sum_k w_k^2 \ , \tag{5.9}$$

  where the hyperparameter α allows us to tune the relative importance of the regularization with respect to the primary objective function. Intuitively, regularization penalizes a neural network whose structure is a too complex, in order to avoid overfitting of the model. Test data $\mathbf{Y}_{\text{test}}$ can be processed by computing $p(\mathbf{Y}_{\text{test}}|w_{\text{opt}})$ when this is feasible or computationally tractable.

[1] This introdution to generative modeling is inspired from [33].

- **Bayesian learning.** For density networks, like GTM, a probability distribution over the model parameters $\mathbf{w}$ is obtained before considering any datum. Actually, such a distribution is based on a *prior* distribution $p(\mathbf{w})$ that expresses an initial belief about the value of $\mathbf{w}$. Afterwards, given data $\mathbf{Y}$, the prior distribution is updated to a *posterior* distribution using Bayes's rule:
$$p(\mathbf{w}|\mathbf{Y}) = \frac{p(\mathbf{Y}|\mathbf{w})p(\mathbf{w})}{p(\mathbf{Y})} \propto L(\mathbf{w})p(\mathbf{w}) \quad . \tag{5.10}$$
Test data $\mathbf{Y}_{\text{test}}$ can be processed by computing
$$p(\mathbf{Y}_{\text{test}}|\mathbf{Y}) = \int p(\mathbf{Y}_{\text{test}}|\mathbf{w})p(\mathbf{w}|\mathbf{Y})d\mathbf{w} \tag{5.11}$$
when this is feasible or computationally tractable.

According to Eq. (5.8), traditional learning may be viewed as a maximum a posteriori probability (MAP) estimate of Bayesian learning:

$$\begin{aligned}\mathbf{w}_{\text{opt}} &= \arg\max_{\mathbf{w}} p(\mathbf{w}|\mathbf{Y}) && (5.12)\\ &= \arg\max_{\mathbf{w}} \ln(L(\mathbf{w})p(\mathbf{w})) && (5.13)\\ &= \arg\max_{\mathbf{w}} (\ln L(\mathbf{w}) + \ln p(\mathbf{w})) \quad , && (5.14)\end{aligned}$$

with a prior $p(\mathbf{w}) = \exp R(\mathbf{w})$. For the quadratic regularizer (Eq. (5.9)), the prior would be proportional to a Gaussian density with variance $1/\alpha$.

Compared to frequentist learning, the Bayesian approach has the advantage of finding a distribution for the parameters in $\mathbf{w}$, instead of a single value. Unfortunately, this is earned at the expense of introducing the prior, whose selection is often criticized as being arbitrary.

Within the framework of Bayesian learning, density networks like GTM are intended to model a certain distribution $p(\mathbf{y})$ in the data space $\mathbb{R}^D$ by a small number P of latent variables. Given a data set (in matrix form) $\mathbf{Y} = [\mathbf{y}(i)]_{1\leq i \leq N}$ drawn independently from the distribution $p(\mathbf{y})$, the likelihood and log-likelihood become

$$L(\mathbf{w}) = p(\mathbf{Y}|\mathbf{w}) = \prod_{i=1}^{N} p(\mathbf{y}(i)|\mathbf{w}) \quad , \tag{5.15}$$

$$l(\mathbf{w}) = \ln p(\mathbf{Y}|\mathbf{w}) = \sum_{i=1}^{N} \ln p(\mathbf{y}(i)|\mathbf{w}) \quad , \tag{5.16}$$

respectively. Starting from these equations, density networks are completely determined by choosing the following parameters or elements:

- The dimension P of the latent space.
- The prior distribution in the latent P-dimensional space: $p(\mathbf{x})$, $\mathbf{x}$ being a random vector of $\mathbb{R}^P$.

- A smooth mapping $\mathbf{m}$ from the latent space onto a P-manifold $\mathcal{Y}$ in the D-dimensional data space, with parameter vector $\mathbf{w}$ (for example, if $\mathbf{m}$ is an MLP, $\mathbf{w}$ would be the weights and biases):

$$\mathbf{m} : \mathbb{R}^P \to \mathcal{Y} \subset \mathbb{R}^D, \mathbf{x} \mapsto \mathbf{y} = \mathbf{m}(\mathbf{x}, \mathbf{w}) \ . \tag{5.17}$$

  The possibility to reduce the dimensionality clearly holds in this jump from P to D dimensions.
- An error function $G_i(\mathbf{x}, \mathbf{w}) = \ln p(\mathbf{y}(i)|\mathbf{x}, \mathbf{w}) = \ln p(\mathbf{y}(i)|\mathbf{y})$. Applying Bayes's rule, the posterior can then be computed in the latent space from the prior and the error function:

$$p(\mathbf{x}|\mathbf{y}(i), \mathbf{w}) = \frac{p(\mathbf{y}(i)|\mathbf{x}, \mathbf{w})p(\mathbf{x})}{p(\mathbf{y}(i)|\mathbf{w})} = \frac{\exp(G_i(\mathbf{x}, \mathbf{w}))p(\mathbf{x})}{p(\mathbf{y}(i)|\mathbf{w})} \ , \tag{5.18}$$

  with the normalization constant

$$p(\mathbf{y}(i)|\mathbf{w}) = \int p(\mathbf{y}(i)|\mathbf{x}, \mathbf{w})p(\mathbf{x})d\mathbf{x} \ . \tag{5.19}$$

  Using Bayes's rule once again, as in Eq. (5.10), gives the posterior in the parameter space:

$$p(\mathbf{w}|\mathbf{Y}) = \frac{p(\mathbf{Y}|\mathbf{w})p(\mathbf{w})}{p(\mathbf{Y})} = \frac{L(\mathbf{w})p(\mathbf{w})}{p(\mathbf{Y})} \ , \tag{5.20}$$

- An optimization algorithm in order to find the parameter $\mathbf{w}$ that maximizes the posterior in the parameter space $p(\mathbf{w}|\mathbf{Y})$. In practice, this is achieved by maximizing the log-likelihood, for example by gradient descent on Eq. (5.19), when this is computationally feasible.

Embedding of data set

In GTM, the various elements of a density network are set as follows:

- The error functions $G_i(\mathbf{x}, \mathbf{w})$: the probabilities $p(\mathbf{y}(i)|\mathbf{x}, \mathbf{w})$ are spherical Gaussian kernels $N(\mathbf{m}(\mathbf{x}, \mathbf{W}), \beta^{-1}\mathbf{I})$, with parameters $\mathbf{w} = \{\mathbf{W}, \beta\}$. The kernels are centered on $\mathbf{m}(\mathbf{x}, \mathbf{W})$ and have a variance equal to β^{-1}:

$$p(\mathbf{y}(i)|\mathbf{x}, \mathbf{W}, \beta) = \left(\frac{\beta}{2\pi}\right)^{\frac{D}{2}} \exp\left(-\frac{\beta}{2}\|\mathbf{y}(i) - \mathbf{m}(\mathbf{x}, \mathbf{W})\|^2\right) \ , \tag{5.21}$$

  which behaves as an isotropic Gaussian noise model for $\mathbf{y}(\mathbf{x}, \mathbf{W})$ that extends the manifold $\mathcal{Y}$ to $\mathbb{R}^D$: a given data vector $\mathbf{y}(i)$ could have been generated by any point $\mathbf{x}$ with probability $p(\mathbf{y}(i)|\mathbf{x}, \mathbf{W}, \beta)$. It can be also remarked that the error function $G_i(\mathbf{x}, \mathbf{W}, \beta)$ trivially depends on the squared distance between the observed data point $\mathbf{y}(i)$ and the generating point $\mathbf{x}$.

- The prior distribution in latent space:

$$p(\mathbf{x}) = \frac{1}{C}\sum_{r=1}^{C}\delta(\mathbf{x}-\mathbf{g}(r)) = \begin{cases} 0 & \text{if } \mathbf{x} \neq \mathbf{g}(r) \\ 1/C & \text{if } \mathbf{x} = \mathbf{g}(r) \end{cases} \tag{5.22}$$

  where the C points $\mathbf{g}(r)$ stand on a regular grid in latent space, in the same way as the prototypes of an SOM. This discrete choice of the prior distribution directly simplifies the integral in Eq. (5.19) into a sum. Otherwise, for an arbitrary $p(\mathbf{x})$, the integral must be explicitly discretized (Monte Carlo approximation). Then, finally, Eq. (5.19) becomes in the case of GTM:

$$p(\mathbf{y}(i)|\mathbf{W},\beta) = \frac{1}{C}\sum_{r=1}^{C} p(\mathbf{y}(i)|\mathbf{g}(r),\mathbf{W},\beta) \ , \tag{5.23}$$

  which is a constrained mixture of Gaussian kernels (because the kernels lie on the P-dimensional manifold $\mathcal{Y}$ in the data space). Similarly, the log-likelihood becomes

$$l(\mathbf{W},\beta) = \sum_{i=1}^{N} \ln\left(\frac{1}{C}\sum_{r=1}^{C} p(\mathbf{y}(i)|\mathbf{W},\beta)\right) \ . \tag{5.24}$$

- The mapping from latent space to data space is a generalized linear model $\mathbf{m}(\mathbf{x},\mathbf{W}) = \mathbf{W}\boldsymbol{\phi}(\mathbf{x})$, where $\mathbf{W}$ is D-by-B matrix and $\boldsymbol{\phi}$ a B-by-1 vector consisting of B (nonlinear) basis functions. Typically, these B basis functions are Gaussian kernels with explicitly set parameters: their centers are drawn from the grid in the latent space, and their common variance is proportional to the mean distances between the centers. In other words, the mapping used by GTM roughly corresponds to an RBF network with constrained centers: by comparison with the MLP or usual density networks, the RBFN remains an universal approximator but yields considerable simplifications in the subsequent computations (see ahead). The exact positioning of the centers and the tuning of the width σ are not discussed here; for more details, see [176]. Nevertheless, in order to get a smooth manifold and avoid overfitting, it is noteworthy that (i) the number of kernels in the constrained RBF must be lower than the number of grid points and (ii) the width σ must be larger than the mean distance between neighboring centers. As in other RBF networks, additional linear terms and biases may complement the basis functions in order to easily take into account linear trends in the mapping.
- The optimization algorithm is the expectation-maximization (EM) procedure [50, 21]. This choice is typical when maximizing the likelihood and working with mixtures of Gaussian kernels. By design, the prior in latent space is a mixture of Gaussian kernels (see Eq. (5.23)) that fortunately makes EM applicable. In other density networks, a more complex choice

of the prior does not allow simplifying the integral of Eq. (5.19) into a sum. In the case of GTM, the objective function is the log-likelihood function. Without going into technical details [176], the log-likelihood function

$$l(\mathbf{W}, \beta) = \sum_{i=1}^{N} \ln p(\mathbf{y}(i)|\mathbf{W}, \beta) \tag{5.25}$$

leads to the following two steps:

E step.

Computation of the responsibilities $\rho_{i,r}(\mathbf{W}, \beta)$ (see Eq. (5.28) ahead).

M step.

The partial derivatives of $L(\mathbf{W}, \beta)$ with respect to parameters $\mathbf{W}$ and β give:

- A matrix equation for $\mathbf{W}$:

$$\mathbf{W}^{\text{new}} \mathbf{\Phi} \mathbf{G}_{\text{old}} \mathbf{\Phi}^T = \mathbf{Y} \mathbf{P}_{\text{old}} \mathbf{\Phi}^T \quad , \tag{5.26}$$

  solvable for $\mathbf{W}_{\text{new}}$ with standard matrix inversion techniques. This simple update rule results from the adoption of an RBFN-like approximator instead of, for example, an MLP, which would have required a gradient ascent as optimization procedure.

- A re-estimation formula for β:

$$\frac{1}{\beta} \leftarrow \frac{1}{ND} \sum_{r=1}^{C} \sum_{i=1}^{N} \rho_{i,r}(\mathbf{W}^{\text{new}}, \beta) \, \|\mathbf{y}(i) - \mathbf{m}(\mathbf{g}(r), \mathbf{W}^{\text{new}})\|^2 \quad , \tag{5.27}$$

where

- $\mathbf{\Phi} = [\boldsymbol{\phi}(\mathbf{g}(r))]_{1 \leq r \leq C}$ is a B-by-C constant matrix,
- $\mathbf{Y} = [\mathbf{y}(i)]_{1 \leq i \leq N}$ is the data set (constant D-by-N matrix),
- $\mathbf{P} = [\rho_{i,r}(\mathbf{W}, \beta)]$ is a varying N-by-C matrix of posterior probabilities or *responsibilities*:

$$\begin{aligned} \rho_{i,r}(\mathbf{W}, \beta) &= p(\mathbf{g}(r)|\mathbf{y}(i), \mathbf{W}, \beta) \\ &= \frac{p(\mathbf{y}(i)|\mathbf{g}(r), \mathbf{W}, \beta) p(\mathbf{g}(r))}{\sum_{s=1}^{C} p(\mathbf{y}(i)|\mathbf{g}(s), \mathbf{W}, \beta) p(\mathbf{g}(s))} \\ &= \frac{p(\mathbf{y}(i)|\mathbf{g}(r), \mathbf{W}, \beta)}{\sum_{s=1}^{C} p(\mathbf{y}(i)|\mathbf{g}(s), \mathbf{W}, \beta)} \quad , \end{aligned} \tag{5.28}$$

- $\mathbf{G}$ is a diagonal C-by-C matrix with entries

$$g_{r,r}(\mathbf{W}, \beta) = \sum_{i=1}^{N} \rho_{i,r}(\mathbf{W}, \beta) \quad . \tag{5.29}$$

Because the EM algorithm increases the log-likelihood monotonically [50], the convergence of GTM is guaranteed. According to [24], convergence is generally attained after a few tens of iterations. As initial weights, one can take the first P principal components of the data set (see [176] for more details). After convergence, a small value of the variance $1/\beta_{\text{opt}}$ generally indicates a good approximation of the data.

Once the optimal values for the parameters $\mathbf{W}$ and β are known, the embedding $\mathbf{x}(i)$ in the latent space of the data points $\mathbf{y}(i)$ can easily be computed. By construction, the optimal parameters allow defining $p(\mathbf{y}(i)|\mathbf{g}(r))$, i.e., the probability distribution in the data space, conditioned by the latent variables. Knowing the prior distribution $p(\mathbf{x})$ over the latent variables (see Eq. (5.22)) and using Bayes's theorem, the posterior probability distribution can be written as

$$p(\mathbf{g}(r)|\mathbf{y}(i)) = \frac{p(\mathbf{y}(i)|\mathbf{g}(r), \mathbf{W}_{\text{opt}}, \beta_{\text{opt}})p(\mathbf{g}(r))}{\sum_{s=1}^{C} p(\mathbf{y}(i)|\mathbf{g}(s), \mathbf{W}, \beta)p(\mathbf{g}(s))} \tag{5.30}$$

$$= \frac{p(\mathbf{y}(i)|\mathbf{g}(r), \mathbf{W}_{\text{opt}}, \beta_{\text{opt}})}{\sum_{s=1}^{C} p(\mathbf{y}(i)|\mathbf{g}(s), \mathbf{W}, \beta)} \tag{5.31}$$

$$= \rho_{i,r}(\mathbf{W}_{\text{opt}}, \beta_{\text{opt}}) \tag{5.32}$$

and closely resembles the computation of the responsibilities in Eq. (5.28), where again the $p(\mathbf{g}(r))$ disappear. Until here, only the probabilities of the different points of the grid in the latent space are known. In order to compute an estimate $\mathbf{x}(i)$ in vector form, two possibilities exist:

- the mode of the posterior distribution in the latent space:

$$\hat{\mathbf{x}}(i) = \arg\max_{\mathbf{g}(r)} p(\mathbf{g}(r)|\mathbf{y}(i)) \ , \tag{5.33}$$

- the mean of the posterior distribution in the latent space:

$$\hat{\mathbf{x}}(i) = \sum_{r=1}^{C} \mathbf{g}(r)p(\mathbf{g}(r)|\mathbf{y}(i)) \ . \tag{5.34}$$

The second possibility is clearly the best except when the posterior distribution is multimodal. Putting together all the above-mentioned ideas leads to the procedure presented in Fig. 5.5. A GTM MATLAB® package is available at `http://www.ncrg.aston.ac.uk/GTM/`. The parameter list of GTM is quite long. As with an SOM, the shape of the grid or lattice may be changed (rectangular or hexagonal). The basis function of the RBF-like layer can be tuned, too. Other parameters are related to the EM procedure (initialization, number of iterations, etc.).

```
 1. Generate the grid of latent points $\{\mathbf{g}(r)\}_{1 \leq r \leq C}$.
 2. Generate the grid of the centers used in the basis functions.
 3. Select the width $\sigma$ of the basis functions.
 4. Evaluate the basis functions at the latent points $\mathbf{g}(r)$, and store the results
    in $\mathbf{\Phi}$.
 5. Initialize $\mathbf{W}$ either randomly or using PCA.
 6. Initialize the inverse of the noise variance $\beta$.
 7. Compute the matrix of squared distances $\mathbf{D} = [\|\mathbf{y}(i) - \mathbf{W}\boldsymbol{\phi}(\mathbf{g}(r))\|^2]$.
 8. E step:
    • Compute $\mathbf{P}$ from Eq. (5.28) using $\mathbf{D}$ and $\beta$.
    • Compute $\mathbf{G}$ from Eq. (5.29) using $\mathbf{R}$.
 9. M step:
    • Update $\mathbf{W}$ with Eq. (5.26): $\mathbf{W} = \mathbf{Y}\mathbf{P}\mathbf{\Phi}^T(\mathbf{\Phi}\mathbf{G}\mathbf{\Phi}^T)^{-1}$.
    • Compute $\mathbf{D}$ as above, with the updated value of $\mathbf{W}$.
    • Update $\beta$ according Eq. (5.27), using $\mathbf{R}$ and $\mathbf{D}$.
10. Return to step 8 if convergence is not yet reached.
11. Compute the embedding of the data points using $\mathbf{P}$ and then Eq. (5.33)
    or (5.34).
```

Fig. 5.5. Algorithm implementing the generative topographic mapping.

Embedding of test set

By construction, GTM aims at providing an easy way to generalize the dimensionality reduction to new points. The embedding of a test point $\mathbf{y}$ can be computed in the same way as for the data points $\mathbf{y}(i)$, using Eq. (5.32) and then Eq. (5.33) or (5.34).

Example

Figure 5.6 illustrates how GTM embeds the "Swiss roll" and "open box" manifolds introduced in Section 1.5. The points of the data sets (see Fig. 1.4) are embedded in the latent space (a square 10-by-10 grid) using Eq. (5.34). The mapping $\mathbf{m}$ works with a grid of 4-by-4 basis functions.

As can be seen, GTM fails to embed the Swiss roll correctly: all turns of the spiral are superposed. On the other hand, the square latent space perfectly suits the cubic shape of the open box: the bottom face is in the middle of the square, surrounded by the four lateral faces. The upper corners of the box correspond to the corners of the square latent space. By comparison with an SOM (Fig. 5.3), GTM yields a much more regular embedding. The only visible shortcoming is that the lateral faces are a bit shrunk near the borders of the latent space.

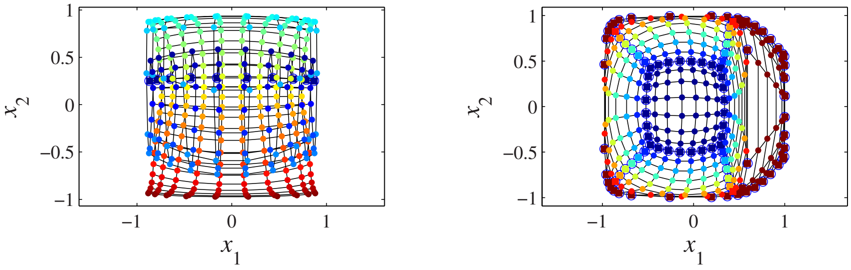


Fig. 5.6. Two-dimensional embeddings of the “Swiss roll” and “open box” data sets (Fig. 1.4), found by the GTM.

Classification

The essential difference between GTM and almost all other methods described in this book is that GTM relies on the principle of Bayesian learning. This probabilistic approach leads to a different optimization technique: an EM algorithm is used instead of a (stochastic) gradient descent or a spectral decomposition. As described above, GTM is a batch method, but a version that works with a stochastic EM procedure also exists.

Because GTM determines the parameters of a generative model of data, the dimensionality reduction easily generalizes to new points. Therefore, it can be said that GTM defines an implicit mapping although the latent space is discrete.

If the implementation does not impose a two-dimensional grid, an external procedure is needed to estimate the intrinsic dimensionality of data, in order to determine the right dimension for the latent space.

If several embeddings with various dimensionalities are desired, GTM must be run again.

Advantages and drawbacks

By comparison with an SOM, GTM provides a generative model for data. Moreover, the probabilistic approach that has come along has several advantages.

First, in addition to finding the latent coordinates $\hat{\mathbf{x}}$ of a point $\mathbf{y}$, GTM can also approximate $\hat{p}(\mathbf{x}|\mathbf{y})$, i.e., the probability of the embedding to be located at coordinates $\mathbf{x}$ in the latent space. This allows us to detect problems in the dimensionality reduction when, for instance, the probability distribution is not unimodal.

Second, from an algorithmic point of view, GTM optimizes a well-defined objective function, namely the log-likelihood, whereas no such function exists for the SOM [57]. While most methods described in this book work with (stochastic) gradient descents or related techniques, GTM optimizes the log-likelihood using an EM algorithm. Compared with these classical optimization techniques, the EM is guaranteed to maximize the likelihood monotically and converges to a maximum after a few tens of iterations.

As an SOM, GTM is limited to low-dimensional latent spaces: typically, only one- or two-dimensional grids are used. Actually, the mapping $\mathbf{m}$ does not manage very well with high-dimensional latent spaces for the following two reasons. First, the number of grid points must grow exponentially, like in an SOM. But in the case of GTM, the number of kernels defining $\mathbf{m}$ must grow as well. Second, as remarked in Subsection 1.2.2, the Gaussian kernels used in $\mathbf{m}$ behave surprisingly in high-dimensional spaces. Therefore, the dimensionality of the latent space must remain low, as for an SOM.

By the way, regarding the mapping $\mathbf{m}$, it is important to remark that although the universal approximation property holds for RBF networks in theory, the RBF-like mapping of GTM is utterly simple and constrained:

- The number of kernels is limited by the number of latent points in the grid.
- The width σ is isotropic in the latent space and shared by all kernels.
- The kernel centers and variances are the same for all D dimensions of the data space.
- The kernel centers and the variance are set in advance and are constant; only the weights are adjusted.

By comparison, RBF networks used for function approximation [93, 18] are far more flexible:

- The number of kernels is only limited by the number of available data points.
- The variance is often individualized for each kernel and is sometimes not isotropic (complete covariance matrix).
- The kernel centers and variances are optimized separately for each output dimension.
- The kernel centers and variances are optimized on the same footing as the weights.

Unfortunately, the integration of such a generic RBF network appears difficult in GTM, since many simplifications become impossible.

A table summarizing the differences between GTM and an SOM is given in [33].

Variants

Some extensions to GTM are described in [176]. For example, the probabilistic framework of GTM can be adapted to data sets with missing entries. Another variant uses a different noise model: the Gaussian kernels include a full covariance matrix instead of being isotropic. The mapping $\mathbf{m}$ can also be easily modified in order to cope with high-dimensional latent spaces. Mixtures of several GTMs are also considered.

5.3 Data-driven lattice

In contrast with methods using a predefined lattice studied in the previous section, the methods described here make no assumption about the shape and topology of the embedding. Instead, they use information contained in data in order to establish the topology of the data set and compute the shape of the embedding accordingly. Thus, the embedding is not constrained in any way and can adapt itself in order to capture the manifold shape.

In all methods detailed in the coming sections, the data-driven lattice is formalized by a graph whose vertices are the data points and whose edges represent neighborhood relationships.

5.3.1 Locally linear embedding

By comparison with an SOM and GTM, locally linear embedding [158, 166] (LLE) considers topology preservation from a slightly different point of view. Usual methods like an SOM and GTM try to preserve topology by keeping neighboring points close to each other (neighbors in the lattice are maintained close in the data space). In other words, for these methods, the qualitative notion of topology is concretely translated into relative proximities: points are close to or far from each other. LLE proposes another approach based on conformal mappings. A *conformal mapping* (or conformal map or biholomorphic map) is a transformation that preserves local angles. To some extent, the preservation of local angles and that of local distances are related and may be interpreted as two different ways to preserve local scalar products.

Embedding of data set

As LLE builds a conformal mapping, the first task of LLE consists of determining which angles to take into account. For this purpose, LLE selects a couple of neighbors for each data point $\mathbf{y}(i)$ in the data set $\mathbf{Y} = [\ldots, \mathbf{y}(i), \ldots, \mathbf{y}(j), \ldots]_{1 \leq i,j \leq N}$. Like other methods already studied, LLE can perform this task with several techniques (see Appendix E). The most often used ones associate with each point $\mathbf{y}(i)$ either the K closest other points or all points lying inside an ϵ-ball centered on $\mathbf{y}(i)$.

If the data set is sufficiently large and not too noisy, i.e., if the underlying manifold is well sampled, then one can assume that a value for K (or ϵ) exists such that the manifold is approximately linear, on the local scale of the K-ary neighborhoods (or ϵ-balls). The idea of LLE is then to replace each point $\mathbf{y}(i)$ with a linear combination of its neighbors. Hence, the local geometry of the manifold can be characterized by linear coefficients that reconstruct each data point from its neighbors. The total reconstruction error can be measured by the simple quadratic cost function

$$\mathcal{E}(\mathbf{W}) = \sum_{i=1}^{N} \left\| \mathbf{y}(i) - \sum_{j \in \mathcal{N}(i)} w_{i,j} \mathbf{y}(j) \right\|^2 , \tag{5.35}$$

where $\mathcal{N}(i)$ is the set containing all neighbors of point $\mathbf{y}(i)$ and $w_{i,j}$, the entries of the N-by-N matrix $\mathbf{W}$, weight the neighbors in the reconstruction of $\mathbf{y}(i)$. Briefly put, $\mathcal{E}(\mathbf{W})$ sums all the squared distances between a point and its locally linear reconstruction. In order to compute the coefficients $w_{i,j}$, the cost function is minimized under two constraints:

- Points are reconstructed solely by their neighbors, i.e., the coefficients $w_{i,j}$ for points outside the neighborhood of $\mathbf{y}(i)$ are equal to zero: $w_{i,j} = 0 \; \forall j \notin \mathcal{N}(i)$;
- The rows of the coefficient matrix sum to one: $\sum_{j=1}^{N} w_{i,j} = 1$.

The constrained weights that minimize the reconstruction error obey an important property: for any particular data point $\mathbf{y}(i)$, they are invariant to rotations, rescalings, and translations of that data point and its neighbors. The invariance to rotations and rescalings follows immediately from the particular form of Eq. (5.35); the invariance to translations is enforced by the second constraint on the rows of matrix $\mathbf{W}$. A consequence of this symmetry is that the reconstruction weights characterize intrinsic geometric properties of each neighborhood, as opposed to properties that depend on a particular frame of reference. The key idea of LLE then consists of assuming that these geometric properties would also be valid for a low-dimensional representation of the data.

More precisely, as stated in [158], LLE assumes that the data lie on or near a smooth, nonlinear manifold of low intrinsic dimensionality. And then, to a good approximation, there exists a linear mapping, consisting of a translation, rotation, and rescaling, that maps the high-dimensional coordinates of each neighborhood to global intrinsic coordinates on the manifold. By design, the reconstruction weights $w_{i,j}$ reflect intrinsic geometric properties of the data that are invariant to exactly these transformations. Therefore, it is expected that their characterization of local geometry in the original data space be equally valid for local patches on the manifold. In particular, the same weights $w_{i,j}$ that reconstruct the data point $\mathbf{y}(i)$ in the D-dimensional data space should also reconstruct its manifold coordinates in a P-dimensional embedding space.

LLE constructs a neighborhood-preserving embedding based on the above assumption. In the final step of LLE indeed, each high-dimensional data point is mapped to a low-dimensional vector representing global intrinsic coordinates on the manifold. This is done by choosing P-dimensional coordinates to minimize the embedding cost function:

$$\Phi(\hat{\mathbf{X}}) = \sum_{i=1}^{N} \left\| \hat{\mathbf{x}}(i) - \sum_{j \in \mathcal{N}(i)} w_{i,j} \hat{\mathbf{x}}(j) \right\|^2 \quad . \tag{5.36}$$

This cost function, very similar to the previous one in Eq. (5.35), sums the reconstruction errors caused by locally linear reconstruction. In this case, however, the errors are computed in the embedding space and the coefficients $w_{i,j}$ are fixed. The minimization of $\Phi(\hat{\mathbf{X}})$ gives the low-dimensional coordinates $\hat{\mathbf{X}} = [\ldots, \hat{\mathbf{x}}(i), \ldots, \hat{\mathbf{x}}(j), \ldots]_{1 \leq i,j \leq N}$ that best reconstruct $\mathbf{y}(i)$ given $\mathbf{W}$.

In practice, the minimization of the two cost functions $\mathcal{E}(W)$ and $\Phi(\hat{\mathbf{X}})$ is achieved in two successive steps.

First, the constrained coefficients $w_{i,j}$ can be computed in closed form, for each data point separately. Considering a particular data point $\mathbf{y}(i)$ with K nearest neighbors, its contribution to $\mathcal{E}(\mathbf{W})$ is

$$\mathcal{E}_i(\mathbf{W}) = \left\| \mathbf{y}(i) - \sum_{j \in \mathcal{N}(i)} w_{i,j} \mathbf{y}(j) \right\|^2 \quad , \tag{5.37}$$

which can be reformulated as

$$\begin{aligned} \mathcal{E}_i(\boldsymbol{\omega}(i)) &= \|\mathbf{y}(i) - \sum_{r=1}^{K} \omega_r(i) \boldsymbol{\nu}(r)\|^2 & (5.38)\\ &= \| \sum_{j=1}^{K} \omega_r(i)(\mathbf{y}(i) - \boldsymbol{\nu}(r))\|^2 & (5.39)\\ &= \sum_{r,s=1}^{K} \omega_r(i)\omega_s(i) g_{r,s}(i) \quad , & (5.40) \end{aligned}$$

where $\boldsymbol{\omega}(i)$ is a vector that contains the nonzero entries of the ith (sparse) row of $\mathbf{W}$ and $\boldsymbol{\nu}(r)$ the rth neighbor of $\mathbf{y}(i)$, corresponding to $\mathbf{y}(j)$ in the notation of Eq. (5.37). The second equality holds thanks to the (reformulated) constraint $\sum_{r=1}^{K} \omega_r(i) = 1$, and the third one uses the K-by-K local Gram matrix $\mathbf{G}(i)$ whose entries are defined as

$$g_{r,s}(i) = (\mathbf{y}_i - \boldsymbol{\nu}(r))^T (\mathbf{y}_i - \boldsymbol{\nu}(s)) \quad . \tag{5.41}$$

The matrices $\mathbf{G}(i)$ can be interpreted as a kind of local covariance matrices around $\mathbf{y}(i)$. The reconstruction error can be minimized in closed form, using a Lagrange multiplier to enforce the constraint $\sum_{r=1}^{K} \omega_r(i) = 1$. In terms of the inverse of $\mathbf{G}(i)$, the optimal weights are given by

$$\omega_r(i) = \frac{\sum_{s=1}^{K} (\mathbf{G}^{-1}(i))_{r,s}}{\sum_{r,s=1}^{K} (\mathbf{G}^{-1}(i))_{r,s}} \quad . \tag{5.42}$$

This solution requires an explicit inversion of the local covariance matrix. In practice, a more efficient way to minimize the error is simply to solve the linear system of equations $\sum_{r=1}^{K} g_{r,s}\omega_r(i)$ and then to rescale the coefficients so that they sum to one, yielding the same result. By construction, the matrix $\mathbf{G}(i)$ is symmetric and positive semidefinite. Unfortunately, it can be singular or nearly singular, for example, when there are more neighbors than the dimensions in the data space ($K > D$). In this case, $\mathbf{G}$ can be conditioned, before solving the system, by adding a small multiple of the identity matrix:

$$\mathbf{G} \leftarrow \mathbf{G} + \frac{\varDelta^2 \ \mathrm{tr}(\mathbf{G})}{K}\mathbf{I} \ , \tag{5.43}$$

where $\varDelta$ is small compared to the trace of C. This amounts to penalizing large weights that exploit correlations beyond some level of precision in the data sampling process. Actually, $\varDelta$ is somehow a “hidden” parameter of LLE.

The minimization of the second cost function $\varPhi(\hat{\mathbf{X}})$ can be done at once by solving an eigenproblem. For this purpose, $\varPhi(\hat{\mathbf{X}})$ is developed as follows:

$$\varPhi(\hat{\mathbf{X}}) = \sum_{i=1}^{N} \left\| \hat{\mathbf{x}}(i) - \sum_{j \in \mathcal{N}(i)} w_{i,j}\hat{\mathbf{x}}(j) \right\|^2 \tag{5.44}$$

$$= \sum_{i=1}^{N} \left\| \sum_{j \in \mathcal{N}(i)} w_{i,j}(\hat{\mathbf{x}}(i) - \hat{\mathbf{x}}(j)) \right\|^2 \tag{5.45}$$

$$= \sum_{i,j=1}^{N} m_{i,j}(\hat{\mathbf{x}}(i)^T\hat{\mathbf{x}}(j)) \ , \tag{5.46}$$

where $m_{i,j}$ are the entries of an N-by-N matrix $\mathbf{M}$, defined as

$$\mathbf{M} = (\mathbf{I} - \mathbf{W})^T(\mathbf{I} - \mathbf{W}) \ , \tag{5.47}$$

which is sparse, symmetric, and positive semidefinite. The optimization is then performed subject to constraints that make the problem well posed. It is clear that the coordinates $\hat{\mathbf{x}}(i)$ can be translated by a constant displacement without affecting the cost. This degree of freedom disappears if the coordinates are required to be centered on the origin ($\sum_{i=1}^{N} \hat{\mathbf{x}}(i) = 0$). Moreover, in order to avoid degenerate solutions, the latent coordinates are constrained to have unit covariance ($\hat{\mathbf{C}}_{\hat{\mathbf{X}}\hat{\mathbf{X}}} = \frac{1}{N}\hat{\mathbf{X}}\hat{\mathbf{X}}^T = \mathbf{I}$). Such a constraint simply exploits the invariance of the cost function to rotations and homogenous rescalings. The optimal embedding, up to a global rotation of the embedding space, is found by computing the bottom $P + 1$ eigenvectors of the matrix $\mathbf{M}$. The last eigenvector of $\mathbf{M}$, which is actually discarded by LLE, is a scaled unit vector with all components equal; it represents a free translation mode and is associated with a zero eigenvalue. Discarding this eigenvector enforces the constraint that the embeddings have a zero mean, since the components of other eigenvectors must sum to zero by virtue of orthogonality with the last one. The remaining P eigenvectors give the estimated P-dimensional coordinates of the points $\hat{\mathbf{x}}(i)$ in the latent space. Figure 5.7 summarizes all previous ideas in a short procedure: A MATLAB® function implementing LLE is available

1. For each datum $\mathbf{y}(i)$, compute
   - the K nearest neighbors of $\mathbf{y}(i)$,
   - the regularized matrix $\mathbf{G}(i)$ according to Eq. (5.41) and (5.43),
   - the weights $\boldsymbol{\omega}(i)$ (Eq. (5.42)).
2. Knowing the vectors $\boldsymbol{\omega}(i)$, build the sparse matrices $\mathbf{W}$ and $\mathbf{M}$ (Eq. (5.47)).
3. Compute the EVD of $\mathbf{M}$; the estimated coordinates are given by the eigenvectors associated with the second-to-$(1+P)$th smallest eigenvalues.

Fig. 5.7. Algorithm for locally linear embedding.

at `http://www.cs.toronto.edu/~roweis/lle/` along with related publications. The main parameters of this MATLAB® function are the embedding dimensionality P and the number of neighbors K. An additional parameter, called “tolerance” and denoted Δ above, is used for regularizing matrix $\mathbf{G}(i)$ computed for each datum. This parameter can play an important part in the result of LLE. Space and time complexities largely depend on the way LLE is implemented. As LLE involves mainly N-by-N sparse matrices and knowing that only a few eigenvectors need to be computed, LLE can take advantage of optimized software libraries.

Embedding of test set

In the original version of LLE [158], no interpolation procedure is provided. However, two different ideas are proposed in [166].

The first idea consists of a local linear interpolation, following the same intuition that motivated LLE. The point $\mathbf{y}$ to embed is compared with the known data points in order to determine its K nearest neighbors. Next, reconstruction weights are computed. Finally, the embedding $\hat{\mathbf{x}}$ is built using the reconstruction weights.

The second idea relies on a parametric model, for instance a neural network like an MLP or an RBFN, which learns the mapping between the data set $\mathbf{Y}$ and its embedding $\hat{\mathbf{X}}$ computed by LLE. Similar ideas have been developed for Sammon’s NLM. This transforms the discrete and explicit mapping between $\mathbf{Y}$ and $\hat{\mathbf{X}}$ into an implicit and continuous one. In the case of LLE, a mixture of Gaussian kernels whose parameters are optimized by an EM algorithm is proposed.

A third possibility is proposed in [16]; it is based on kernel theory and Nyström's formula.

Example

Figure 5.8 shows how LLE embeds the two benchmark manifolds introduced in Section 1.5. The dimensionality of the data sets (see Fig. 1.4) is reduced from three to two using the following parameter values: $K = 7$ and $\Delta^2 = 1e{-}2$ for the Swiss roll and $K = 4$ and $\Delta^2 = 10$ for the open box. Perhaps because of the low number of data points, both parameters require careful tuning. It is noteworthy that in the case of the open box, Δ highly differs from the proposed all-purpose value ($\Delta^2 = 1e - 4$).

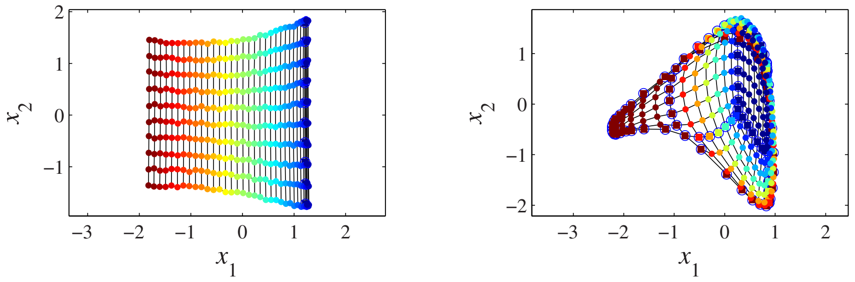


Fig. 5.8. Two-dimensional embeddings of the "Swiss roll" and "open box" data sets (Fig. 1.4), found by LLE.

Once the parameters are correctly set, the embedding looks rather good: there are no tears and the box is deformed smoothly, without superpositions. The only problem for the open box is that at least one lateral face is completely crushed.

Classification

Like MDS, Isomap, and KPCA, LLE is a batch (offline) method working with simple algebraic operations. As many other spectral methods that rely on an EVD, LLE is able to build embeddings incrementally just by appending or removing eigenvectors. The mapping provided by LLE is explicit and discrete.

In contrast with classical metric MDS, LLE assumes that data are linear locally, not globally. Hence, the model of LLE allows one to unfold nonlinear manifolds, as expected. More precisely, LLE assumes that the manifold can be mapped to a plane using a conformal mapping. Although both MDS and LLE use an EVD, which is purely linear, the nonlinear capabilities of LLE actually come from its first step: the computation of the nearest neighbors.

This operation can be interpreted as a kind of thresholding, which is nonlinear by nature.

Another essential difference between these two methods holds in the selection of the eigenvectors involved in the coordinates of the embedding. In the case of MDS, the eigenvectors associated with the largest eigenvalues are kept, whereas in LLE those with the smallest eigenvalues are taken.

Advantages and drawbacks

The greatest advantage of LLE lies in its sound theoretical foundation. The principle of the method is elegant and simple. Like Isomap, LLE can embed a manifold in a nonlinear way while sticking by an eigensolver. Importantly, even if the reconstruction coefficients for each data point are computed from its local neighborhood only, independently from other neighborhoods, the embedding coordinates involve the solution of an N-by-N eigenproblem. This means that although LLE primarily relies on local information about the manifold, a global operation couples all data points in the connected components of the graph underlying matrix $\mathbf{W}$.

From the computational viewpoint, LLE needs to compute the EVD of an N-by-N matrix, where N is the number of points in the data set. Hence, it may be feared that too large a data set may rapidly become computationally untractable. Fortunately, the matrix to be decomposed is sparse, enabling specialized EVD procedures to keep the computational load low. Furthermore, it is noteworthy that in contrast with most other methods using an EVD or SVD described in this book, LLE looks for eigenvectors associated with the *smallest* eigenvalues. In practice, this specificity combined with the fact that the matrix is large, reinforces the need for high-performance EVD procedures.

In contrast to what is claimed in [158], finding good parameters for LLE is not so easy, as reported in [166], for instance. Actually, two parameters must be tuned carefully: K, the number of neighbors (or ϵ when the neighborhoods are determined by the ϵ-rule) and Δ, the regularization factor. Depending on these two parameters, LLE can yield completely different embeddings.

Variants

Historically, LLE itself can be considered a variant or an evolution of Local PCA (LPCA [101, 32]; see also Subsection 3.3.1). The development of LLE was indeed partly motivated by the need to overcome the main shortcoming of LPCA, i.e., the fact that LPCA cannot yield an embedding in a unique coordinate system (see Subsection 2.5.8). This shortcoming prevents the use of LPCA in many applications, although the method has many advantages: it is simple, is fast, relies on well-known linear algebraic procedures, and can yet perform a nonlinear dimensionality reduction. To some extent, LLE can be seen as an LPCA without vector quantization, along with a procedure to patch together the small local linear pieces of manifold in a unique coordinate system. The coordination or alignment of locally linear models seems to be a very promising approach that is developed in [159, 189, 178, 29].

Another variant of LLE is Hessian LLE [56, 55], which overcomes some shortcomings of LLE. This variant computes locally an estimate of the manifold Hessian, for each data point and its K neighbors.

5.3.2 Laplacian eigenmaps

The method called "Laplacian eigenmaps" [12, 13] (LE in short) belongs to the now largely developed family of NLDR techniques based on spectral decomposition. The method was intended to remedy some shortcomings of other spectral methods like Isomap (Subsection 4.3.2) and LLE (Subsection 5.3.1). In contrast with Isomap, LE develops a local approach to the problem of nonlinear dimensionality reduction. In that sense, LE is closely related to LLE, although it tackles the problem in a different way: instead of reproducing small linear patches around each datum, LE relies on graph-theoretic concepts like the Laplacian operator on a graph. LE is based on the minimization of local distances, i.e., distances between neighboring data points. In order to avoid the trivial solution where all points are mapped to a single points (all distances are then zero!), the minimization is constrained.

Embedding of data set

LE relies on a single and simple hypothesis: the data set

$$\mathbf{Y} = [\ldots, \mathbf{y}(i), \ldots, \mathbf{y}(\mathbf{j}), \ldots]_{1 \leq i,j \leq N} \tag{5.48}$$

contains a sufficiently large number N of points lying on (or near) a smooth P-manifold. As only the data points are given, the manifold itself remains unknown. However, if N is large enough, the underlying manifold can be represented with good accuracy by a graph $G = (V_N, E)$. In this representation, a vertex v_i of the graph is associated with each datum $\mathbf{y}(i)$, and an edge connects vertices v_i and v_j if the corresponding data points are neighbors. The neighborhood relationships can be determined using either K-ary neighborhoods or ϵ-ball neighborhoods, as for other graph-based methods (see also Appendix E). Neighborhood relationships can be encoded in a specific data structure or more simply in a (sparse) adjacency matrix $\mathbf{A}$. The binary entries $a_{i,j} \in \{0, 1\}$ indicate whether data points $\mathbf{y}(i)$ and $\mathbf{y}(j)$ are neighbors or not. For reasons that will be given ahead, $\mathbf{A}$ must be symmetric, meaning that the graph G must be undirected.

The aim of LE is to map $\mathbf{Y}$ to a set of low-dimensional points

$$\mathbf{X} = [\ldots, \mathbf{x}(i), \ldots, \mathbf{x}(\mathbf{j}), \ldots]_{1 \leq i,j \leq N} \tag{5.49}$$

that keep the same neighborhood relationships. For this purpose, the following criterion is defined:

$$E_{\mathrm{LE}} = \frac{1}{2} \sum_{i,j=1}^{N} \|\mathbf{x}(i) - \mathbf{x}(j)\|_2^2 w_{i,j} \ , \tag{5.50}$$

where entries $w_{i,j}$ of the symmetric matrix $\mathbf{W}$ are related to those of the adjacency matrix in the following way: $w_{i,j} = 0$ if $a_{i,j} = 0$; otherwise, $w_{i,j} \geq 0$. Several choices are possible for the nonzero entries. In [12] it is recommended to use a Gaussian bell-shaped kernel

$$w_{i,j} = \exp\left(-\frac{\|\mathbf{y}(i) - \mathbf{y}(j)\|_2^2}{2T^2}\right) \ , \tag{5.51}$$

where parameter T can be thought of as a temperature in a heat kernel involved in diffusion equations. A simpler option consists of taking $w_{i,j} = 1$ if $a_{i,j} = 1$. This amounts to setting $T = \infty$ in the heat kernel.

According to the definition of $\mathbf{W}$, minimizing E_{LE} under appropriate constraints is an attempt to ensure that if $\mathbf{y}(i)$ and $\mathbf{y}(j)$ are close to each other, then $\mathbf{x}(i)$ and $\mathbf{x}(j)$ should be close as well. In other words, the topological properties (i.e., the neighborhood relationships) are preserved and the weights $w_{i,j}$ act as penalties that are heavier (resp., small or null) for close (resp., faraway) data points.

Knowing that $\mathbf{W}$ is symmetric, the criterion E_{LE} can be written in matrix form as follows:

$$E_{\mathrm{LE}} = \operatorname{tr}(\mathbf{X}\mathbf{L}\mathbf{X}^T) \ . \tag{5.52}$$

In this equation, $\mathbf{L}$ is the weighted Laplacian matrix of the graph G, defined as

$$\mathbf{L} = \mathbf{W} - \mathbf{D} \ , \tag{5.53}$$

where $\mathbf{D}$ is a diagonal matrix with entries $d_{i,i} = \sum_{j=1}^{N} w_{i,j}$. To prove the equality, it suffices to notice that for a p-dimensional embedding:

$$E_{\mathrm{LE}} = \frac{1}{2} \sum_{i,j=1}^{N} \|\mathbf{x}(i) - \mathbf{x}(j)\|_2^2 w_{i,j} \tag{5.54}$$

$$= \frac{1}{2} \sum_{p=1}^{P} \sum_{i,j=1}^{N} (x_p(i) - x_p(j))^2 w_{i,j} \tag{5.55}$$

$$= \frac{1}{2} \sum_{p=1}^{P} \sum_{i,j=1}^{N} (x_p^2(i) + x_p^2(j) - 2x_p(i)x_p(j)) w_{i,j} \tag{5.56}$$

$$= \frac{1}{2} \sum_{p=1}^{P} \left(\sum_{i=1}^{N} x_p^2(i) d_{i,i} + \sum_{j=1}^{N} x_p^2(j) d_{j,j} - 2 \sum_{i,j=1}^{N} x_p(i)x_p(j) w_{i,j} \right) \tag{5.57}$$

$$= \frac{1}{2} \sum_{p=1}^{P} 2\mathbf{f}_p^T(y)\mathbf{D}\mathbf{f}_p(y) - 2\mathbf{f}_p^T(y)\mathbf{W}\mathbf{f}_p(y) \tag{5.58}$$

$$= \frac{1}{2} \sum_{p=1}^{P} \mathbf{f}_p^T(y)\mathbf{L}\mathbf{f}_p(y) = \mathrm{tr}(\mathbf{X}\mathbf{L}\mathbf{X}^T) \ , \tag{5.59}$$

where $\mathbf{f}_p(y)$ is an N-dimensional vector giving the pth coordinate for each embedded point, and $\mathbf{f}_p(y)$ is the transpose of the pth row of $\mathbf{X}$. By the way, it is noteworthy that the above calculation also shows that $\mathbf{L}$ is positive semidefinite.

Minimizing E_{LE} with respect to $\mathbf{X}$ under the constraint $\mathbf{X}\mathbf{D}\mathbf{X}^T = \mathbf{I}_{P \times P}$ reduces to solving the generalized eigenvalue problem $\lambda\mathbf{D}\mathbf{f} = \mathbf{L}\mathbf{f}$ and looking for the P eigenvectors of $\mathbf{L}$ associated with the smallest eigenvalues. As $\mathbf{L}$ is symmetric and positive semidefinite, all eigenvalues are real and not smaller than zero. This can be seen by solving the problem incrementally, i.e., by computing first a one-dimensional embedding, then a two-dimensional one, and so on. At this point, it must noticed that $\lambda\mathbf{D}\mathbf{f} = \mathbf{L}\mathbf{f}$ possesses a trivial solution. Indeed, for $\mathbf{f} = \mathbf{1}_N$ where $\mathbf{1}_N = [1, \ldots, 1]^T$, it comes out that $\mathbf{W}\mathbf{1}_N = \mathbf{D}\mathbf{1}_N$ and thus that $\mathbf{L}\mathbf{1}_N = \mathbf{0}_N$. Hence $\lambda_N = 0$ is the smallest eigenvalue of $\mathbf{L}$ and $\mathbf{f}_N(y) = \mathbf{1}$.

An equivalent approach [16] to obtain the low-dimensional embedding (up to a componentwise scaling) consists of normalizing the Laplacian matrix:

$$\mathbf{L}' = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2} = \left[\frac{l_{ij}}{\sqrt{d_{ii}d_{jj}}} \right]_{1 \leq i,j \leq N} \ , \tag{5.60}$$

and finding directly its eigenvectors:

$$\mathbf{L}' = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^T \ . \tag{5.61}$$

The eigenvectors associated with the P smallest eigenvalues (except the last one, which is zero) form a P-dimensional embedding of the data set. The eigenvalues are the same as for the generalized eigenvalue problem, and the following relationship holds for the eigenvectors: $\mathbf{u}_i = \mathbf{D}^{1/2}\mathbf{f}_i$.

The Laplacian matrix as computed above is an operator on the neighborhood graph, which is a discrete representation of the underlying manifold. Actually, the Laplacian matrix stems from a similar operator on smooth manifolds, the Laplace-Beltrami operator. Cast in this framework, the eigenvectors $\mathbf{u}_i$ are discrete approximations of the eigenfunctions of the Laplace-Beltrami operator applied on the manifold. More details can be found in [13, 17].

Laplacian eigenmaps can be implemented with the procedure shown in Fig. 5.9. A software package in the MATLAB® language is available at `http://people.cs.uchicago.edu/~misha/ManifoldLearning/`. The parameters of Laplacian eigenmaps are the embedding dimensionality, the neighborhood type, and the corresponding parameter (K-ary neighborhoods or ϵ-balls). Depending on the kernel that yields matrix $\mathbf{W}$, additional parameters may need to be tuned, such as the temperature T in the heat kernel. Laplacian eigenmaps involve an EVD of an N-by-N nonsparse matrix. However, only a few eigenvectors need to be calculated; this reduces the time complexity when efficient linear algebra libraries are available.

1. If data consist of pairwise distances, then skip step 2 and go directly to step 3.
2. If data consist of vectors, then compute all pairwise distances.
3. Determine either K-ary neighborhoods or ϵ-ball neighborhoods.
4. Build the corresponding graph and its adjacency matrix $\mathbf{A}$.
5. Apply the heat kernel (or another one) to adjacent data points, and build matrix $\mathbf{W}$ as in Eq. (5.51).
6. Sum all columns of $\mathbf{W}$ in order to build the diagonal matrix $\mathbf{D}$, which consists of the rowwise sums of $\mathbf{W}$.
7. Compute $\mathbf{L}$, the Laplacian of matrix $\mathbf{W}$: $\mathbf{L} = \mathbf{W} - \mathbf{D}$.
8. Normalize the Laplacian matrix: $\mathbf{L}' = \mathbf{D}^{-1/2}\mathbf{L}\mathbf{D}^{-1/2}$.
9. Compute the EVD of the normalized Laplacian: $\mathbf{L}' = \mathbf{U}\boldsymbol{\Lambda}\mathbf{U}^T$.
10. A low-dimensional embedding is finally obtained by multiplying eigenvectors by $\mathbf{D}^{1/2}$, transposing them, and keeping those associated with the P smallest eigenvalues, except the last one, which is zero.

Fig. 5.9. Algorithm of Laplacian eigenmaps.

Embedding of test set

An extension of LE for new points is very briefly introduced in [16], based on the Nyström formula.

Example

Figure 5.10 shows how LE embeds the two benchmark manifolds introduced in Section 1.5. The dimensionality of the data sets (see Fig. 1.4) is reduced from three to two using the following parameter values: $K = 7$ for the Swiss roll and $K = 8$ for the open box. These values lead to graphs with more edges than the lattices shown in the figure. Moreover, the parameter that controls the graph building (K or ϵ) requires careful tuning to obtain satisfying embeddings. Matrix $\mathbf{W}$ is computed with the degenerate heat kernel ($T = \infty$). As can be seen, the Swiss roll is only partially unfolded; moreover, the third dimension of the spiral is crushed. Results largely depend on parameter K: changing it can yield embeddings of a completely different shape. In the case of the open box, the result is more satisfying although at least one face of the open box is crushed.

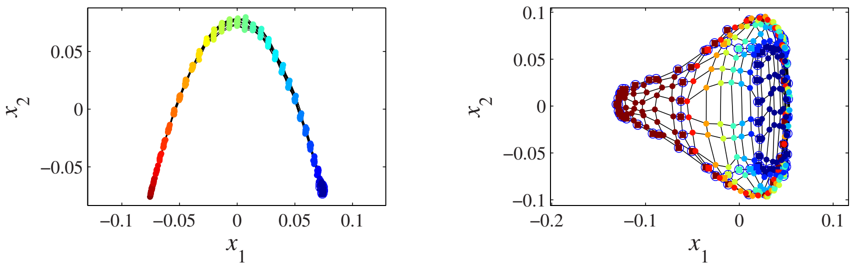


Fig. 5.10. Two-dimensional embeddings of the "Swiss roll" and "open box" data sets (Fig. 1.4), found by LE.

Classification

LE is a batch method that processes data offline. No vector quantization is integrated in it.

Like metric MDS, Isomap and LLE, LE is a spectral method. This enables LE to build embeddings in an incremental way, by adding or discarding eigenvectors. In contrast with metric MDS and Isomap, LE embeds data using the eigenvectors associated with the smallest eigenvalues, like LLE. Unfortunately, it is quite difficult to estimate the data dimensionality from those eigenvalues.

The mapping provided by LE is discrete. A continuous generalization can be obtained using the Nyström formula. The data model is discrete as well: it involves the Laplacian matrix of a graph. Replacing it with the Laplace-Beltrami operator enables LE to work on smooth manifolds.

Advantages and drawbacks

LE is almost parameter-free when the heat kernel is used with $T = +\infty$: only K or ϵ remains; this is an advantage over LLE in this respect. Nevertheless, the choice of these two last parameters may have a dramatic influence on the results of LE. Moreover, it is also possible to change the kernel function.

Like KPCA, LE usually yields poor embeddings. Connections between LE and spectral clustering (see ahead) indicate that the method performs better for data clustering than for dimensionality reduction.

Another explanation of the poor performance in dimensionality reduction can be found in its objective function. Minimizing distances between neighboring points seems to be an appealing idea at first sight. (Note, by the way, that semi-definite embedding follows exactly the opposite approach: distances between nonneighboring points are maximized; see Subsection 4.4.2 for more details.) But as shown above, this can lead to degenerate solutions, such as an embedding having identical coordinates for all data points. If this trivial solution can easily be found by looking at the equations, it is likely that other configurations minimize the objective function of LE but do not provide a suitable embedding. Intuitively, it is not difficult to imagine what such configurations should look like. For instance, assume that data points are regularly distributed on a plane, just as in an SOM grid, and that parameter K is high enough so that three aligned points on the grid are all direct neighbors of each other. Applying LE to that data set can lead to a curved embedding. Indeed, curving the manifold allows LE to minimize the distance between the first and third points of the alignment. This phenomenon can easily be verified experimentally and seriously questions the applicability of LE to dimensionality reduction.

From the computational viewpoint, LE works in a similar way as LLE, by computing a Gram-like matrix and extracting eigenvectors associated with the *smallest* eigenvalues. Therefore, LE requires robust EVD procedures. As with LLE, specific procedures can exploit the intrinsic sparsity of the Laplacian matrix.

Variants

In [13], a connection between LE and LLE is established. LE can thus be seen as a variant of LLE under some conditions. If the Laplacian matrix involved in LE approximates the Laplace-Beltrami operator on manifolds, then the result of applying twice the operator can be related to the Gram matrix built in LLE.

Diffusion maps [35, 36, 141] involve a heat kernel as in LE and follow the same approach (spectral decomposition and focus on the bottom eigenvectors). Approaches developed in [164, 206] are also closely related and connect the pseudo-inverse of the normalized graph Laplacian with so-called commute-time distances in a Markov random field. Whereas bottom eigenvectors of the normalized Laplacian $\mathbf{L}'$ are involved in LE, top eigenvectors of the pseudo-inverse of $\mathbf{L}'$ are used in [164, 206]. These papers also refer to works about electrical networks (composed of resistance only): the underlying theory is identical, except that the kernel function is inversely proportional to the distance ($w_{i,j} = \|\mathbf{y}(i) - \mathbf{y}(j)\|^{-1}$). This connection between Laplacian-based embeddings and commute-time distances or network resistances is important and means that the local approach developed in LE (K-ary neighborhoods) can be considered from a global point of view. This allows us to relate a topology-based spectral method (LE) to distance-based spectral methods (KPCA and Isomap).

Similarly, LE is also closely related to spectral clustering [143], as indicated in [13, 164, 206, 141], and to graph partitioning [172, 199, 173]. Intuitively, graph partitioning aims at dividing a graph into two parts while minimizing the "damage" (the normalized sum of edge weights). Within this framework, the eigenvector associated with the smallest nonzero eigenvalue of the normalized Laplacian matrix is a "continuous" indicator of a binary graph partition (flags are only required to be in $\mathbb{R}$ instead of being strictly binary, i.e., in $\{-1, +1\}$).

Finally, it is noteworthy that locality preserving projections (LPP [81], a.k.a. Laplacianfaces [82]) is a linear variant of Laplacian eigenmaps. This method works in the same way as PCA, by building a P-by-D transformation matrix that can be applied on any vector in $\mathbb{R}^D$. This method keeps many advantages of PCA although the objective function is different: while PCA tries to preserve the global structure of the data set, LPP attempts to preserve the local structure. This method involves K-ary neighborhoods like LE.

5.3.3 Isotop

The aim of Isotop [114] consists of overcoming some of the limitations of the SOMs when they are used for nonlinear dimensionality reduction. In this case, indeed, the vector quantization achieved by an SOM may be useless or even undesired. Moreover, an SOM usually imposes a rectangular latent space, which seldom suits the shape of the manifold to be embedded. Isotop addresses these two issues by separating the vector quantization (that becomes optional) and the dimensionality reduction. These two steps are indeed tightly interlaced in an SOM and Isotop clearly separates them in order to optimize them independently.

Embedding of data set

Isotop reduces the dimensionality of a data set by breaking down the problem into three successive steps:

1. vector quantization (optional),
2. graph building,

3. low-dimensional embedding.

These three steps are further detailed ahead. Isotop relies on a single and simple hypothesis: the data set

$$\mathbf{Y} = [\ldots, \mathbf{y}(i), \ldots, \mathbf{y}(\mathbf{j}), \ldots]_{1 \leq i,j \leq N} \tag{5.62}$$

contains a sufficiently large number N of points lying on (or near) a smooth P-manifold.

If the data set contains too many points, the first step of Isotop consists of performing a vector quantization in order to reduce the number of points (see Appendix D). This optional step can easily be achieved by various algorithms, like Lloyd's, or a competitive learning procedure. In contrast with an SOM, no neighborhood relationships between the prototypes are taken into account at this stage in Isotop. For the sake of simplicity, it is assumed that the first step is skipped, meaning the subsequent steps work directly with the raw data set $\mathbf{Y}$.

Second, Isotop connects neighboring data points (or prototypes), by using the graph-building rules proposed in Appendix E, for example. Typically, each point $\mathbf{y}(i)$ of the data set is associated with a graph vertex v_i and then connected with its K closest neighbors or with all other points lying inside an ϵ-ball. The obtained graph $G = (V_N, E)$ is intended to capture the topology of the manifold underlying the data points, in the same way as it is done in other graph-based methods like Isomap, GNLM, CDA, LLE, LE, etc. In contrast with an SOM, G may be completely different from a rectangular lattice, since it is not predefined by the user. Instead, G is "data-driven", i.e., completely determined by the available data. Moreover, until this point, no low-dimensional representation is associated with the graph, whereas it is precisely such a representation that predetermines the lattice in an SOM. Eventually, the second step of Isotop ends by computing the graph distances $\delta(v_i, v_j)$ for all pairs of vertices in the graph (see Subsection 4.3.1). These distances will be used in the third step; in order to compute them, each edge (v_i, v_j) of the graph is given a weight, which is equal to the Euclidean distance $\|\mathbf{y}(i) - \mathbf{y}(j)\|$ separating the corresponding points in the data space. The graph distances approximate the geodesic distances in the manifold, i.e., the distances between points *along* the manifold.

The third step of Isotop is the core of the method. While the first and second steps aim at converting the data, given as D-dimensional coordinates, into graph G, the third step achieves the inverse transformation. More precisely, the goal consists of translating graph G into P-dimensional coordinates. For this purpose, the D-dimensional coordinates $\mathbf{y}(i)$ associated with the vertices of the graph are replaced with P-dimensional coordinates $\mathbf{x}(i)$, which are initialized to zero. At this stage, the low-dimensional representation $\mathbf{X}$ of $\mathbf{Y}$ is built but, obviously, $\mathbf{X}$ does not truly preserve the topology of $\mathbf{Y}$ yet: it must be modified or updated in some way. For this purpose, $\mathbf{Y}$ may be forgotten henceforth: Isotop will use only the information conveyed by G in order to update $\mathbf{X}$.

In order to unfold the low-dimensional representation $\mathbf{X}$ associated with G, a Gaussian kernel $N(\mathbf{x}(i),\mathbf{I})$ of unit variance is centered on each point $\mathbf{x}(i)$. The normalized sum of the N Gaussian kernels gives a distribution that itself is a Gaussian kernel just after the initialization, since $\mathbf{x}(i)=0$ for all i in $\{1,\ldots,N\}$. The main idea of Isotop then consists of performing a vector quantization of that distribution, in which $\mathbf{X}$ plays the role of the codebook and is updated using a similar learning rule as in an SOM. More precisely, the three following operations are carried out:

1. Randomly draw a point $\mathbf{r}$ from the distribution.
2. Determine the index j of the nearest point from $\mathbf{r}$ in the codebook, i.e.,
$$j = \arg\min_i = d(\mathbf{r},\mathbf{x}(i)) \ , \tag{5.63}$$
where d is typically the Euclidean distance.
3. Update all points $\mathbf{x}(i)$ according to the rule
$$\mathbf{x}(i) \leftarrow \mathbf{x}(i) + \alpha\nu_\lambda(i,j)(\mathbf{r}-\mathbf{x}(i)) \ , \tag{5.64}$$
where the learning rate α, which satisfies $0 \leq \alpha \leq 1$, plays the same role as the step size in a Robbins–Monro procedure.

In the update rule (5.64), $\nu_\lambda(i,j)$ is called the neighborhood function and is defined as follows:

$$\nu_\lambda(i,j) = \exp\left(-\frac{1}{2}\,\frac{\delta^2_{\mathbf{y}}(i,j)}{\lambda^2\ \mu^2_{(v_h,v_j)\in E}(\delta_{\mathbf{y}}(h,j))}\right) \ , \tag{5.65}$$

where the first factor λ of the denominator is a neighborhood width, acting and scheduled exactly as in an SOM. The second factor of the denominator is simply the square of the mean graph distance between the vertices v_h and v_j (or equivalently, between the points $\mathbf{y}(i)$ and $\mathbf{y}(j)$) if the edge (v_h,v_j) belongs to E. In other words, this factor is the square distance between v_j and its direct neighbors in the graph. Because the neighbors are direct and the edges are labeled with the Euclidean distance:

$$\mu_{(v_h,v_j)\in E}(\delta_{\mathbf{y}}(h,j)) = \mu_{(v_h,v_j)\in E}(d_{\mathbf{y}}(h,j)) = \mu_{(v_h,v_j)\in E}(\|\mathbf{y}(h)-\mathbf{y}(j)\|) \ . \tag{5.66}$$

The second factor of the denominator aims at normalizing the numerator $\delta^2_{\mathbf{y}}(i,j)$, in order to roughly approximate the *relative* distances from $\mathbf{y}(j)$ to $\mathbf{y}(i)$ inside the manifold. Without this factor, Isotop would depend too much on the local density of the points on the manifold: smaller (resp., larger) distances are measured in denser (resp., sparser) regions.

Typically, the three steps above are repeated N times with the same values for the parameters α and λ; such a cycle may be called an *epoch*, as for other adaptive algorithms following the scheme of a Robins-Monro procedure [156]. Moreover, instead of drawing $\mathbf{r}$ from the normalized sum of Gaussian kernels

N times, it is computationally easier and statistically more interesting to draw it successively from each of the N kernels, in random order. Doing so allows us to generate $\mathbf{r}$ by adding white Gaussian noise successively to each point $\mathbf{x}(i)$. This also allows us to visit more or less equally each region of the distribution during a single epoch.

Intuitively, the above learning rule unfolds the connected structure in a low-dimensional space, trying to preserve the neighborhoods. As a side effect, the mixture of Gaussian distributions evolves concurrently in order to capture the shape of the manifold.

Figure 5.11 shows a procedure that implements Isotop. A C++ implementation of Isotop can also be downloaded from `http://www.ucl.ac.be/mlg/`. Isotop inherits parameters from the vector quantization performed in its first step. Other parameters are related to the graph-building step (K for K-ary neighborhood or ϵ for ϵ-balls). Like an SOM, Isotop also involves parameters in its learning phase (number of iterations, learning rate, neighborhood width, and neighborhood function). Regarding space and time complexities, Isotop requires computing pairwise geodesic distances for all prototypes; this demands $\mathcal{O}(M^2)$ memory entries and $\mathcal{O}(M^2 \log M)$ operations, where $M \leq N$ is the number of prototypes. Once geodesic distances are known, the learning phase of Isotop is relatively fast: the time complexity of each epoch is $\mathcal{O}(M^2)$. Finally, some intuitive arguments that explains why the procedure detailed in Fig. 5.11 works are gathered hereafter.

1. Perform a vector quantization of the data set; this step is optional and can be done with any standard quantization method.
2. Build a graph structure with an appropriate rule (see Appendix E), and compute all pairwise graph distances.
3. Initialize low-dimensional coordinates $\mathbf{x}(i)$ of all vertices v_i to zero.
4. Initialize the learning rate α and the neighborhood width λ with their scheduled values for epoch number q.
5. For each vertex v_i in the graph,
   - Generate a point $\mathbf{r}$ randomly drawn from a Gaussian distribution centered on the associated coordinates $\mathbf{x}(i)$.
   - Compute the closest vertex from $\mathbf{r}$ according to Eq. (5.63).
   - Update the coordinates of all vertices according to rule (5.64).
6. Increase q and return to step 4 if convergence is not reached.

Fig. 5.11. Isotop algorithm.

The vertices do not collapse on each other.

As the update rule acts as an attractive force between the prototypes, it may be feared that the third step of Isotop will yield a trivial solution: all low-dimensional coordinates converge on the same point. Fortunately, however, this does not happen, because Isotop involves Gaussian distributions centered on each vertex in the low-dimensional embedding space. The update rule takes into account points drawn from the distributions in order to move the vertices. This means that near the "boundary" of the graph some random points can move vertices outward. Actually, the entire "boundary" of the embedded graph contributes to stretching it; this effect balances the attractive force induced by the update rule. If Isotop's third step is compared to an SOM running in a low-dimensional space, then to some extent Isotop performs a vector quantization on a (dynamically varying) mixture of Gaussian distributions. In the worst case, all Gaussian kernels are superposed; but still in this situation, the prototypes will disperse and try to reproduce the mixture of Gaussian kernels (only an infinite neighborhood width impeaches this process). Since the dispersion of the prototypes is ensured, they may all be initialized to zero.

The vertices do not infinitely expand.

On the other side, the vertices do not diverge toward infinitely faraway positions. This could only happen if the Gaussian kernels centered on the prototypes were infinitely large. The probability to draw a point from a Gaussian distribution lying very far away from its center is very low. Eventually, as long as the neighborhood width is kept larger than zero, the update rule generates an attractive force between the vertices that limits their expansion.

Embedding of test set

Isotop does not provide a generalization procedure. Actually, the generalization to new points is difficult, as it is for an SOM, because Isotop can heavily deform the graph structure, even on a local scale. Fortunately, the generalization problem appears less critical for Isotop than for an SOM. Because vector quantization is a mandatory step in the latter, even the embedding of the data points is not known.

Example

Figure 5.12 shows how Isotop embeds the two benchmark manifolds introduced in Section 1.5. The graph is built using ϵ-balls, in order to obtain the graphs shown in the figure. No superpositions occur in the obtained embeddings. In the case of the open box, the bottom face is shrunk, allowing the lateral faces to be embedded without too many distortions; similarily, the box lid is stretched but remains square. The two examples clearly illustrate the ability of Isotop to either preserve the manifold shape (in the case of the Swiss roll) or deform some regions of the manifold if necessary (for the box).

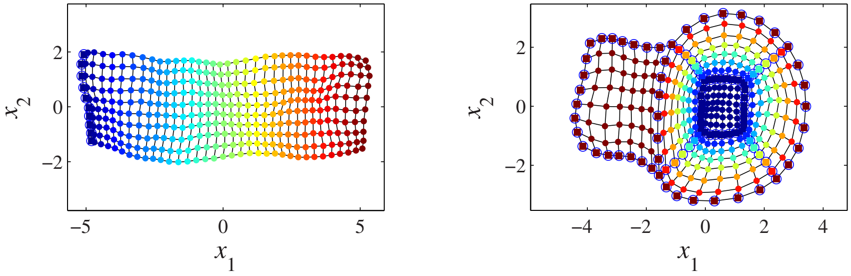


Fig. 5.12. Two-dimensional embeddings of the “Swiss roll” and “open box” data sets (Fig. 1.4), found by Isotop.

Classification

Isotop shares many characteristics with an SOM. Both methods rely on a nonlinear model and use vector quantization (optional in the case of Isotop). They also both belong to the world of artificial neural networks and use approximate optimization techniques. Because Isotop is divided into three successive steps, it cannot easily be implemented as an online algorithm, unlike a SOM.

The mapping Isotop produces between the high- and low-dimensional spaces is discrete and explicit. Hence, the generalization to new points is not easy.

Advantages and drawbacks

The comparison between Isotop and an SOM is unavoidable since both methods are closely related. Actually, Isotop can be interpreted as an SOM working in “reverse gear”. Indeed, the following procedure describes how a SOM works:

1. Determine the shape of the low-dimensional embedding (usually a two-dimensional grid of regularly spaced points).
2. Build a lattice between the grid points (this second step is often implicitly merged with the first one).
3. Perform a vector quantization in the high-dimensional space in order to place and deform the lattice in the data cloud; this step defines the mapping between the high- and low-dimensional spaces.

As can be seen, data are involved only (and lately) in the third step. In other words, an SOM weirdly starts by defining the shape of the low-dimensional embedding, without taking the data into account! In the framework of dimensionality reduction, this approach goes in the opposite direction compared to the other usual methods. Isotop puts things back in their natural order and the three above steps occur as follows:

1. Perform a vector quantization in the high-dimensional space (no lattice or neighborhoods intervene in this first step, which is performed by standard quantization methods); this step is optional.
2. Build a lattice (or graph) between the prototypes obtained after the quantization according to their coordinates in the high-dimensional space.
3. Determine the shape of the low-dimensional embedding; this step defines the mapping between the high- and low-dimensional spaces.

As in an SOM, the mapping is determined in the third step. However, data intervene earlier, already in the first step. Moreover, it is noteworthy that Isotop uses the SOM update rule in the low-dimensional space. Besides the processing order, this is a second and very important difference between Isotop and an SOM. The advantage of working in the low-dimensional space, as do most other DR methods, is clearly demonstrated in Section 6.1.

Another advantage of Isotop over the SOM is that the quantization is optional and not interlaced with the computation of the embedding. This clear separation between both tasks allows a better control of each of them (parameters, convergence, etc.).

In the current development state of Isotop, it is not known whether the method optimizes a well-defined objective function. The same problem has been remarked in [57] for the SOMs. However, it is hoped that an objective function could be found, because the way Isotop works allows the introduction of many simplifications. For example, the quantization of the Gaussian mixtures does not matter in the third step of Isotop, since the kernels follow the embedded points. Hence, the definition of an objective function may focus on topology preservation only. In addition, the third step of Isotop takes place in a space whose dimensionality is a priori equal to the dimensionality of the manifold to be embedded.

Variants

To some extent, Isotop can be related to spring-based layouts and other graph-embedding techniques. See, for example, [52].

Some "historical" aspects regarding the development of Isotop can be found in [119]. In earlier versions, only one Gaussian kernel was used in order to unfold the manifold and counterbalance the attractive force induced by Eq. (5.64).

Like GTM, which was proposed as a principled variant of Kohonen's SOM, stochastic neighbor embedding (SNE) [87] can be seen as a principled version of Isotop. SNE follows a probabilistic approach to the task of embedding and, like Isotop, associates a Gaussian kernel with each point to be embedded. The set of all these kernels allows SNE to model the probability of one point to be the neighbor of the others. This probability distribution can be measured for each point in the high-dimensional data space and, given a (random) embedding, corresponding probability distributions can also be computed in the low-dimensional space. The goal of SNE is then to update the embedding in order to match the distributions in both high- and low-dimensional spaces. In practice, the objective function involves a sum of Kullback-Leibler divergences, which measure the “distances” between pairs of corresponding distributions in their respective space. Because the minimization of the objective function is difficult and can get stuck in local minima, SNE requires complex optimization techniques to achieve good results. A version of SNE using graph distances instead of Euclidean ones in the data space is mentioned in [186, 187] and compared to other NLDR methods.

6

Method comparisons

Overview. This chapter illustrates all NLDR methods that are decribed in the previous two chapters with both toy examples and real data. It also aims to compare the results of the different methods in order to shed some light on their respective strengths and weaknesses.

6.1 Toy examples

Most examples in the next subsections are two-manifolds embedded in a three-dimensional space. The most popular of them is undoubtedly the Swiss roll. Those simple manifolds help to clearly visualize how the different methods behave in order to reduce the dimensionality.

6.1.1 The Swiss roll

The standard Swiss roll

The Swiss roll is an illustrative manifold that has been used in [179, 180] as an example to demonstrate the capabilities of Isomap. Briefly put, it is a spiral with a third dimension, as shown in Fig. 6.1. The name of the manifold originates from a delicious kind of cake made in Switzerland: jam is spread on a one-centimeter-thick layer of airy pastry, which is then rolled up on itself. With some imagination, the manifold in Fig. 6.1 can then be interpreted as a very thick slice of Swiss roll, where only the jam is visible.

The parametric equations that generate the Swiss roll are

$$\mathbf{y} = \begin{bmatrix} \sqrt{2+2x_1}\cos(2*\pi*\sqrt{2+2x_1}) \\ \sqrt{2+2x_1}\sin(2*\pi*\sqrt{2+2x_1}) \\ 2x_2 \end{bmatrix} \quad , \tag{6.1}$$

where $\mathbf{x} = [x_1, x_2]^T$ is uniformly distributed in the interval $[-1, +1]^2$. The purpose of the square root in EQ. 6.1 is to obtain a uniform distribution on

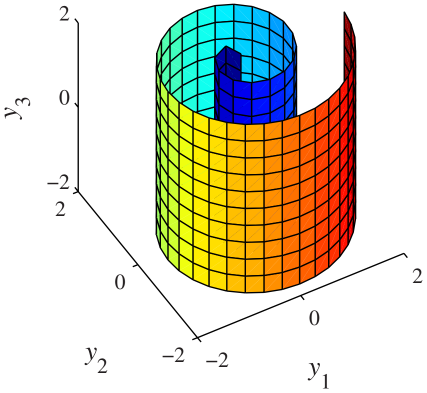


Fig. 6.1. The "Swiss roll" manifold.

the manifold as well. As can be easily seen, each coordinate of the manifold depends on a single latent variable. Hence, the Swiss roll is a developable two-manifold embedded in $\mathbb{R}^3$.

The Swiss roll is the ideal manifold to demonstrate the benefits of using graph distances. Because it is developable, all methods using graph distances can easily unfold it and reduce its dimensionality to two. On the contrary, methods working with Euclidean distances embed it with difficulty, because it is heavily crumpled on itself. For example, it has been impossible to get convincing results with NLM or CCA. Figure 6.2 shows the best CCA embedding obtained using Euclidean distances. This result has required carefully tuning the method parameters. In light of this result, Euclidean distance-preserving methods (MDS, NLM, CCA) are discarded in the experiments ahead.

Concretely, 5000 points or observations of $\mathbf{y}$ are made available in a data set. These points are generated according to Eq. 6.1. The corresponding points in the latent space are drawn randomly; they are not placed on a regular grid, as was the case for the Swiss roll described in Section 1.5 to illustrate the methods described in Chapters 4 and 5. As the number of points is relatively high, a subset of fewer than 1000 points is already representative of the manifold and allows the computation time to be dramatically decreased. As all methods work with N^2 distances or an N-by-N Gram-like matrix, they work 25 times faster with 1000 points than with 5000. In [180] the authors suggest choosing the subset of points randomly among the available ones. Figure 6.3 shows the results of Isomap, GNLM, and CDA with a random subset of 800 points. The graph distances are computed in the same way for three methods, i.e., with the K-rule and $K = 5$. Other parameters are left to their default "all-purpose" values.

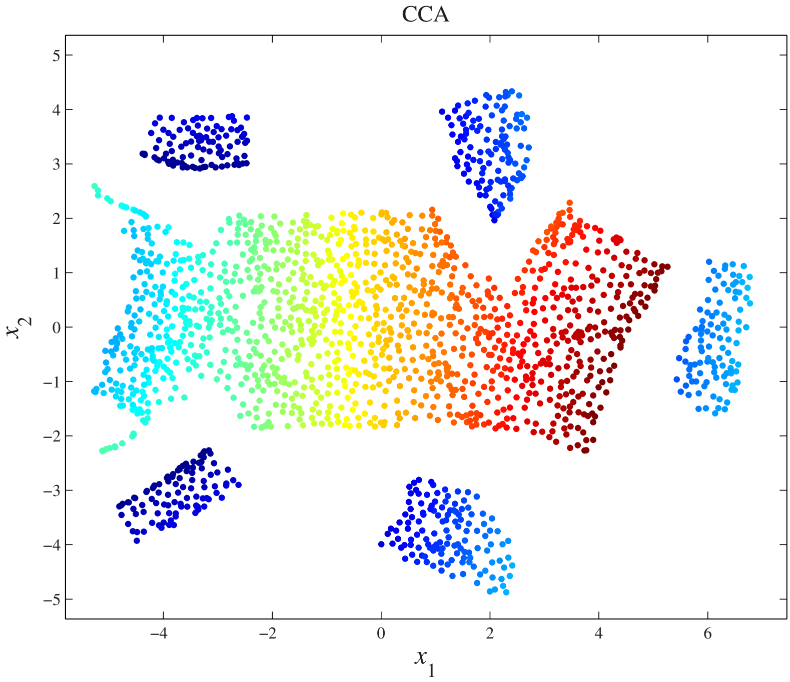


Fig. 6.2. Two-dimensional embedding of the “Swiss roll” manifold by CCA, using 1800 points.

As expected, all three methods succeed in unfolding the Swiss roll. More importantly, however, the random subset used for the embeddings is not really representative of the initial manifold. According to Eq. (6.1), the distribution of data points is uniform on the manifold. At first sight, points seem indeed to be more or less equally distributed in all regions of the embeddings. Nevertheless, a careful inspection reveals the presence of holes and bubbles in the distribution (Brand speaks of manifold “foaming” [30]). It looks like jam in the Swiss roll has been replaced with a slice of ... Swiss cheese [117, 120]! This phenomenon is partly due to the fact that the effective data set is resampled, i.e., drawn from a larger but finite-size set of points. As can be seen, Isomap amplifies the Swiss-cheese effect: holes are larger than for the two other methods. The embeddings found by GNLM and CDA look better.

Instead of performing a random subset selection, vector quantization (see Appendix D) could be applied to the Swiss roll data set. If the 800 randomly chosen points are replaced with only 600 prototypes, obtained with a simple competitive learning procedure, results shown in Fig. 6.4 can be obtained. As can be seen, the embeddings are much more visually pleasing. Beyond the visual feeling, the prototypes also represent better the initial manifold

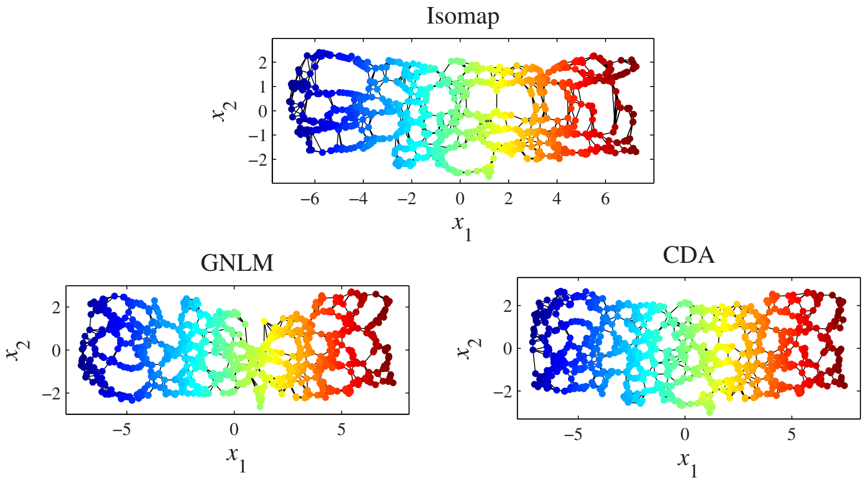


Fig. 6.3. Two-dimensional embedding of the "Swiss roll" manifold by Isomap, GNLM, and CDA. A subset of 800 points is randomly chosen among the 5000 available points.

than randomly chosen points. This results, for example, in embeddings with neater corners and almost perfectly rectangular shapes, thus reproducing more faithfully the latent space. Such results definitely pledge in favor of the use of vector quantization when the size of the data set can be or has to be reduced. Accordingly, vector quantization is always used in the subsequent experiments unless otherwise specified.

Figure 6.5 shows the embeddings obtained by topology-preserving methods (an SOM, GTM, LLE, LE, and Isotop) with the same set of prototypes, except for the SOM, since it is a vector quantization method by nature. By doing so, it is hoped that the comparison between all methods is as fair as possible. The SOM (15-by-40 map) and GTM (40-by-40 latent grid, 10×10 kernels, 200 EM iterations) suffer from jumps and shortcuts joining the successive whorls. LLE ($K = 5$, $\Delta = 0.02$) unfolds the Swiss roll and perfectly preserves the topology. As often happens with LLE, the embedding has a triangular or cuneiform shape (see [30] for a technical explanation). Finally, Isotop yields a visually pleasant result: the topology is perfectly preserved and even the rectangular shape of the Swiss roll remains visible.

Some flaws of spectral methods like Isomap, SDE, LLE, and LE can be explained by inspecting 3D embeddings instead of 2D ones, such as those shown in Fig. 6.6. Such 3D embeddings can be obtained easily starting from the 2D one, knowing that spectral methods can build embeddings incrementally just by taking into account an additional eigenvector of the Gram-like matrix. While the embeddings of SDE appear almost perfect (the third dimension is negligible), the result of Isomap shows that the manifold is not totally made

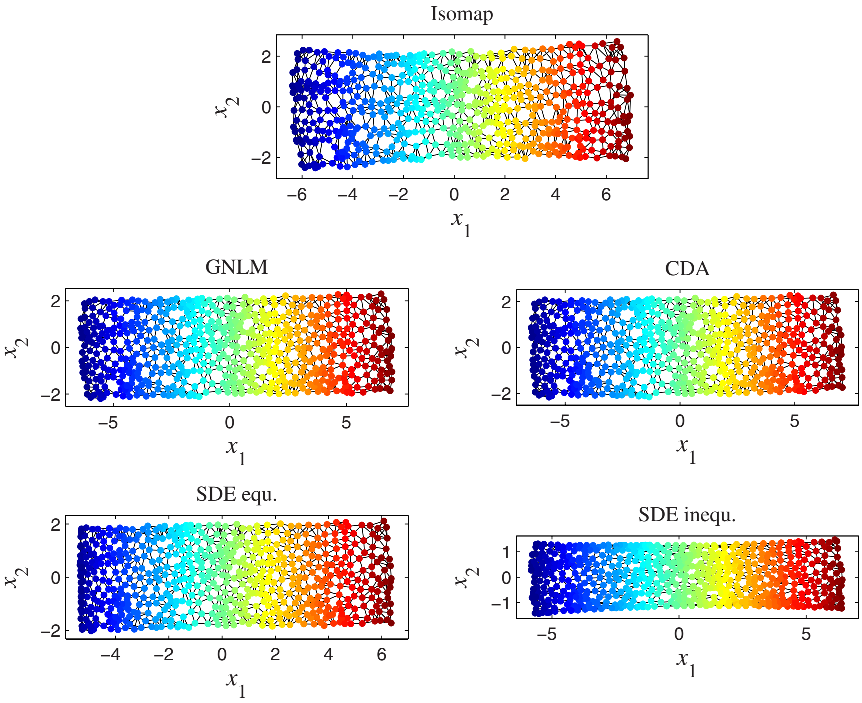


Fig. 6.4. Two-dimensional embeddings of the "Swiss roll" manifold by Isomap, GNLM, CDA, and SDE (with both equality and inequality constraints). The three methods embed 600 prototypes obtained beforehand with vector quantization.

flat. The "thickness" of the third dimension can be related to the importance of discrepancies between graph distances and true geodesic ones. These discrepancies can also explain the Swiss-cheese effect: long graph distances follow a zigzagging path and are thus longer than the corresponding geodesic distances. On the contrary, short graph distances, involving a single graph edge, can be slightly shorter than the true geodesic length, which can be curved. For spectral topology-preserving methods, the picture is even worse in 3D. As can be seen, the LLE embedding looks twisted; this can explain why LLE often yields a triangular or cuneiform embedding although the latent space is known to be square or rectangular. So LLE unrolls the manifold but introduces other distortions that make it not perfectly flat. The LE embedding is the worst: this method does not really succeed in unrolling the manifold.

Obviously, until here, all methods have worked in nice conditions, namely with an easy-to-embed developable manifold and with standard values for their parameters. Yet even in this ideal setting, the dimensionality reduction may be not so easy for some methods.

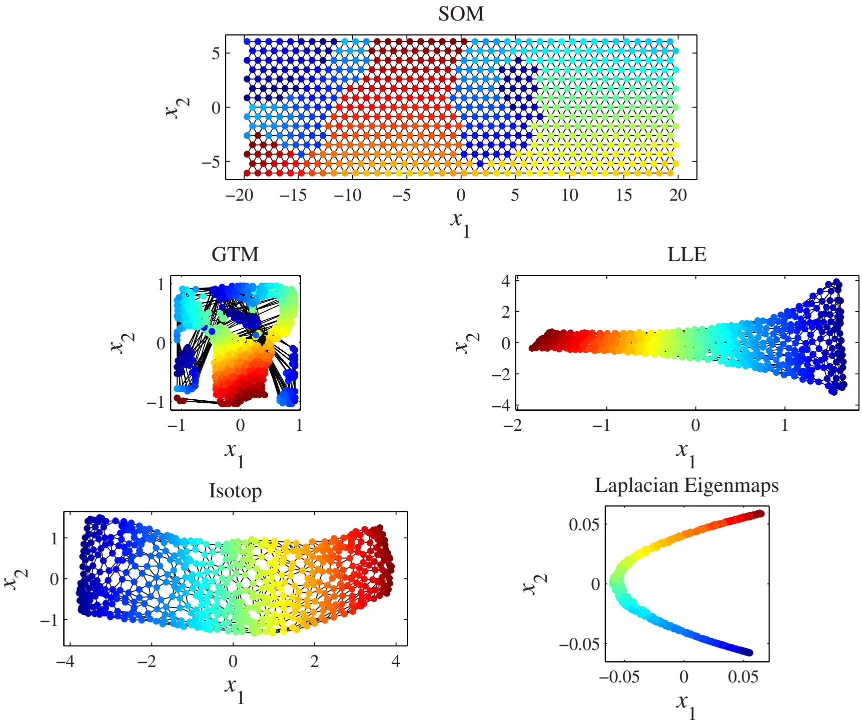


Fig. 6.5. Two-dimensional embeddings of the "Swiss roll" manifold by a SOM, GTM, LLE, LE, and Isotop. The data set consists of 600 prototypes resulting from a vector quantization on 5000 points of the manifold. The 5000 points are given as such for the SOM.

The rolled Japanese flag

Another more difficult example consists of removing a well-chosen piece of the Swiss roll. More precisely, a disk centered on $[0, 0]^T$, with radius 0.5, is removed from the latent space, which then looks like the Japanese flag, as shown in the first plot of Fig. 6.7. This flag can be rolled using Eq. (6.1); vector quantization is then applied on the obtained three-dimensional points. Next, all methods reduce the dimensionality back to two, as shown in Figs. 6.8 and 6.9, and the embedding can be compared with the Japanese flag. Unfortunately, the initial aspect ratio of the latent space is lost due to the particular form of the parametric equations. Such a scaling cannot, of course, be retrieved. What appears more surprising is that the region where the hole lies looks vertically stretched. This phenomenon is especially visible for Isomap, a bit less for GNLM, and nearly absent for CDA. For SDE, depending on the use of equalities or inequalities, the second dimension can be shrunk.

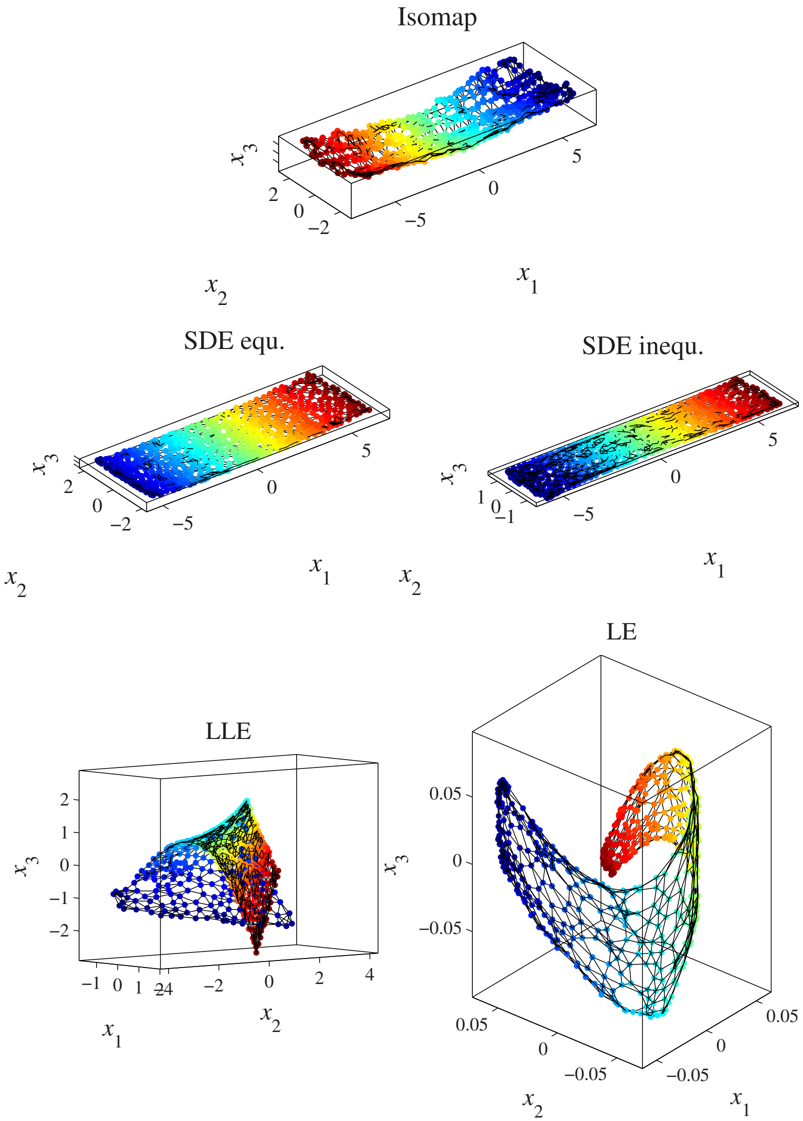


Fig. 6.6. Three-dimensional embeddings of the "Swiss roll" manifold by Isomap, SDE, LLE, and LE.

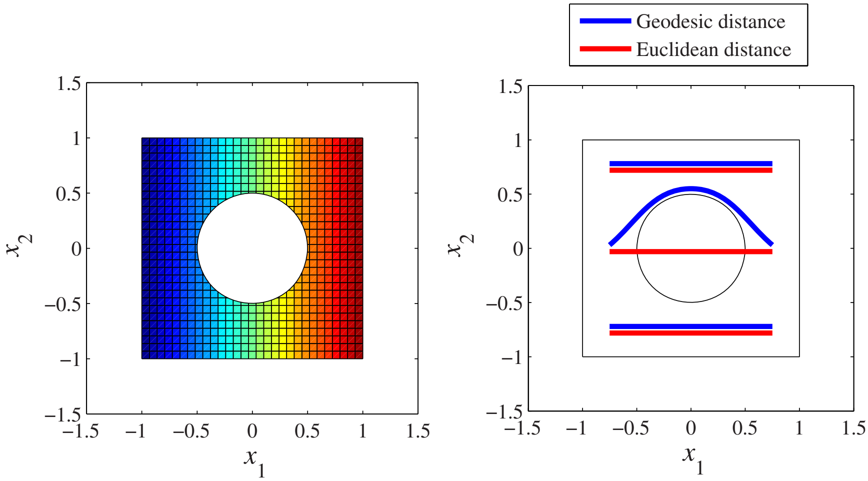


Fig. 6.7. On the left: latent space for the "Japanese flag" manifold. On the right: the fact that geodesic (or graph) distances (blue curve) are replaced with Euclidean ones (red line) in the embedding space explains the stretched-hole phenomenon that appears when the Japanese flag is embedded. The phenomenon is visible in Fig. 6.8.

For methods relying on graph distances, the explanation is rather simple: as mentioned in Section 4.3, they measure graph distances in the data space but use Euclidean distances in the embedding space, because the latter metric offers nice properties. For nonconvex manifolds like the Japanese flag, Euclidean distances cannot be made equal to the measured graph distances for some pairs of points; this is illustrated by the second plot of Fig. 6.7. Although the manifold is already unrolled, the geodesic distance remains longer than the corresponding Euclidean one because it circumvents the hole. As a consequence, NLDR methods try to preserve these long distances by stretching the hole. In particular, a spectral method like Isomap, whose ability to embed curved manifolds depends exclusively on the way distances are measured or deformed, is misleading. More complex methods, which can already deal with nonlinear manifolds without graph distances, like GNLM and CDA, behave better. They favor the preservation of short distances, which are less affected by the presence of holes. The SDE case is more complex: with strict equalities, the method behaves perfectly. When local distances are allowed to decrease, the whole manifold gets shrunk along the second dimension.

It is noteworthy that the hole in the Japanese flag manifold in Fig. 6.7 also explains the poor results of Isomap when it uses resampling instead of vector

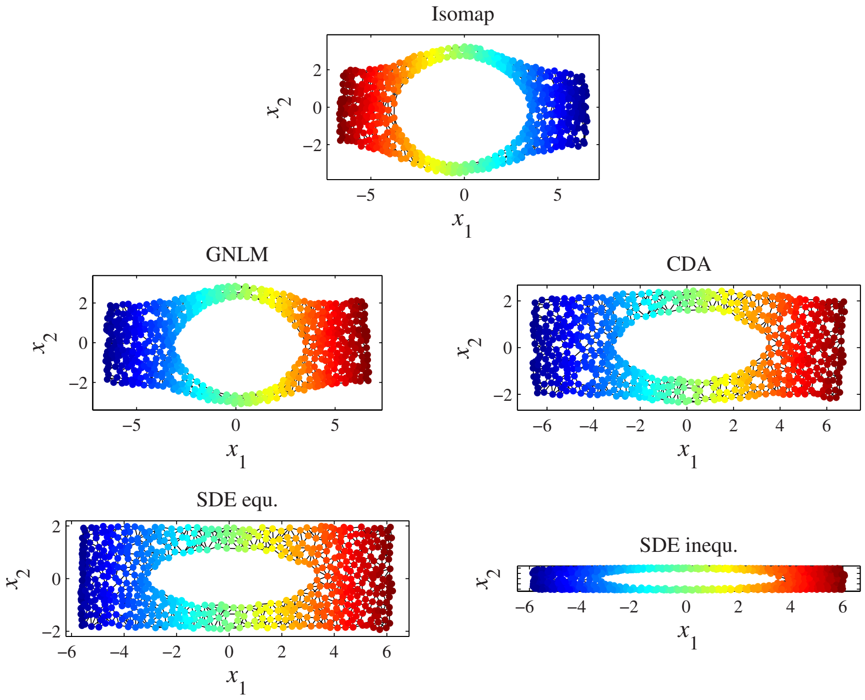


Fig. 6.8. Two-dimensional embeddings of the rolled Japanese flag obtained with distance-preserving NLDR methods. All methods embed 600 prototypes obtained beforehand with vector quantization.

quantization, as in Fig. 6.3. Small holes appear in the manifold distribution, and Isomap tends to stretch them; as a direct consequence, denser regions become even denser.

Regarding topology-preserving methods illustrated in Fig. 6.9, the results of the SOM and GTM do not require particular comments: discontinuities in the embeddings can clearly be seen. LLE produces a nice result, but it can easily be guessed visually that the manifold is not perfectly unrolled. Isotop also yields a good embedding: the topology is well preserved though some distortions are visible. Finally, the embedding provided by LE looks like Isomap's one. Considering LE as a method that preserves commute-time distances, this kind of result is not surprising. Since both ends of the Japanese flag are nearly disconnected, the commute time from one to the other is larger than required to embed correctly the manifold.

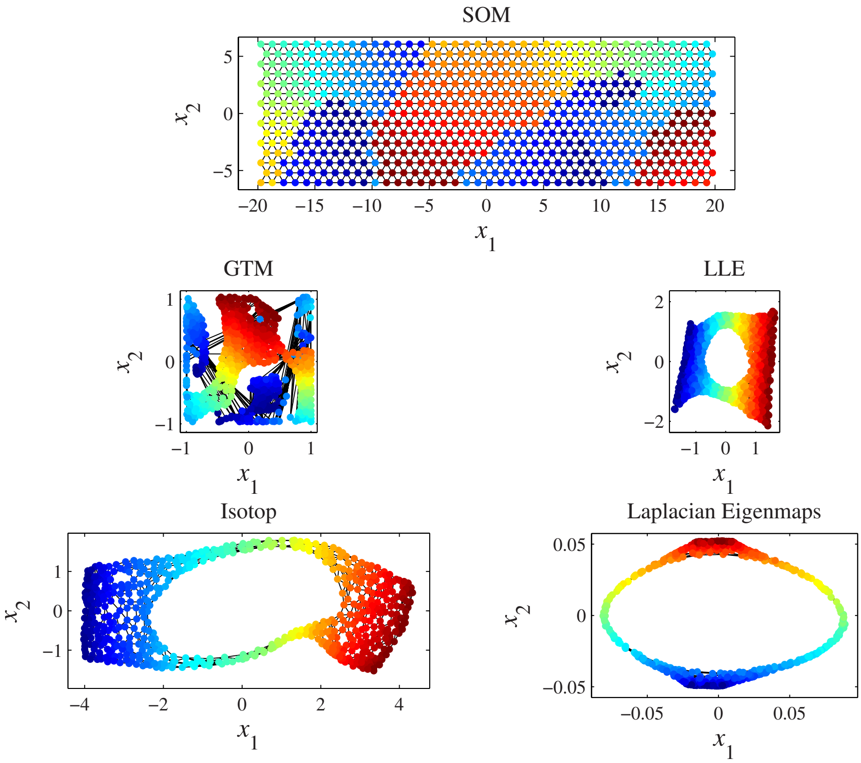


Fig. 6.9. Two-dimensional embeddings of the rolled Japanese flag obtained with topology-preserving NLDR methods. All methods embed 600 prototypes obtained beforehand with vector quantization. The 5000 points are given as such for the SOM.

The thin Swiss roll slice

Light can be shed on other flaws of NLDR methods by considering a very thin slice of a Swiss roll, as illustrated in Fig. 6.10. Actually, this manifold is identical to the one displayed in Fig. 6.1 and generated by Eq. (6.1), except that its height is divided by four, i.e., $y_3 = x_2/2$. NLDR methods embed the thin slice as shown in Figs. 6.11 and 6.12. The best result is undoubtedly given by SDE with strict equalities (with distances allowed to shrink, the embedding almost loses its second dimension and looks like a sine wave). GNLM and CDA produce twisted embeddings. On the other hand, Isomap suffers from a weird "bone effect": both ends of the rectangular manifold are wider than its body. Again, these two phenomena can be explained by the fact that graph distances only approximate true geodesic ones. More precisely, the length of a long, smooth curve is replaced with the length of a broken line, which is longer since it zigzags between the few available points in the data

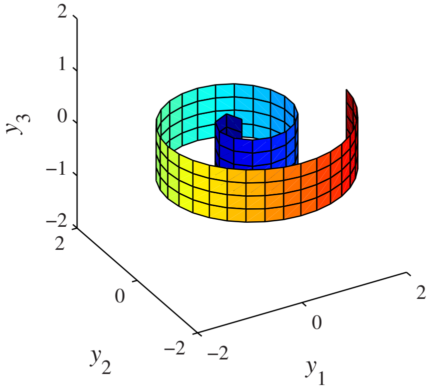


Fig. 6.10. A very thin slice of Swiss roll.

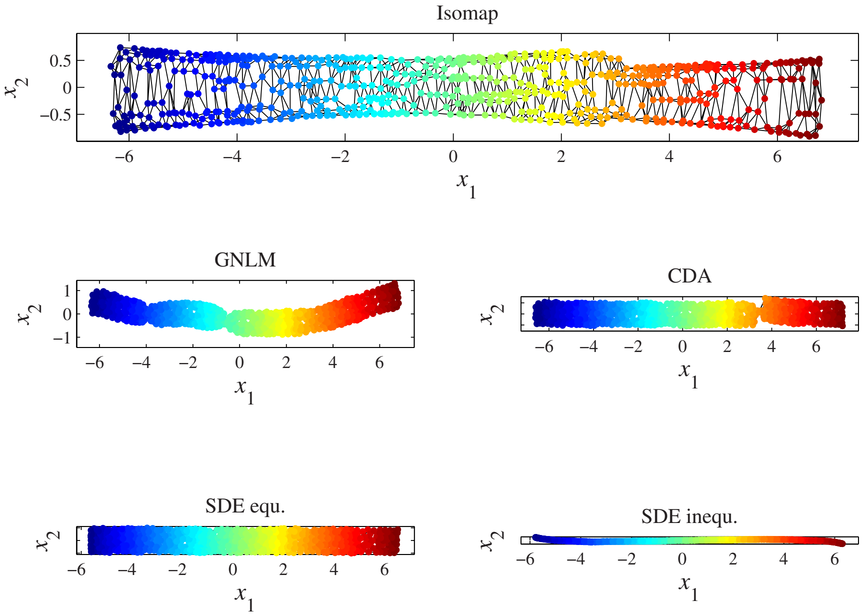


Fig. 6.11. Two-dimensional embedding of a thin slice of Swiss roll by distance-preserving methods. All methods embed 600 prototypes obtained beforehand with vector quantization.

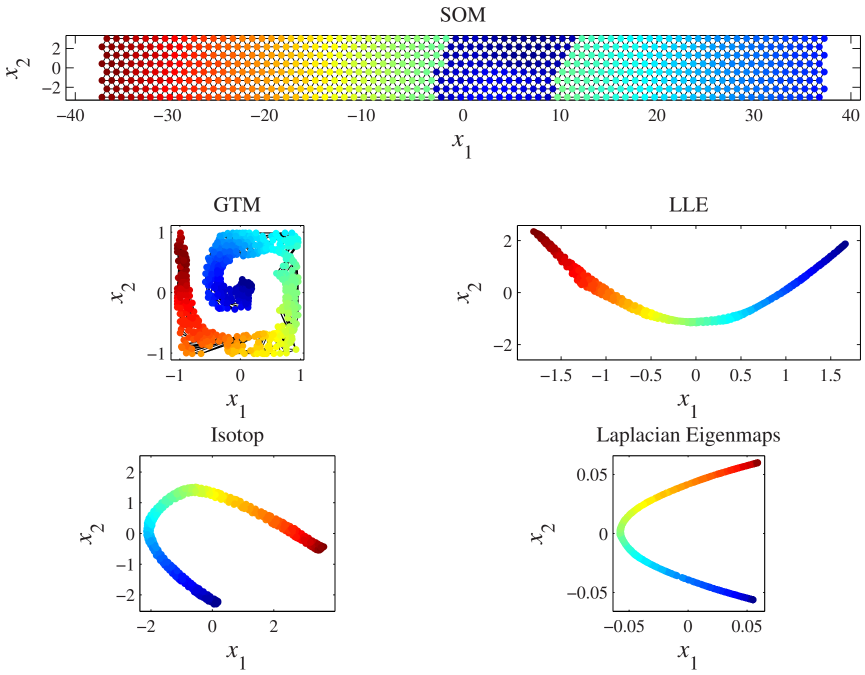


Fig. 6.12. Two-dimensional embedding of a thin slice of Swiss roll by topology-preserving methods. All methods embed 600 prototypes obtained beforehand with vector quantization. The 5000 points are given as such for the SOM.

set. Moreover, in the case of the thin Swiss roll, these badly approximated geodesics are more or less oriented in the same way, along the main axis of the rectangular manifold. Isomap tries to find a global tradeoff between overestimated long distances and well-approximated short ones by stretching vertically the manifold. GNLM and CDA act differently: as they favor the preservation of small distances, they cannot stretch the manifold; instead, they twist it.

Topology-preserving methods provide poor results, except GTM, which succeeds better in embedding the thin Swiss roll than the normal one does. Still, the embedding is twisted and oddly stretched; this is because the Swiss roll must be cast inside the square latent space assumed by GTM. Changing the shape of the latent space did not lead to a better embedding. Clearly, the SOM does not yield the expected result: the color does not vary smoothly along the rectangular lattice as it does along the Swiss roll. As for the standard Swiss roll, the SOM lattice “jumps” between the successive whorls of the manifold, as illustrated in Fig. 6.13. Actually, the SOM tries to occupy the same regions as the thin Swiss roll, but those jumps break the perfect preservation

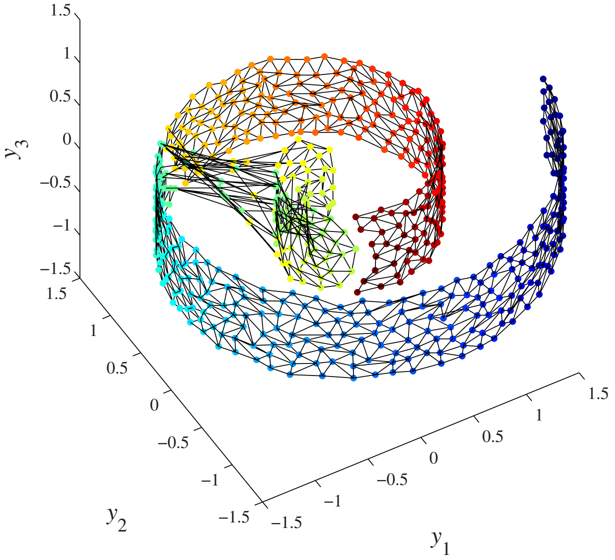


Fig. 6.13. Three-dimensional view showing how a 8-by-75 SOM typically unfurls in a thin slice of a Swiss roll.

of neighborhoods. These "shortcuts" are possible because the SOM works in the high-dimensional space of data. Other methods do not unroll the manifold completely. This can easily be understood, as distances are not taken directly into account. Constraints on the shape of the manifold are thus almost completely relaxed, and such a longer-than-wide manifold can easily be embedded in two dimensions without making it perfectly straight.

Inadequate parameter values

Still another flaw of many NLDR methods lies in a wrong or inappropriate setting of the parameters. Most methods define local K-ary or ϵ-ball neighborhoods. Until now, the value of K in the K-rule that builds the graph has been given an appropriate value (according to the number of available data points, the manifold curvature, the data noise, etc.). Briefly put, the graph induced by the neighborhoods is representative of the underlying manifold. For example, what happens if K is set to 8 instead of 5? The resulting graph is shown in Fig. 6.14. An undesired or "parasitic" edge appears in the graph; it connects an outer corner of the Swiss roll to a point lying on the next whorl. At first sight, one can think that this edge is negligible, but actually

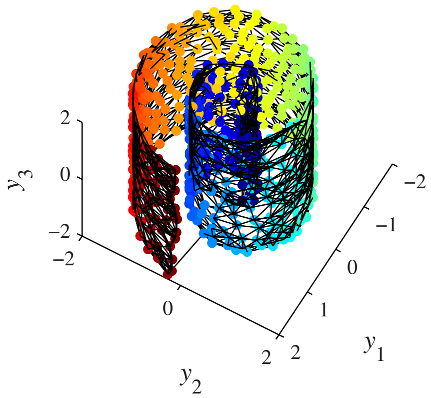


Fig. 6.14. After vector quantization, the K-rule weaves a graph that connects the 600 prototypes. Because K equals 8 instead of 5, undesired or "parasitic" edges appear in the graph. More precisely, a link connects the outer corner of the Swiss roll to a point lying in another whorl.

it can completely mislead the NLDR methods since they take it into account, just as they would with any other normal edge. For instance, such an edge can jeopardize the approximation of the geodesic distances. In that context, it can be compared to a shortcut in an electrical circuit: nothing works as desired. With $K = 8$, NLDR methods embed the Swiss roll, as shown in Figs. 6.15 and 6.16. As can be seen, the parasitic link completely misleads Isomap and GNLM. Obviously, CDA yields a result very similar to the one displayed in Fig. 6.4, i.e., in the ideal case, without undesired links. The good performance of CDA is due to the parameterized weighting function F_λ, which allows CDA to tear some parts of the manifold. In this case, the parasitic link has been torn. However, this nice result can require some tuning of the neighborhood proportion π in CDA. Depending on the parameter value, the tear does not always occur at the right place. A slight modification in the parameter setting may cause small imperfections: for example, the point on the corner may be torn off and pulled toward the other end of the parasitic link.

Results provided by SDE largely vary with respect to the paramater values. Specifically, imposing a strict preservation of local distances or allowing them to shrink leads to completely different embeddings. With strict equalities required, the presence of the parasitic edge prevents the semidefinite programming procedure included in SDE to unroll the manifold. Hence, the result looks like a slightly deformed PCA projection. With inequalities, SDE nearly succeeds in unfolding the Swiss roll; unfortunately, some regions of the manifold are superposed.

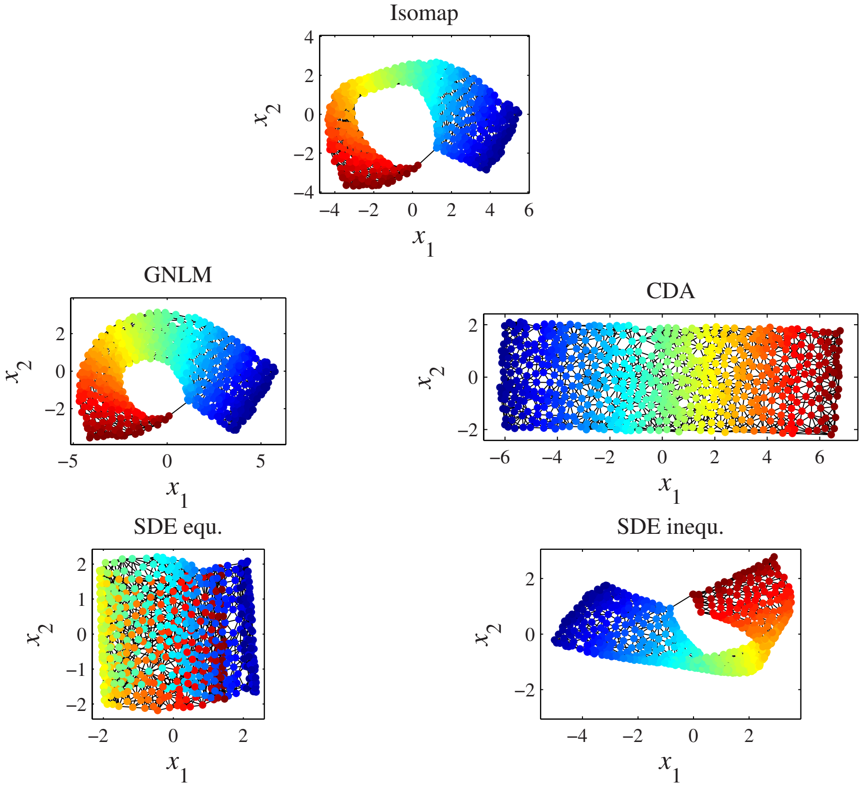


Fig. 6.15. Two-dimensional embeddings of the "Swiss roll" manifold by distance-preserving methods. All methods embed 600 prototypes obtained beforehand with vector quantization. In contrast with Fig. 6.4, the value of K in the K-rule is too high and an undesired edge appears in the graph connecting the prototypes, as shown in Fig. 6.14. As a consequence, graph distances fail to approximate the true geodesic distances.

Until now, topology-preserving methods using a data-driven lattice (LLE, LE, Isotop) seemed to outperform those working with a predefined lattice (SOM, GTM). Obviously, the former methods depend on the quality of the lattice. These methods cannot break the parasitic edge and fail to unfold the Swiss roll nicely. However, because they are not constrained to preserve distances strictly, even locally, those methods manage to distort the manifold in order to yield embeddings without superpositions. The SOM and GTM, of course, produce the same results as for the standard Swiss roll.

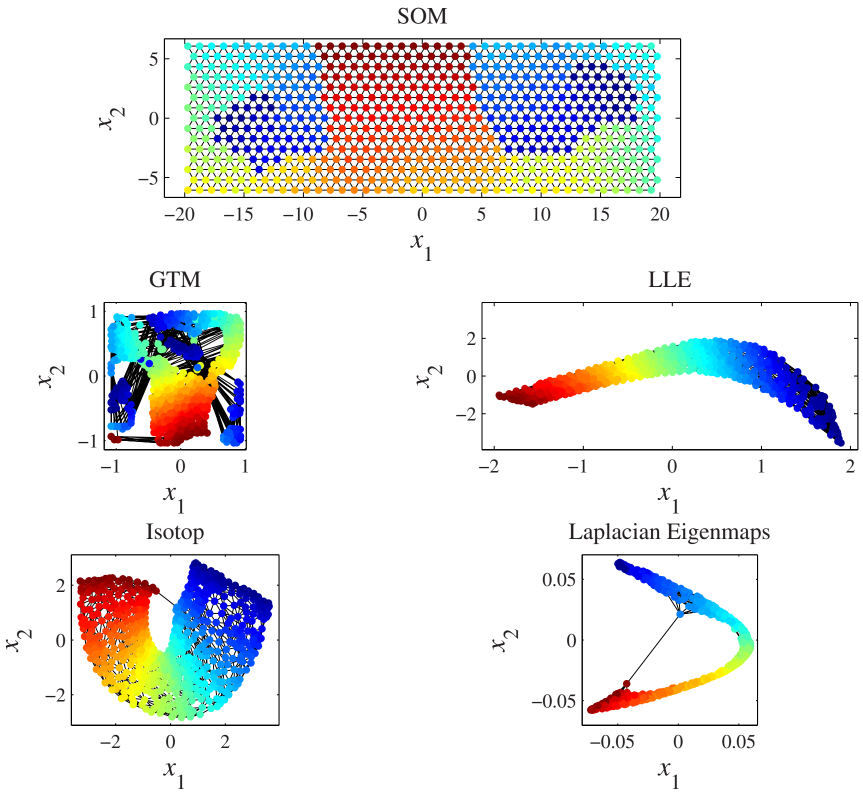


Fig. 6.16. Two-dimensional embeddings of the "Swiss roll" manifold by topology-preserving methods. All methods embed 600 prototypes obtained beforehand with vector quantization. The 5000 points are given as such for the SOM. In contrast with Fig. 6.4, the value of K in the K-rule is too high and an undesired edge appears in the graph connecting the prototypes, as shown in Fig. 6.14. As a consequence, the embedding quality decreases.

Nondevelopable Swiss roll

What happens when we attempt to embed a nondevelopable manifold? The answer may be given by the "heated" Swiss roll, whose parametric equations are

$$\mathbf{y} = \begin{bmatrix} (1+x_2^2)\sqrt{1+x_1}\cos(2\pi\sqrt{1+x_1}) \\ (1+x_2^2)\sqrt{1+x_1}\sin(2\pi\sqrt{1+x_1}) \\ 2x_2 \end{bmatrix} \quad , \tag{6.2}$$

where $\mathbf{x} = [x_1, x_2]^T$ is uniformly distributed in the interval $[-1,+1]^2$. The square root ensures that the distribution remains uniform in the 3D embedding. The first two coordinates, y_1 and y_2, depend on both latent variables x_1

and x_2 in a nonlinear way. Hence, conditions to have a developable manifold are broken. Figure 6.17 shows the resulting manifold, which looks like a Swiss roll that has melted in an oven. As for the normal Swiss roll, 5000 points are generated, but in this case at least 800 prototypes are needed, instead of 600, because consecutive whorls are closer to each other. The K-rule is used with $K = 5$. The results of NLDR methods are shown in Figs. 6.18 and 6.19. As expected, Isomap performs poorly with this nondevelopable manifold: points in the right part of the embedding are congregated. GNLM yields a better result than Isomap; unfortunately, the embedding is twisted. CDA does even better but needs some parameter tuning in order to balance the respective influences of short and long distances during the convergence. The neighborhood proportion must be set to a high value for CDA to behave as GNLM. More precisely, the standard schedule (hyperbolic decrease between 0.75 and 0.05) is replaced with a slower one (between 0.75 and 0.50) in order to avoid undesired tears. Results of SDE also depend on the parameter setting. With strict preservation of the distances, the embedding is twisted, whereas allowing the distances to shrink produces an eye-shaped embedding.

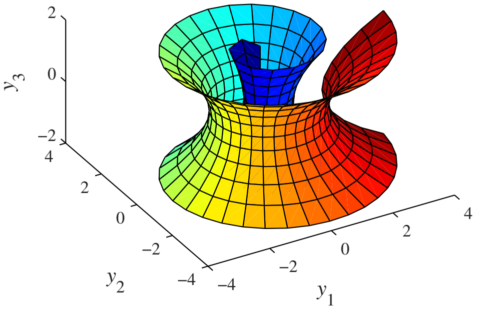


Fig. 6.17. The “heated” Swiss roll manifold.

Topology-preserving methods are expected to behave better with this nondevelopable manifold, since no constraint is imposed on distances. An SOM and GTM do not succeed in unfolding the heated Swiss roll. Spectral methods like LLE and LE fail, too. Only Isotop yields a nice embedding.

In the case of spectral methods, some flaws in the embeddings can be explained by looking to what happens in a third dimension, as was done for the standard Swiss roll. As spectral methods build embeddings incrementally, a third dimension is obtained by keeping an additional eigenvector of the Gram-like matrix. Figure 6.20 shows those three-dimensional embeddings. As can be seen, most embeddings are far from resembling the genuine latent space, namely a flat rectangle. Except for SDE with inequalities, the “span”

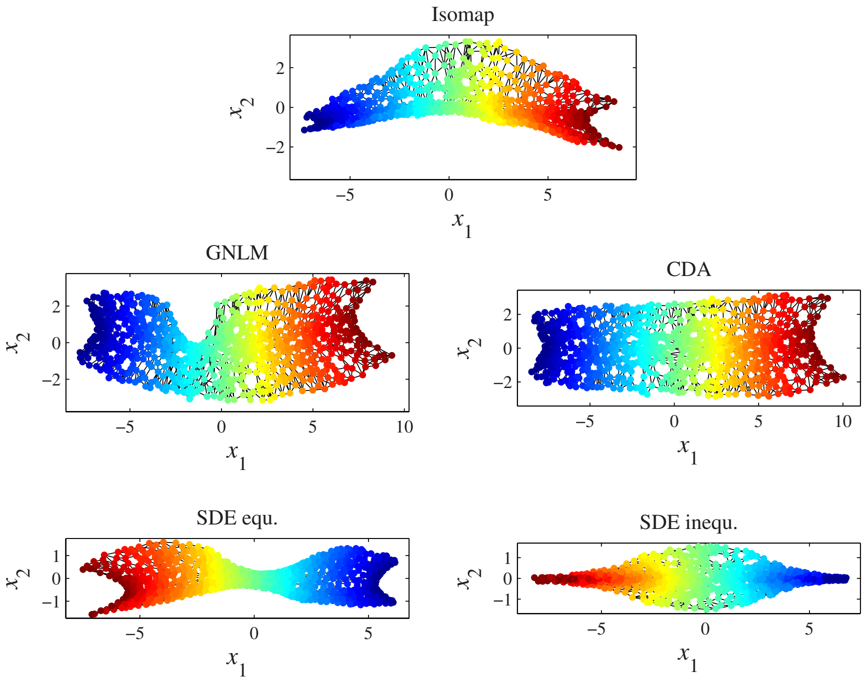


Fig. 6.18. Two-dimensional embeddings of the "heated" Swiss roll manifold computed by distance-preserving methods. All methods embed 800 prototypes obtained beforehand with vector quantization.

(or the variance) along the third dimension is approximately equal to the one along the second dimension. In the case of Isomap, the manifold remains folded lengthways. SDE with equalities leads to a twisted manifold, as does LLE. LE produces a helical embedding.

A last word about the Swiss roll

The bottom line of the above experiments consists of two conclusions. First, spectral methods offer nice theoretical properties (exact optimization, simplicity, possibility to build embeddings in an incremental way, etc.). However, they do not prove to be very robust against departures from their underlying model. For instance, Isomap does not produce satisfying embeddings for non-developable manifolds, which are not uncommon in real-life data. Second, iterative methods based on gradient descent, for example, can deal with more complex objective functions, whereas spectral methods are restricted to functions having "nice" algebraic properties. This makes the former methods more widely applicable. The price to pay, of course, is a heavier computational load and a larger number of parameters to be adjusted by the user.

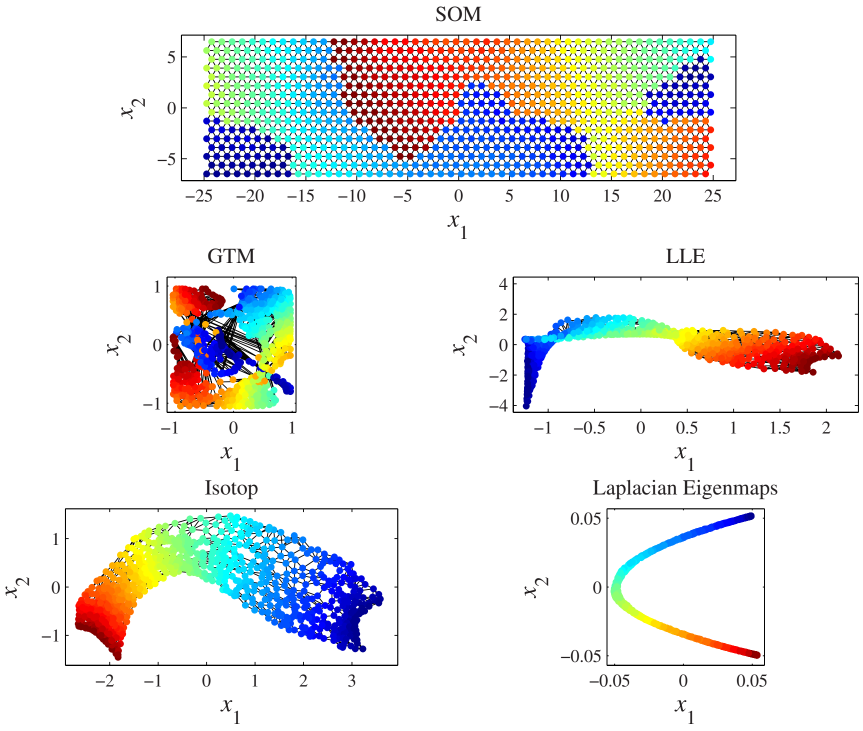
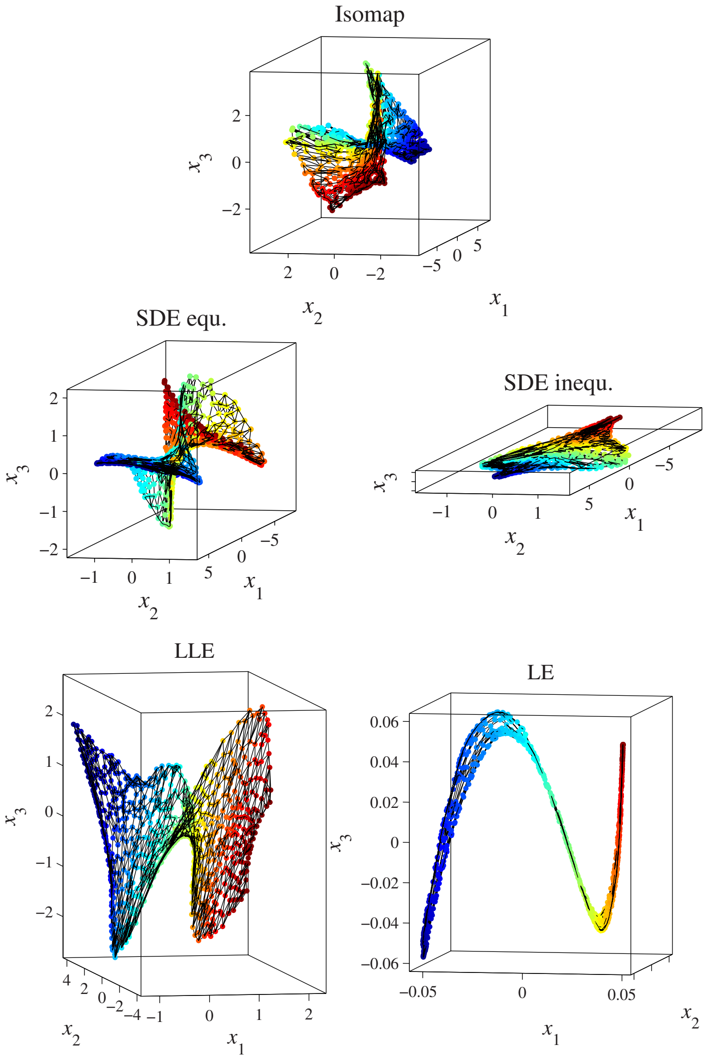


Fig. 6.19. Two-dimensional embeddings of the "heated" Swiss roll manifold computed by topology-preserving methods. All methods embed 800 prototypes obtained beforehand with vector quantization. The 5000 points are given as such for the SOM.

Most recent NLDR methods are based on the definition of neighborhoods, which induce a graph connecting the data points. This graph can be used for approximating either the geodesic distances or the manifold topology. Obviously, building such a graph requires the user to choose a neighborhood type (K-ary neighboroods, ϵ-balls, or still another rule). Of course, none of these rules is perfect and each of them involves parameters to adjust, thereby creating the possibility for the algorithm to fail in case of bad values. The presence of unwanted edges in the graph, for instance, can jeopardize the dimensionality reduction.

In recent distance-preserving NLDR methods, geodesic distances take a key part. This metric offers an elegant way to embed developable manifold. Unfortunately, in practice true geodesic distances cannot be computed; they are approximated by graph distances. In many cases, however, the approximations is far from perfection or can even fail. So, even if a manifold is developable in theory, it may happen that it cannot be considered as such in practice: a perfect isometry cannot be found. This can be due among other reasons to a


Isomap

SDE equ.

SDE inequ.

LLE

LE

Fig. 6.20. Three-dimensional embeddings of the "heated" Swiss roll manifold by Isomap, SDE, LLE, and LE.

data set that is too small or too noisy, or to an inappropriate parameter value. In the worst case, the graph used to compute the graph distances may fail to represent correctly the underlying manifold. This often comes from a wrong parameter value in the rule used to build the graph. Therefore, it is not prudent to rely solely on the graph distance when designing an NLDR method. Other techniques should be integrated in order to compensate for the flaws of the graph distance and make the method more flexible, i.e., more tolerant to data that do not fulfill all theoretical requirements. This is exactly what has been done in GNLM, CDA and other iterative methods. They use all-purpose optimization techniques that allow more freedom in the definition of the objective function. This makes them more robust and more polyvalent. As a counterpart, these methods are less elegant from the theoretical viewpoint and need some parameter tuning in order to fully exploit their capabilities.

Regarding topology preservation, methods using a data-driven lattice (LLE and Isotop) clearly outperform those relying on a predefined lattice. The advantage of the former methods lies in their ability to extract more information from data (essentially, the neighborhoods in the data manifold). Another explanation is that LLE and Isotop work in the low-dimensional embedding space, whereas an SOM and GTM iterate in the high-dimensional data space. In other words, LLE and Isotop attempt to embed the graph associated with the manifold, whereas an SOM and GTM try to deform and fit a lattice in the data space. The second solution offers too much freedom: because the lattice has more "Lebensraum" at its disposal in the high-dimensional space, it can jump from one part of a folded manifold to another. Working in the embedding space like LLE and Isotop do is more constraining but avoids these shortcuts and discontinuities.

6.1.2 Manifolds having essential loops or spheres

This subsection briefly describes how CCA/CDA can tear manifolds with essential loops (circles, knots, cylinders, tori, etc.) or spheres (spheres, ellipsoids, etc.). Three examples are given:

- The trefoil knot (Fig. 6.21) is a compact 1-manifold embedded in a three-dimensional space. The parametric equations are

  $$\mathbf{y} = \begin{bmatrix} 41\cos x - 18\sin x - 83\cos x - 83\sin(2x) - 11\sin(3x) + 27\sin(3x) \\ 36\cos x + 27\sin x - 113\cos(2x) + 30\sin(2x) + 11\cos(3x) - 27\sin(3x) \\ 45\sin x - 30\cos(2x) + 113\sin(2x) - 11\cos(3x) + 27\sin(3x) \end{bmatrix} ,$$

  where $0 \leq x < 2\pi$. The data set consists of 300 prototypes, which are obtained by vector quantization on 20,000 points randomly drawn in the knot. Neighboring prototypes are connected using the K-rule with $K = 2$.

- The sphere (Fig. 6.22) is a compact 2-manifold embedded in a three-dimensional space. For a unit radius, the parametric equations could be

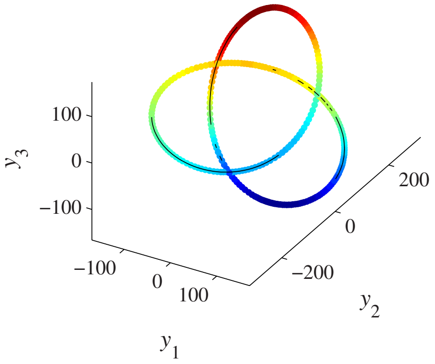


Fig. 6.21. The trefoil knot: a compact 1-manifold embedded in a three-dimensional space. The data set consists of 300 prototypes obtained after a vector quantization on 20,000 points of the knot. The color varies according to y_3. Prototypes are connected using the K-rule with $K = 2$.

$$\mathbf{y} = \begin{bmatrix} \cos(x_1)\cos(x_2) \\ \sin(x_1)\cos(x_2) \\ \sin(x_2) \end{bmatrix} ,$$

where $0 \leq x_1 < \pi$ and $0 \leq x_2 < 2\pi$. But unfortunately, with these parametric equations, a uniform distribution in the latent space yields a nonuniform and nonisotropic distribution on the sphere surface. Actually, points of the sphere are obtained by normalizing points randomly drawn in a three-dimensional Gaussian distribution (see Appendix B). The data set consists of 500 prototypes that are obtained by vector quantization on 20,000 points drawn randomly in the sphere. Neighboring prototypes are connected using the K-rule with $K = 5$.

- The torus (Fig. 6.23) is a compact 2-manifold embedded in a three-dimensional space. The parametric equations are

$$\mathbf{y} = \begin{bmatrix} (2 + \cos(x_1))\cos(x_2) \\ (2 + \cos(x_1))\sin(x_2) \\ \sin(x_1) \end{bmatrix} ,$$

where $0 \leq x_1 < 2\pi$ and $0 \leq x_2 < 2\pi$. With these parametric equations, the distribution on the torus surface is not uniform, but at least it is isotropic in the plane spanned by the coordinates y_1 and y_2. The data set consists of 1000 prototypes obtained by vector quantization on 20,000 points randomly drawn in the torus. Neighboring prototypes are connected using the K-rule with $K = 5$.

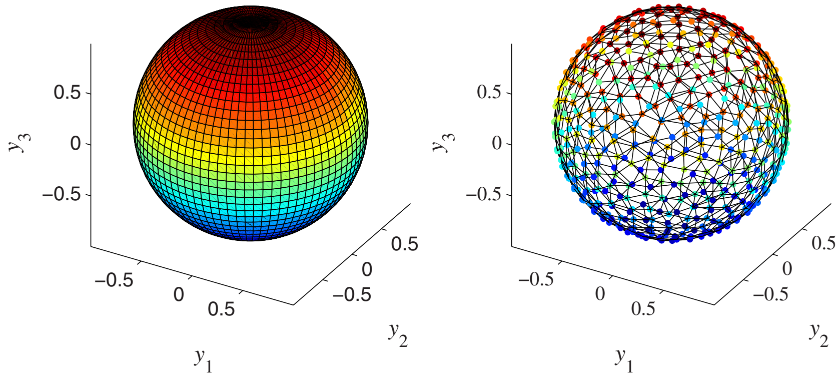


Fig. 6.22. The sphere: a compact 2-manifold embedded in a three-dimensional space. The manifold itself is displayed in the first plot, whereas the second plot shows the 500 prototypes used as the data set. They are obtained by a vector quantization on 20,000 points of the sphere. The color varies according to y_3 in both plots. Prototypes are connected using the K-rule with $K = 5$.

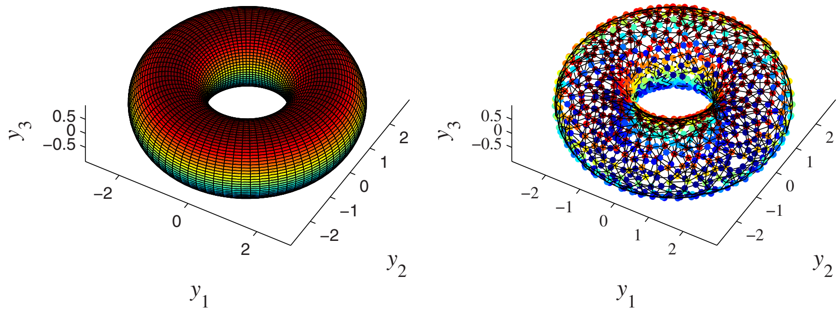


Fig. 6.23. The torus: a compact 2-manifold embedded in a three-dimensional space. The manifold itself is displayed in the first plot, whereas the second plot shows the 1000 prototypes used as the data set. They are obtained by a vector quantization on 20,000 points of the torus. The color varies according to y_3 in both plots. Prototypes are connected using the K-rule with $K = 5$.

For these three manifolds, it is interesting to see whether or not the graph distance helps the dimensionality reduction, knowing that the trefoil knot is the only developable manifold. Another question is: does the use of the graph distance change the way the manifolds are torn?

In the case of the trefoil knot, CCA and CDA can reduce the dimensionality from three to one, as illustrated in Fig. 6.24. In both one-dimensional plots, the graph edges are represented by blue corners (∨ or ∧). As can be seen, CCA tears the knot several times, although it is absolutely not needed. In contrast, CDA succeeds in unfolding the knot with a single tear only. The behavior of CCA and CDA is also different when the dimensionality is not reduced to its smallest possible value. Figure 6.25 illustrates the results of both methods when the knot is embedded in a two-dimensional space. This figure clearly explains why CDA yields a different embedding. Because the knot is highly folded and because CCA must preserve Euclidean distances, the embedding CCA computes attempts to reproduce the global shape of the knot (the three loops are still visible in the embedding). On the contrary, CDA gets no information about the shape of the knot since the graph distance is measured along the knot. As a result, the knot is topologically equivalent to a circle, which precisely corresponds to the embedding computed by CDA. Hence, in the case of the trefoil knot, the graph distance allows one to avoid unnecessary tears. This is confirmed by looking at Fig. 6.26, showing

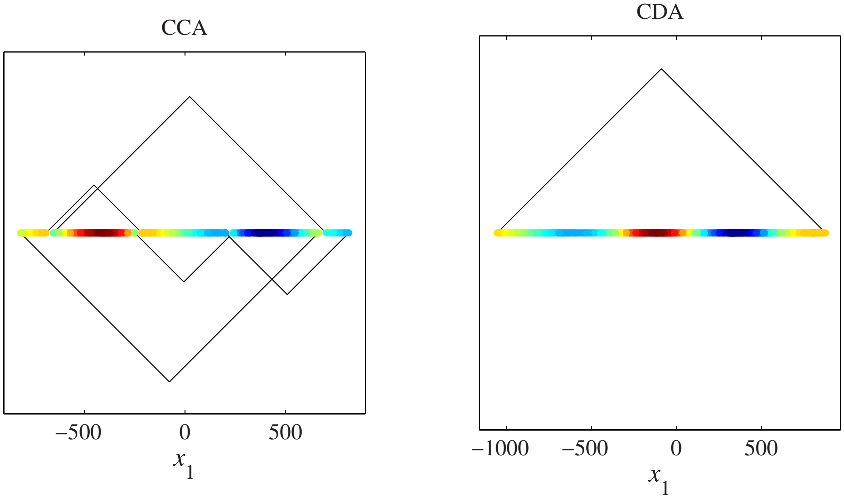


Fig. 6.24. One-dimensional embeddings of the trefoil knot by CCA and CDA. Graph edges between prototypes are displayed with blue corners (∨ or ∧). The color scale is copied from Fig. 6.21.

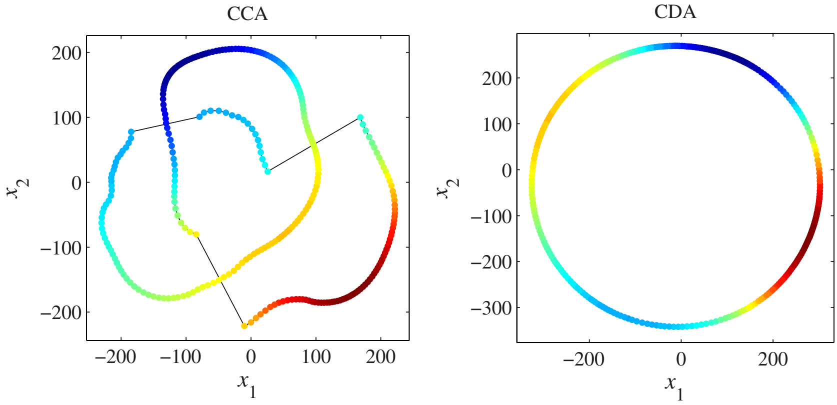


Fig. 6.25. Two-dimensional embeddings of the trefoil knot by CCA and CDA. The color scale is copied from Fig. 6.21.

two-dimensional embeddings computed by Sammon's NLM and its variant using the graph distance.

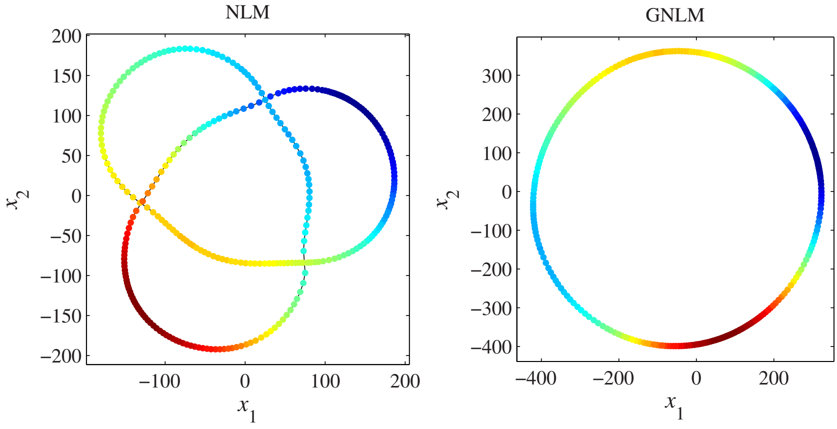


Fig. 6.26. Two-dimensional embeddings of the trefoil knot by NLM and GNLM. The color scale is copied from Fig. 6.21.

For the sphere and the torus, the graph distance seems to play a less important role, as illustrated by Figs. 6.27 and 6.28. Euclidean as well as graph distances cannot be perfectly preserved for these nondevelopable manifolds. The embeddings computed by both methods are similar. The sphere remains

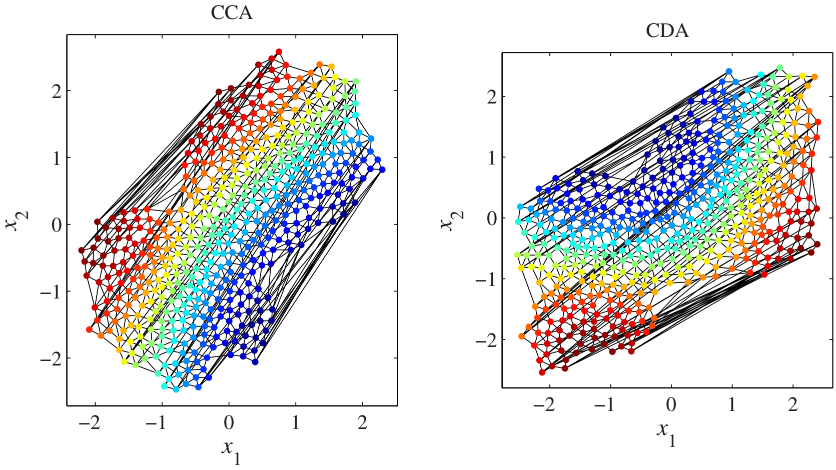


Fig. 6.27. Two-dimensional embeddings of the sphere by CCA and CDA. Colors remains the same as in Fig. 6.22.

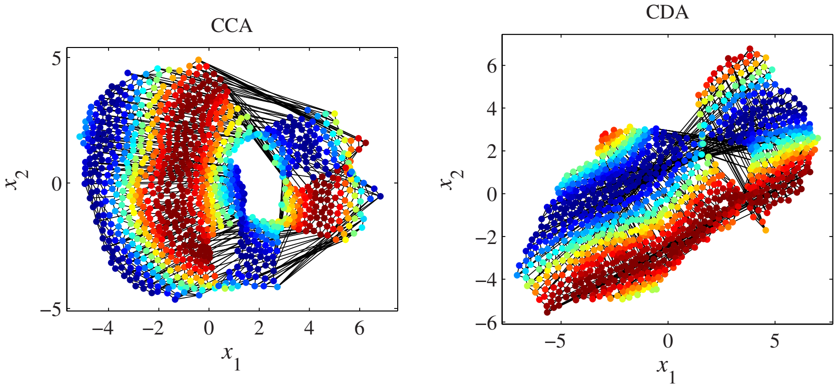


Fig. 6.28. Two-dimensional embeddings of the torus by CCA and CDA. The color scale is copied from Fig. 6.23.

rather easy to embed, but the torus requires some tuning of the parameters for both methods.

As already mentioned, CCA and CDA are the only methods having the intrinsic capabilities to tear manifolds. An SOM can also break some neighborhoods, when opposite edges of the map join each other on the manifold in the data space, for instance. It is noteworthy, however, that most recent NLDR methods working with local neighborhoods or a graph can be extended in order to tear manifolds with essential loops. This can be done by breaking some neighborhoods or edges in the graph before embedding the latter. A complete algorithm is described in [118, 121]. Unfortunately, this technique does not work for P-manifolds with essential spheres when $P > 1$ (loops on a sphere are always contractible and thus never essential).

6.2 Cortex unfolding

The study of the human brain has revealed that this complex organ is composed of roughly two tissues: the white matter and the gray matter. The former occupies a large volume in the center of the brain and contains mainly the axons of the neurons. On the other hand, the latter is only a thin (2–4-mm) shell, called the cortex and containing mainly the cell nuclei of the neurons [193]. In order to maximize the cortical surface without swelling the skull, evolution has folded it up, as illustrated by Fig. 6.29. Data in the latter figure come from the Unité mixte INSERM-UJF 594 (Université Joseph Fourier de Grenoble) and shows the cortical surface of a living patient [28]. How can it be obtained?

Actually, the shape of the cortical surface can be extracted noninvasively by techniques like magnetic resonance imaging (MRI) [192]. The principle of MRI is based on the specific way different tissues react to a varying magnetic field. According to the intensity of the reaction, a three-dimensional image of the tissues can be deduced. In the case of the brain, that image can be further processed in order to keep only the cortical surface. This can be done by segmenting the image, i.e., by delineating the subvolumes occupied by the different tissues [181]. Next, only voxels (3D pixels) corresponding to the cortical surface are kept and are encoded with three-dimensional coordinates.

Obviously, the analysis of the cortical surface as displayed in Fig. 6.29 appears difficult because of the many folds. It would be easier to have a flat, two-dimensional map of the cortex, in the same way as there are maps of the Earth's globe. DR techniques may help build such a representation. The two-dimensional embeddings computed by Sammon's NLM, CCA, GNLM, and CDA are shown in Figs. 6.30 to 6.33, respectively. Because the data set contains more than 5000 points, spectral methods using an EVD, like Isomap, SDE, and KPCA, are not considered. Usual software like MATLAB® failed to manage the huge amount of memory required by those methods. With

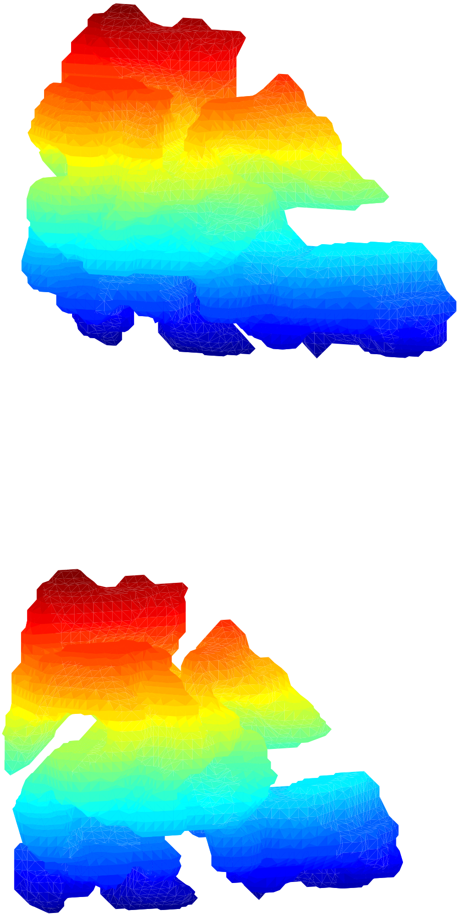


Fig. 6.29. Three-dimensional representations of a small piece of a human cortical surface, obtained by magnetic resonance imaging. Two different views of the same surface are shown, and the color varies according to the height.

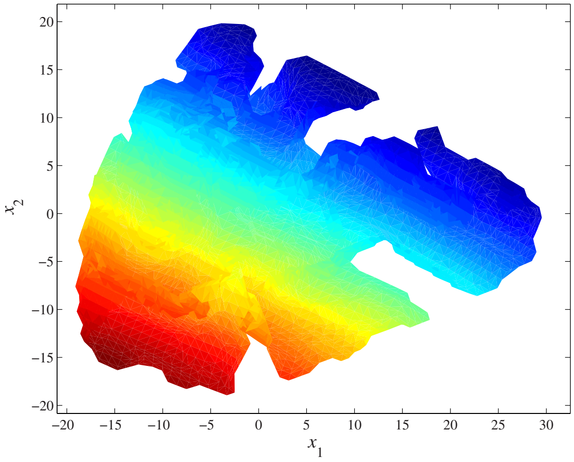


Fig. 6.30. Two-dimensional embedding of the cortical surface shown in Fig. 6.29, achieved by Sammon's NLM.

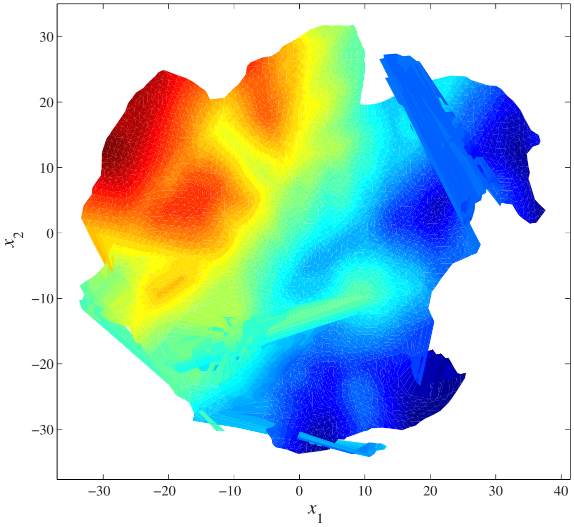


Fig. 6.31. Two-dimensional embedding of the cortical surface shown in Fig. 6.29, achieved by CCA.

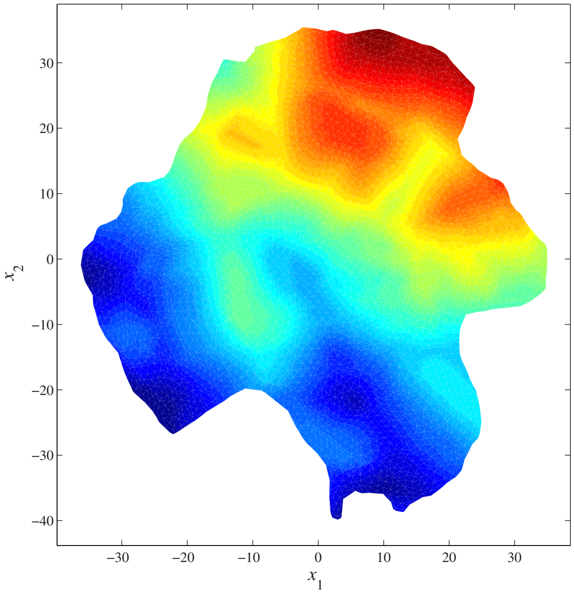


Fig. 6.32. Two-dimensional embedding of the cortical surface shown in Fig. 6.29, achieved by GNLM.

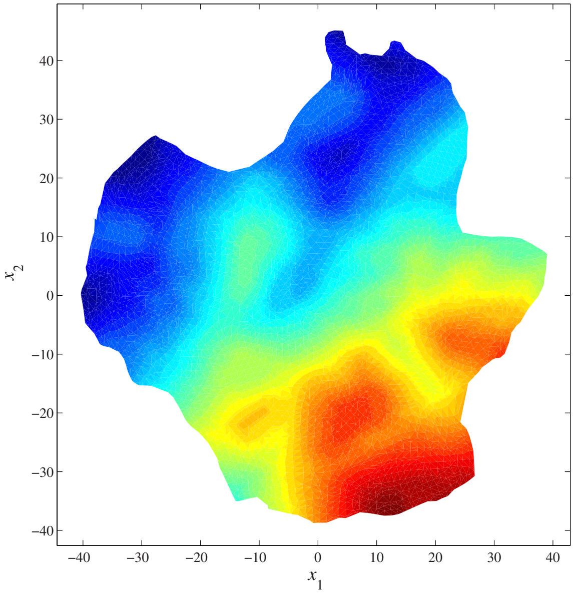


Fig. 6.33. Two-dimensional embedding of the cortical surface shown in Fig. 6.29, achieved by CDA.

of Isomap, we must be also remark that the cortical surface is clearly not a developable manifold.

As can be seen, the embedding computed by methods preserving Euclidean distances and the corresponding ones using graph distance do not yield the same results. Although the manifold is not developable, the graph distance greatly helps to unfold it. This is confirmed intuitively by looking at the final value of Sammon's stress. Using the Euclidean distance, NLM converges on $E_{\text{NLM}} = 0.0162$, whereas GNLM reaches a much lower and better value: $E_{\text{GNLM}} = 0.0038$.

Leaving aside the medical applications of these cortex maps, it is noteworthy that dimensionality reduction was also used to obtain a colored surface-like representation of the cortex data. The cortical surface is indeed not available directly as a surface, but rather as a set of points sampled from this surface. However, the only way to render a surface in a three-dimensional view consists of approximating it with a set of connected triangles. Unfortunately, most usual triangulation techniques that can convert points into coordinates work for two dimensions only. (Graph-building rules mentioned in Sections 4.3 and Appendix E are not able to provide an *exact* triangulation.) Consequently, dimensionality reduction provides an elegant solution to this problem: points are embedded in a two-dimensional space, triangulation is achieved using a standard technique in 2D, like Delauney's one, and the obtained triangles are finally sent back to the three-dimensional space. This allows us to display the three-dimensional views of the cortical surface in Fig. 6.29.

How do topology-preserving methods compare to them? Only results for an SOM (40-by-40 map), GTM (20-by-20 latent grid, 3×3 kernels, 100 EM iterations), and Isotop (without vector quantization, ϵ-rule, $\epsilon = 1.1$) are given. Due to its huge memory requirements, LLE failed to embed the 4961 points describing the cortical surface. The result of the SOM is illustrated by Fig. 6.34. The embedding computed by GTM is shown in Fig. 6.35. Figure 6.36 displays Isotop's result. Again, it must be stressed that methods using a predefined lattice— even if they manage to preserve the topology —often distort the shape of data. This is particularly true for the SOM and questionable for GTM (the latent space can be divided into several regions separated by empty or sparsely populated frontiers). Isotop yields the most satisfying result; indeed, the topology is perfectly preserved and, in addition, so is the shape of data: the embedding looks very close to the results of distance-preserving methods (see, e.g., Fig. 6.33). The main difference between Figs. 6.33 and 6.36 lies in the axes: for Isotop, the size of the embedding is not related to the pairwise distances measured in the data set.

6.3 Image processing

As already mentioned in Subsection 1.1.1), dimensionality reduction can be applied to image processing. Within this framework, each pixel of an image

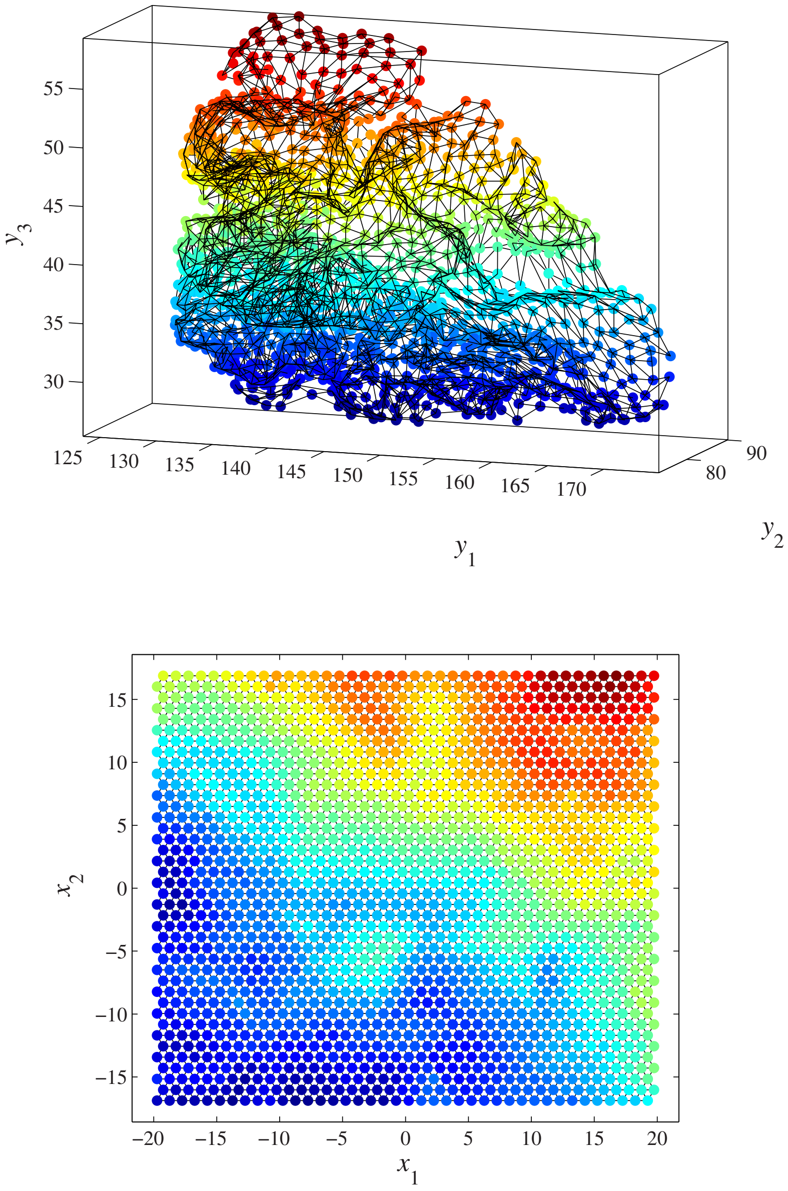


Fig. 6.34. Two-dimensional embedding of the cortical surface shown in Fig. 6.29, achieved by an SOM. The first plot shows how the SOM unfurls in the three-dimensional space, whereas the second represents the two-dimensional SOM lattice.

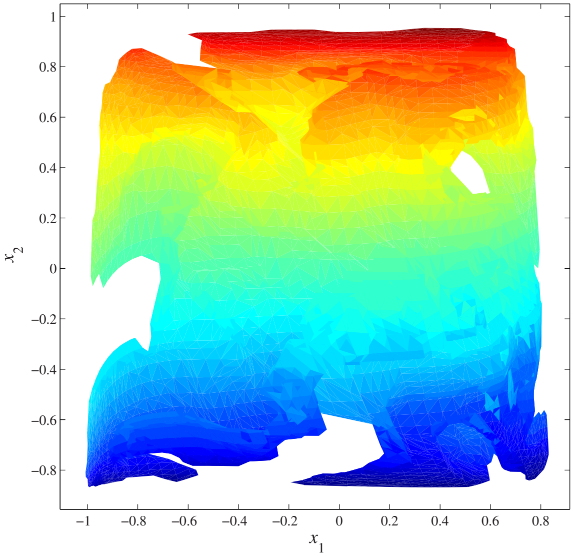


Fig. 6.35. Two-dimensional embedding of the cortical surface shown in Fig. 6.29, achieved by GTM.

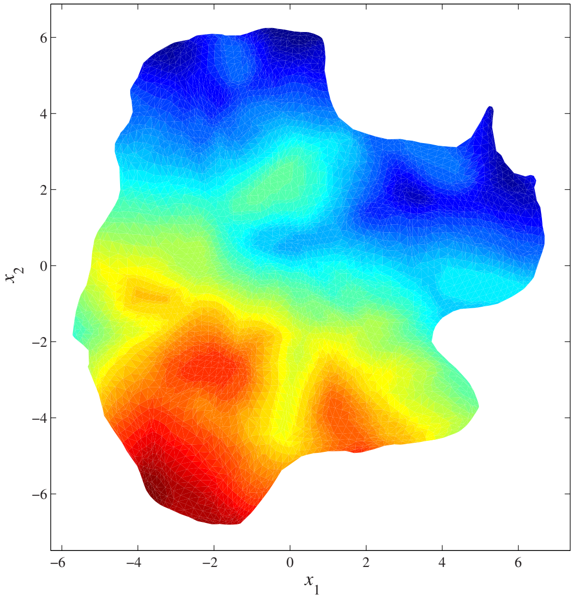


Fig. 6.36. Two-dimensional embedding of the cortical surface shown in Fig. 6.29, achieved by Isotop.

can be considered to be a dimension or observed variable in a very high-dimensional space, namely the whole image. As an example, a small picture, of size 64 by 64 pixels, corresponds to a vector in a 4096-dimensional space. However, in a typical image, it is very likely that neighboring pixels depend on each other; an image is often split into regions having almost the same color. Obviously, the relationships between pixels depend on the particular type of image analyzed. For a given set of images, it can be reasonably stated that the intrinsic dimensionality largely depends on the depicted content, and not on the number of pixels. For instance, if the set contains similar images (several landscapes or several portraits, etc.) or if the same object is depicted in several positions, orientations, or illuminations, then the intrinsic dimensionality can be quite low. In this case, dimensionality reduction can be very useful to "sort" the image set, i.e., to obtain a representation in which similar images are placed close to each other.

6.3.1 Artificial faces

A set of 698 face images is proposed in [180]. The images represent an artificially generated face rendered with different poses and lighting directions. Figure 6.37 shows several faces drawn at random in the set. Each image consists of an array of 64 by 64 pixels, that are associated with a single brightness value. Before dimensionality reduction, each image is converted into a 4096-dimensional vector. As the latter number is very high, an initial dimensionality reduction is performed by PCA, in a purely linear way. The first 240 principal components are kept; they bear more than 99% of the global variance. In other words, PCA achieves an almost lossless dimensionality reduction in this case.

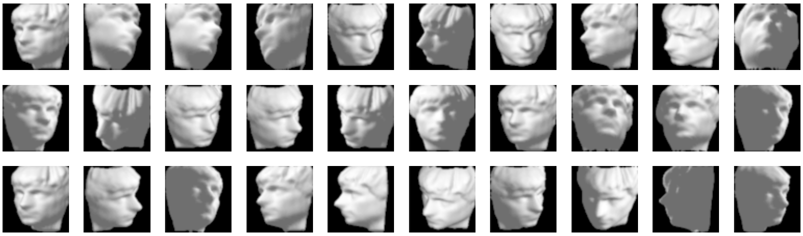


Fig. 6.37. Several face pictures drawn at random from the set of 698 images proposed in [180].

Next, the 240-dimensional vectors are processed by the following distance-preserving methods: metric MDS, Sammon's NLM, CCA, Isomap, GNLM, CDA, and SDE (with equality and inequality constraints). Three topology-preserving methods are also used (LLE, LE, and Isotop); methods using a predefined lattice like an SOM or GTM are often restricted to two-dimensional latent spaces.

Metric MDS is the only linear method. All methods involving neighborhoods, a graph, or graph distances use the K-rule, with $K = 4$ (this value is lower than the one proposed in [180] but still produces a connected graph). No vector quantization is used since the number of images ($N = 698$) is low.

At this point, it remains to decide the dimensionality of the embedding. Actually, it is known that the images have been generated using three degrees of freedom. The first latent variable is the left-right pose, the second one is the up-down pose, and the third one is the lighting direction (left or right). Hence, from a theoretical point of view, the 240-dimensional vectors lie on a three-manifold. From a practical point of view, are the eight distance-preserving methods able to reduce the dimensionality to three?

Figures 6.38–6.48 answer this question in a visual way. Each of these figures represents the three-dimensional embedding computed by one of the eight methods. In order to ease the visualization, the embeddings are processed as follows:

1. Each embedding is rotated in such a way that its three coordinates actually correspond to its three principal components.
2. A rectangular parallelepiped is centered around the embedding.
3. The parallelepiped is divided into 6-by-6-by-6 cells.
4. The six layers along the third coordinate are shown separately for each embedding.

Each layer consists of 36 cells containing some points. Instead of displaying them as simple dots— which does not convey much information —the image associated with one of the points in each cell is displayed. This point is chosen to be the closest one from the average of all points in the cell. Empty cells are left blank.

With such a representation, the embedding quality can be assessed visually by looking at neighboring cells. If images smoothly change from one cell to another, then local distances have been correctly reproduced.

As it can be seen in Fig. 6.38, metric MDS does not perform very well. This demonstrates that a linear method does not suffice to embed data. This is confirmed by looking at the eigenvalues: coordinates along the three principal components carry barely 60% of the global variance measured on the 240-dimensional vectors.

The result of Sammon's NLM, illustrated by Fig. 6.39, seems visually more pleasing. Layers are more regularly populated, and clusters of similar images can be perceived.

By comparison, the embedding computed by CCA in Fig. 6.40 is disappointing. There are discontinuities in layers 3, 4, and 5: neighboring faces are sometimes completely different from each other.

The embedding provided by Isomap is good, as visually confirmed by Fig. 6.41. However, layers are sparsely populated; the left-right and up-down

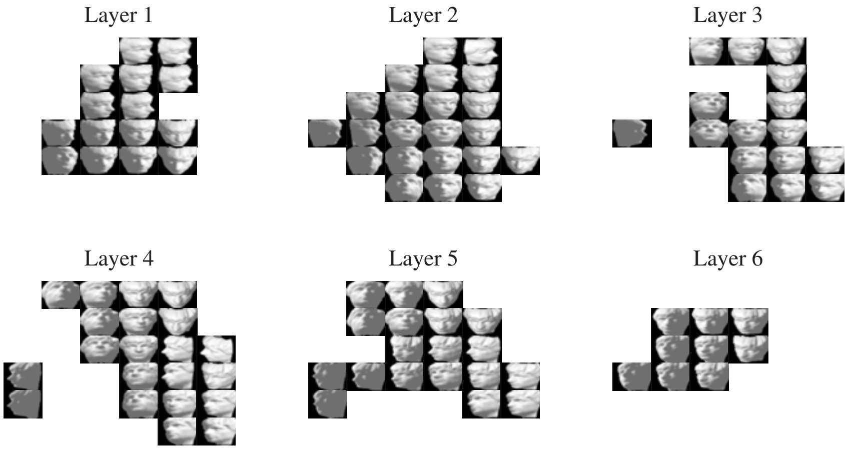


Fig. 6.38. Three-dimensional embedding of the 698 face images computed by metric MDS. The embedding is sliced into six layers, which in turn are divided into 6 by 6 cells. Each cell is represented by displaying the image corresponding to one of the points it contains. See text for details.

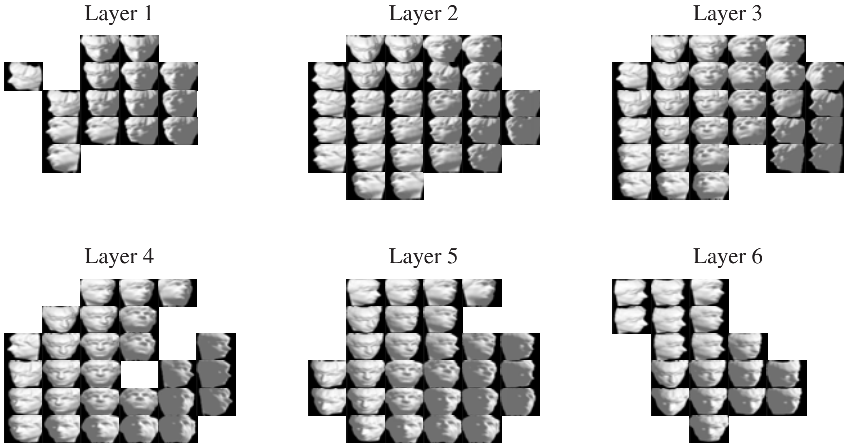


Fig. 6.39. Three-dimensional embedding of the 698 face images computed by Sammon's NLM. The embedding is sliced into six layers, which in turn are divided into 6 by 6 cells. Each cell is represented by displaying the image corresponding to one of the points it contains. See text for details.

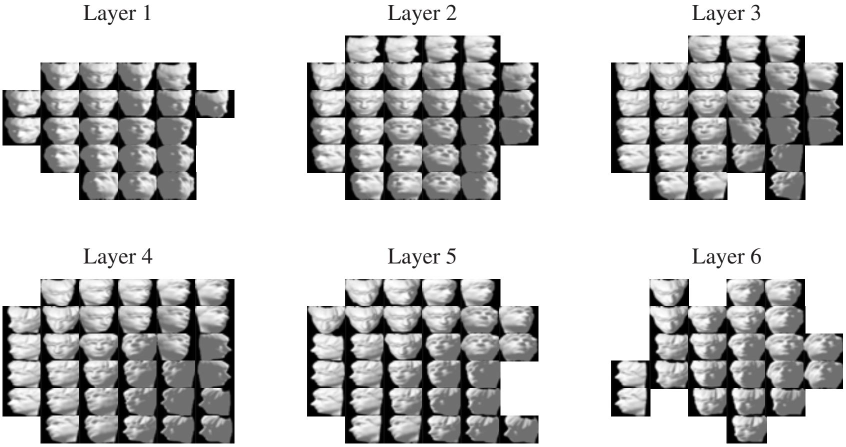


Fig. 6.40. Three-dimensional embedding of the 698 face images computed by CCA. The embedding is sliced into six layers, which in turn are divided into 6 by 6 cells. Each cell is represented by displaying the image corresponding to one of the points it contains. See text for details.

poses vary smoothly accross the layers. The lighting direction can be perceived too: the light source is on the left (resp., right) of the face in the first (resp., last) layers. Can this outstanding performance be explained by the use of the graph distance?

When looking at Fig. 6.42, the answer seems to be yes, since GNLM performs much better than NLM, and does as well as Isomap. The final confirmation comes with the good result of CDA (Fig. 6.43). All layers are quite densely populated, including the first and last ones. As for Isomap, the head smoothly moves from one picture to another. The changes of lighting direction are clearly visible, too.

In addition, two versions of SDE (with equality constraints or with local distances allowed to shrink) work very well. Layers are more densely populated with strict equality constraints, however.

Results of topology-preserving methods are given in Figs. 6.46–6.48. LLE provides a disappointing embedding, though several values for its parameters were tried. Layers are sparsely populated, and transitions between pictures are not so smooth. The result of LE is better, but still far from most distance-preserving methods. Finally, Isotop succeeds in providing a good embedding, though some discontinuities can be observed, for example, in layer 2.

In order to verify the visual impression left by Figs. 6.38–6.48, a quantitative criterion can be used to assess the embeddings computed by the NLDR methods. Based on ideas developped in [9, 74, 10, 190, 106, 103], a simple criterion can be based on the proximity rank. The latter can be denoted as the function $r = \text{rank}(\mathbf{X}, i, j)$ and is computed as follows:

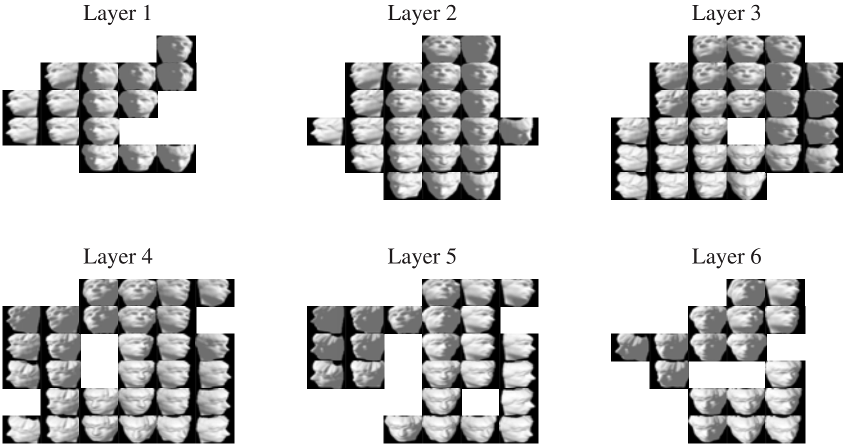


Fig. 6.41. Three-dimensional embedding of the 698 face images computed by Isomap. The embedding is sliced into six layers, which in turn are divided into 6 by 6 cells. Each cell is represented by displaying the image corresponding to one of the points it contains. See text for details.

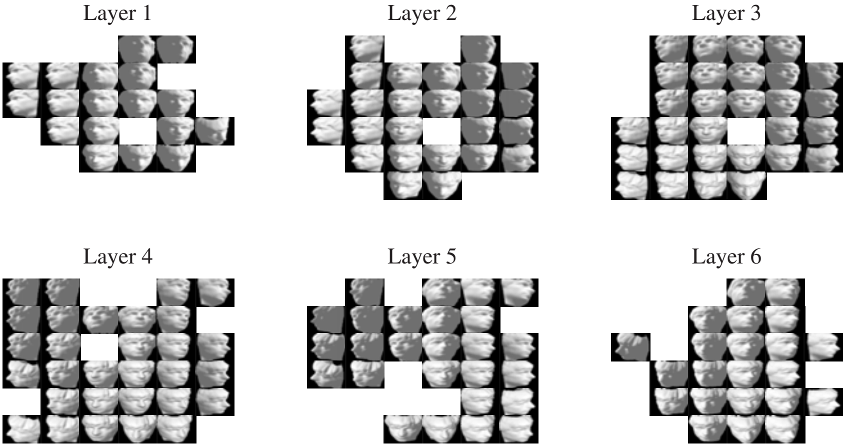


Fig. 6.42. Three-dimensional embedding of the 698 face images computed by GNLM. The embedding is sliced into six layers, which in turn are divided into 6 by 6 cells. Each cell is represented by displaying the image corresponding to one of the points it contains. See text for details.

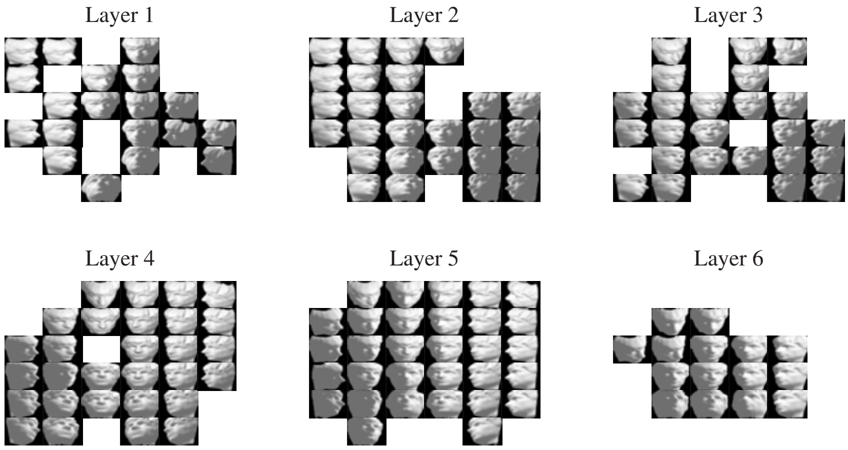


Fig. 6.43. Three-dimensional embedding of the 698 face images computed by CDA. The embedding is sliced into six layers, which in turn are divided into 6 by 6 cells. Each cell is represented by displaying the image corresponding to one of the points it contains. See text for details.

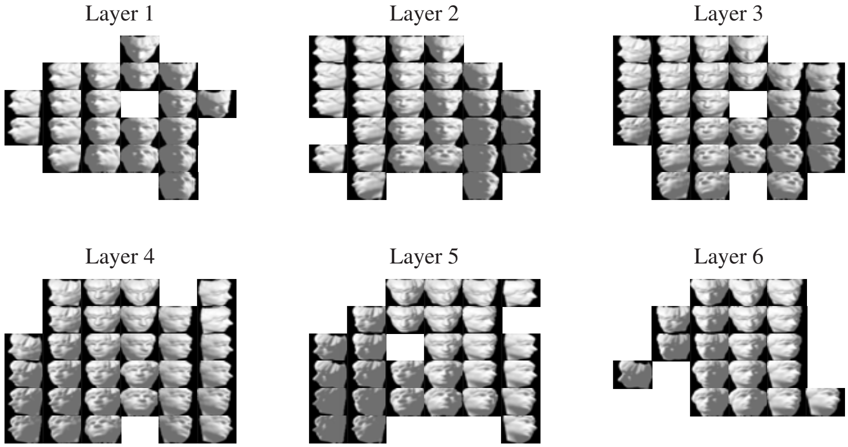


Fig. 6.44. Three-dimensional embedding of the 698 face images computed by SDE (with equality constraints). The embedding is sliced into six layers, which in turn are divided into 6 by 6 cells. Each cell is represented by displaying the image corresponding to one of the points it contains. See text for details.

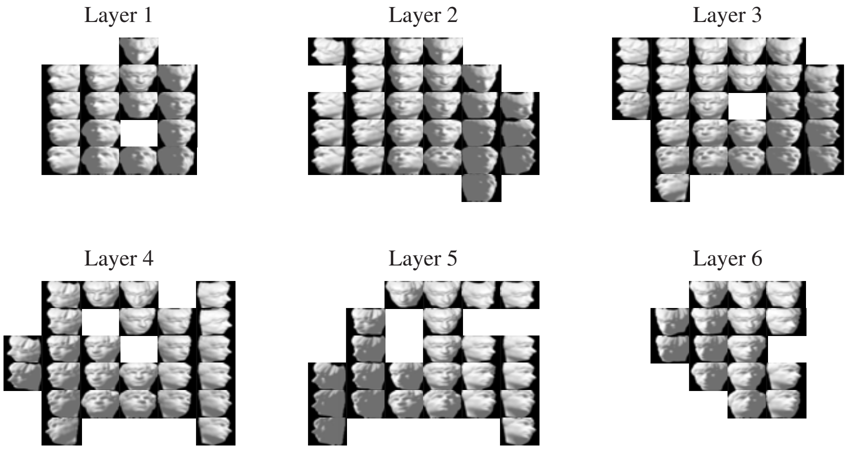


Fig. 6.45. Three-dimensional embedding of the 698 face images computed by SDE (with inequality constraints). The embedding is sliced into six layers, which in turn are divided into 6 by 6 cells. Each cell is represented by displaying the image corresponding to one of the points it contains. See text for details.

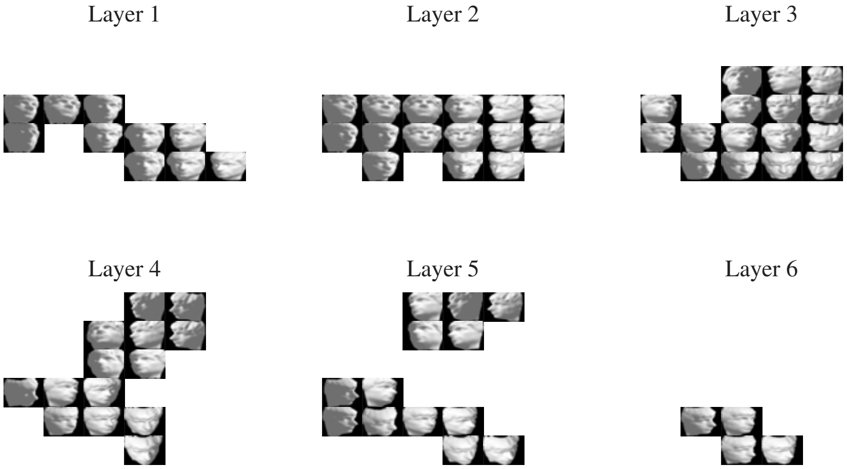


Fig. 6.46. Three-dimensional embedding of the 698 face images computed by LLE. The embedding is sliced into six layers, which in turn are divided into 6 by 6 cells. Each cell is represented by displaying the image corresponding to one of the points it contains. See text for details.

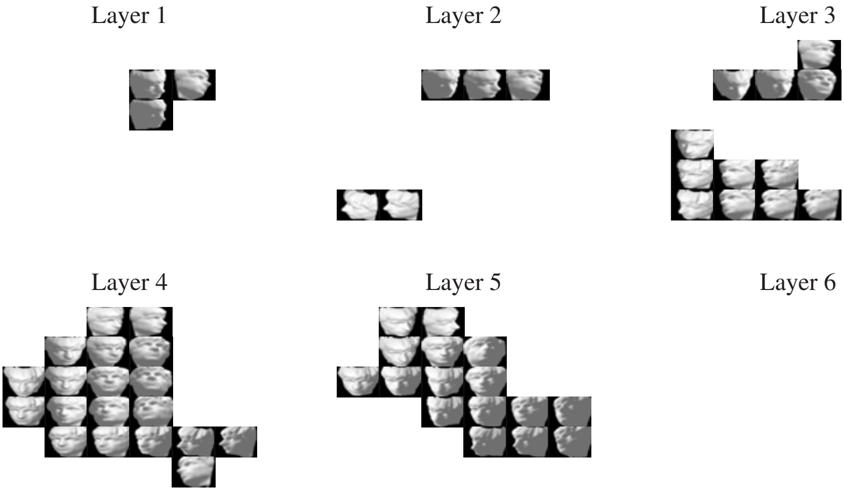


Fig. 6.47. Three-dimensional embedding of the 698 face images computed by LE. The embedding is sliced into six layers, which in turn are divided into 6 by 6 cells. Each cell is represented by displaying the image corresponding to one of the points it contains. See text for details.

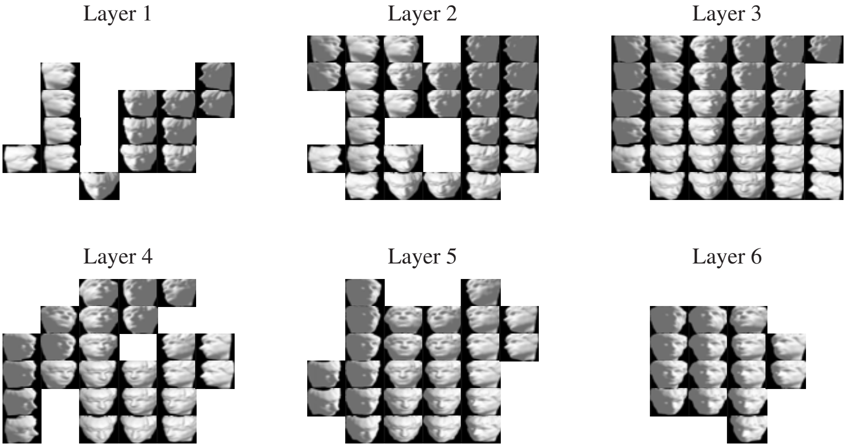


Fig. 6.48. Three-dimensional embedding of the 698 face images computed by Isotop. The embedding is sliced into six layers, which in turn are divided into 6 by 6 cells. Each cell is represented by displaying the image corresponding to one of the points it contains. See text for details.

- Using the vector set $\mathbf{X}$ and taking the ith vector as reference, compute all Euclidean distances $\|\mathbf{x}(k) - \mathbf{x}(i)\|$, for $1 \leq k \leq N$.
- Sort the obtained distances in ascending order, and let output r be the rank of $\mathbf{x}(j)$ according to the sorted distances.

In the same way as in [185, 186], this allows us to write two different measures, called mean relative rank errors:

$$\mathrm{MRRE}_{\mathbf{Y}\rightarrow\mathbf{X}}(K) \triangleq \frac{1}{C}\sum_{i=1}^{N}\sum_{j\in\mathcal{N}_K(\mathbf{y}(i))}\frac{|\mathrm{rank}(\mathbf{X},i,j)-\mathrm{rank}(\mathbf{Y},i,j)|}{\mathrm{rank}(\mathbf{Y},i,j)} \tag{6.3}$$

$$\mathrm{MRRE}_{\mathbf{X}\rightarrow\mathbf{Y}}(K) \triangleq \frac{1}{C}\sum_{i=1}^{N}\sum_{j\in\mathcal{N}_K(\mathbf{x}(i))}\frac{|\mathrm{rank}(\mathbf{X},i,j)-\mathrm{rank}(\mathbf{Y},i,j)|}{\mathrm{rank}(\mathbf{X},i,j)} \quad , \tag{6.4}$$

where $\mathcal{N}_K(\mathbf{x}(i))$ denotes the K-ary neighborhood of $\mathbf{x}(i)$. The normalization factor is given by

$$C = N\sum_{k=1}^{K}\frac{|2k-N-1|}{k} \tag{6.5}$$

and scales the error between 0 and 1. Quite obviously, $\mathrm{MRRE}_{\mathbf{Y}\rightarrow\mathbf{X}}(K)$ and $\mathrm{MRRE}_{\mathbf{X}\rightarrow\mathbf{Y}}(K)$ both vanish if the K closest neighbors of each datum appear in the same order in both spaces. The first error, $\mathrm{MRRE}_{\mathbf{X}\rightarrow\mathbf{Y}}(K)$, can be compared to the continuity measure, whereas $\mathrm{MRRE}_{\mathbf{Y}\rightarrow\mathbf{X}}(K)$, is similar to the trustworthiness [185, 186].

Figure 6.49 shows the evolution of the two errors as a function of K for all NLDR methods. Looking at the curves of $\mathrm{MRRE}_{\mathbf{Y}\rightarrow\mathbf{X}}(K)$ on the left, it can be seen that LLE yields the worst results, though its parameters were carefully tuned ($K = 6$ and $\Delta = 0.01$). Metric MDS performs rather poorly; this is not unexpected since it is the only purely linear method. Better results were obtained by SDE (with strict equality constraints), CDA, and GNLM. Isomap and SDE with inequalities achieve comparable results. Methods preserving geodesic or graph distances outperform those working with Euclidean distances. Isotop achieves very good results, too. It is also noteworthy that there is a discontinuity in the slope of $\mathrm{MRRE}_{\mathbf{Y}\rightarrow\mathbf{X}}(K)$ at $K = 4$ for methods using a graph or K-ary neighborhoods; this precisely corresponds to the value of K in the K-rule (except for LLE).

Looking at the curves of $\mathrm{MRRE}_{\mathbf{Y}\rightarrow\mathbf{X}}(K)$, on the right, LLE is again the worst method. On the other hand, CDA performs the best, followed by GNLM and SDE with strict equality constraints. Again, all geodesic methods outperform their corresponding Euclidean equivalent. Regarding topology preservation, the best embedding is provided by Isotop.

6.3.2 Real faces

The previous section shows an application of NLDR methods to the visualization of a set of articially generated face pictures. In this section, the same

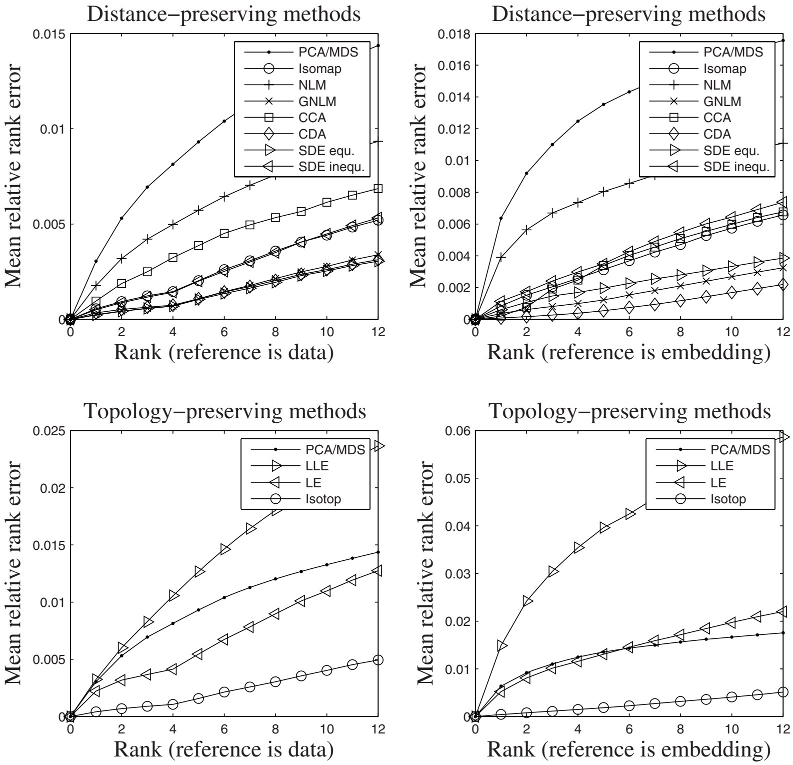


Fig. 6.49. Mean relative rank errors ($\mathrm{MRRE}_{\mathbf{Y}\rightarrow\mathbf{X}}(K)$ on the left and $\mathrm{MRRE}_{\mathbf{X}\rightarrow\mathbf{Y}}(K)$ on the right) of all NLDR methods for the artificial faces.

problem is studied, but with real images. As a direct consequence, the intrinsic dimensionality of the images is not known in advance. Instead of computing it with the methods described in Chapter 3, it is proposed to embed the data set in a two-dimensional plane by means of NLDR methods. In other words, even if the intrinsic dimensionality is probably higher than two, a two-dimensional representation is "forced".

In practice, the data set [158] comprises 1965 images in levels of gray; they are 28 pixels high and 20 pixels wide. Figure 6.50 shows a randomly drawn subset of the images. The data are rearranged as a set of 1965 560-dimensional vectors, which are given as they are, without any preprocessing. (In the previous section, images were larger and were thus preprocessed with PCA.) Six distance-preserving methods are used: metric MDS, Isomap, NLM, GNLM, CCA, and CDA. SDE could not be used due to its huge computational requirements. Four topology-preserving methods are compared too: a 44-by-44 SOM, LLE, LE, and Isotop. GTM is discarded because of the high dimensionality of the data space. All methods involving K-ary neighborhoods

Fig. 6.50. Some faces randomly drawn from the set of real faces available on the LLE website.

or a graph work with $K = 4$, except LLE ($K = 12$, $\Delta = 0.0001$ as advised in [158]). The embedding computed by each method is displayed by means of thumbnail pictures, in a similar way as in the previous section. Actually, the region occupied by the embedding is divided into 22-by-22 cells. Each of them is represented by a "mean" image, which is the average of all images corresponding to the two-dimensional points lying in the considered cell. A graph connecting each data point with its four closest neighbors is shown alongside (except for the SOM, for which the hexagonal grid is represented).

The two-dimensional embedding computed by metric MDS is shown in Fig. 6.51. As can be seen, the embedding has a C shape. In the lower (resp., upper) branch, the boy looks on his left (resp., right). In the left part of the embedding, the boy seems unhappy; he is smiling in the right part. The

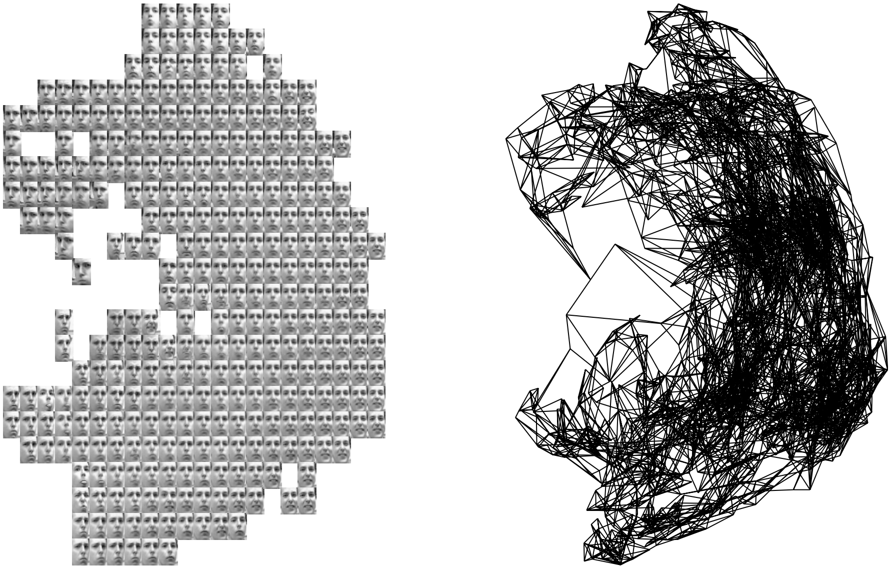


Fig. 6.51. Two-dimensional embedding of the 1965 real faces by metric MDS.

embedding resulting from Isomap, that is, metric MDS with graph distances, is shown in Fig. 6.52. More details appear, and fewer cells are blank, but a Swiss-cheese effect appears: holes in the graph look stretched. Moreover, images corresponding to some regions of the embeddings look blurred or fuzzy. This is due to the fact that regions in between or on the border of the holes are shrunk: too many different images are concentrated in a single cell and are averaged together.

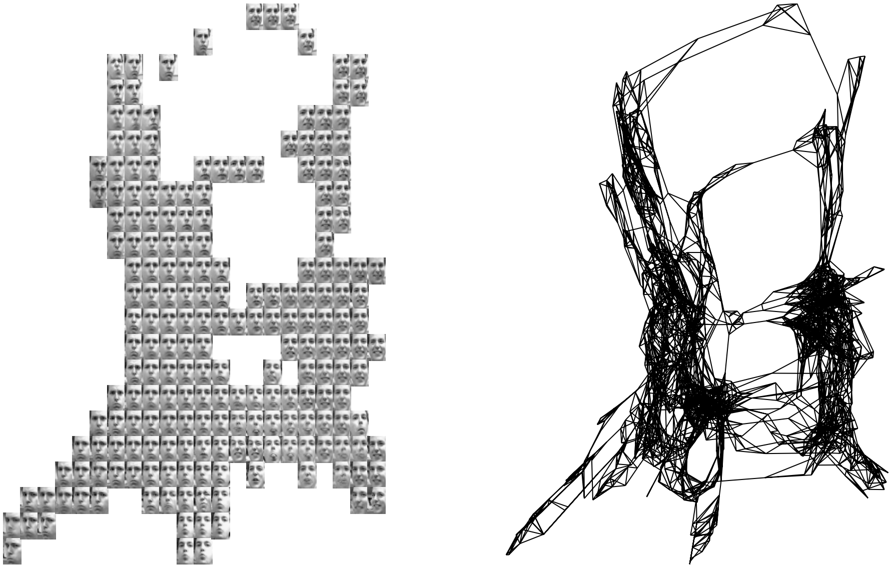


Fig. 6.52. Two-dimensional embedding of the 1965 real faces by Isomap.

The embedding provided by NLM (Fig. 6.53) confirms that the data set is roughly composed of two dense, weakly connected clusters. No Swiss-cheese effect can be observed. Unfortunately, there are still blurred regions and some discontinuities are visible in the embedding. The use of graph distances in Sammon's nonlinear mapping (Fig. 6.54) leads to a sparser embedding. Nevertheless, many different facial expressions appear. Holes in the graph are not stretched as they are for Isomap.

CCA provides a dense embedding, as shown in Fig. 6.55. Few discontinuities can be found. Replacing the Euclidean distance with the graph distance leads to a sparser embedding, just like for NLM and GNLM. This time, several smaller clusters can be distinguished. Due to sparsity, some parts of the main cluster on the right are somewhat blurred.

Figure 6.57 shows the embedding computed by LLE. As usual, LLE yields a cuneiform embedding. Although the way to display the embedding is different in [158], the global shape looks very similar. For the considered data set, the

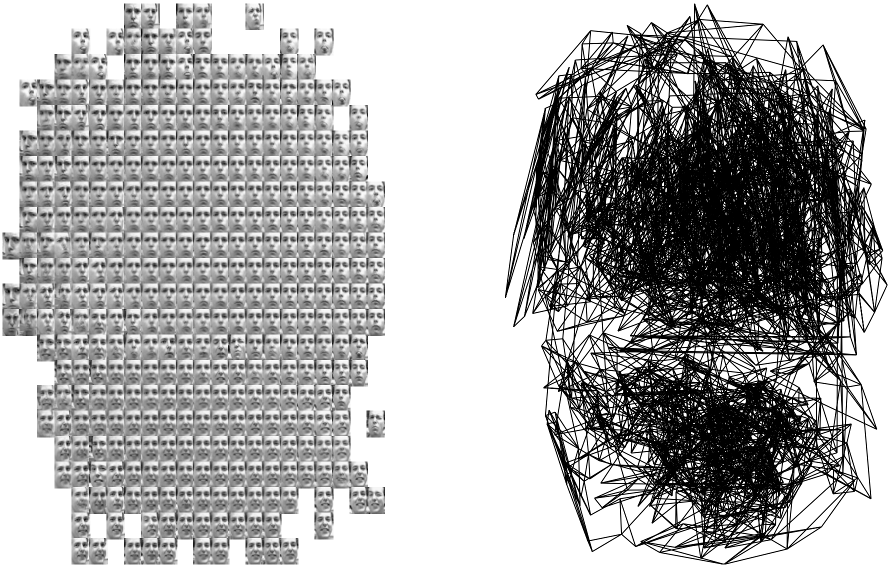


Fig. 6.53. Two-dimensional embedding of the 1965 real faces by NLM.

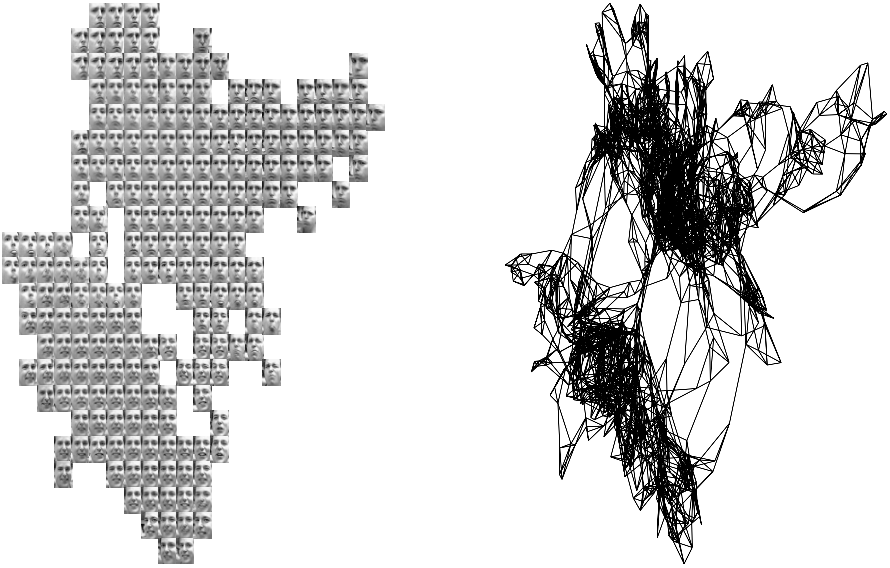


Fig. 6.54. Two-dimensional embedding of the 1965 real faces by GNLM.

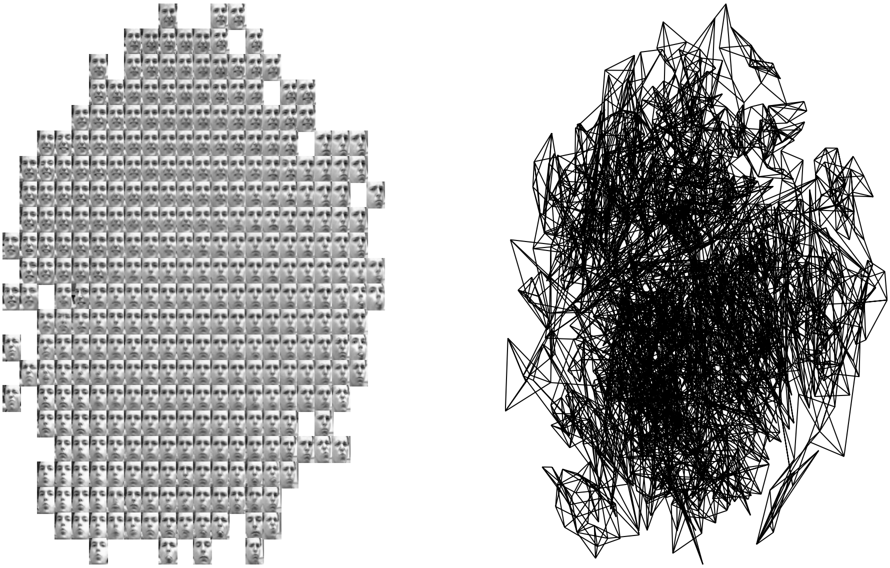


Fig. 6.55. Two-dimensional embedding of the 1965 real faces by CCA.

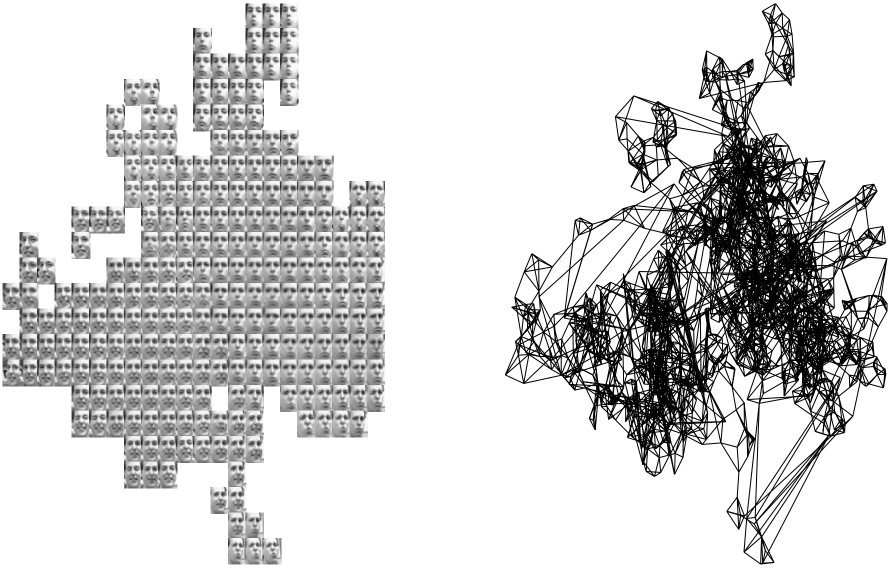


Fig. 6.56. Two-dimensional embedding of the 1965 real faces by CDA.

assumption of an underlying manifold does not really hold: data points are distributed in different clusters, which look stretched in LLE embedding. As for other embeddings, similar faces are quite well grouped, but the triangular shape of the embedding is probably not related to the true shape of the data cloud. Just like other methods relying on a graph or K-ary neighborhoods, LLE produces a very sparse embedding.

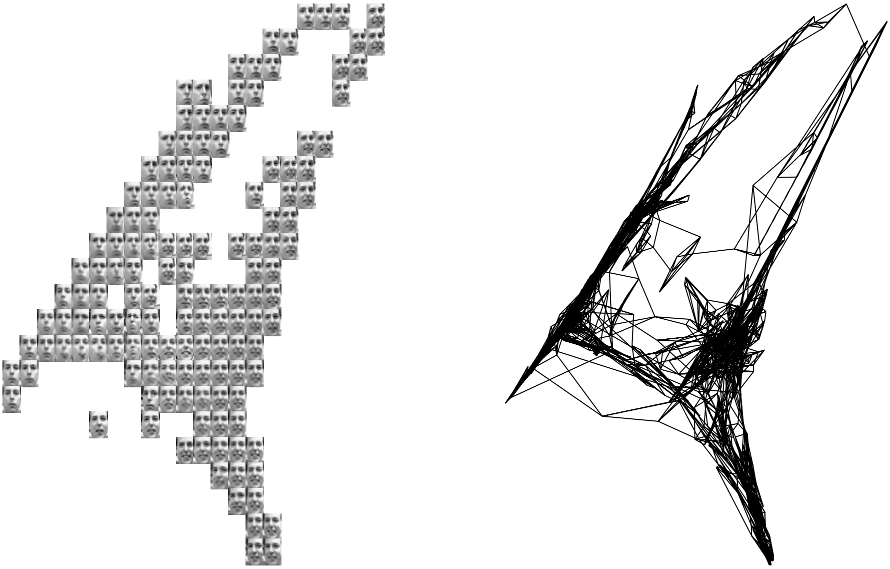


Fig. 6.57. Two-dimensional embedding of the 1965 real faces by LLE.

The embedding computed by LE (Fig. 6.58) emphasizes the separations between the main clusters of the data set, which are shrunk. This confirms the relationship between LE and spectral clustering, which has already been pointed out in the literature. The resulting embedding is sparse and cuneiform, as is the one provided by LLE. The different clusters look like pikes that are beaming in all directions; computing an embedding of higher dimensionality would probably allow us to separate the different clusters quite well.

The embedding computed by Isotop is displayed in Fig. 6.59. In contrast with LLE, the two-dimensional representation given by Isotop is denser and reveals more facial expressions. The result of Isotop looks very similar to the embedding computed by CDA: several small clusters can be distinguished. Transitions between them are usually smooth.

The SOM gives the result illustrated by Fig. 6.60. By construction, the lattice is not data-driven, and thus the embedding is rectangular. As an advantage, this allows the SOM to unfold the data set on the largest available surface. On the other hand, this completely removes any information about the

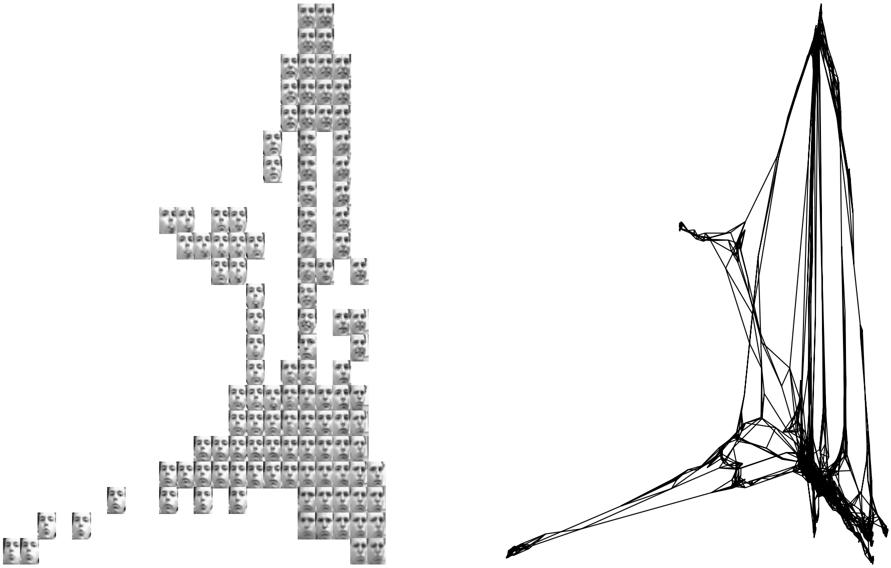


Fig. 6.58. Two-dimensional embedding of the 1965 real faces by LE.

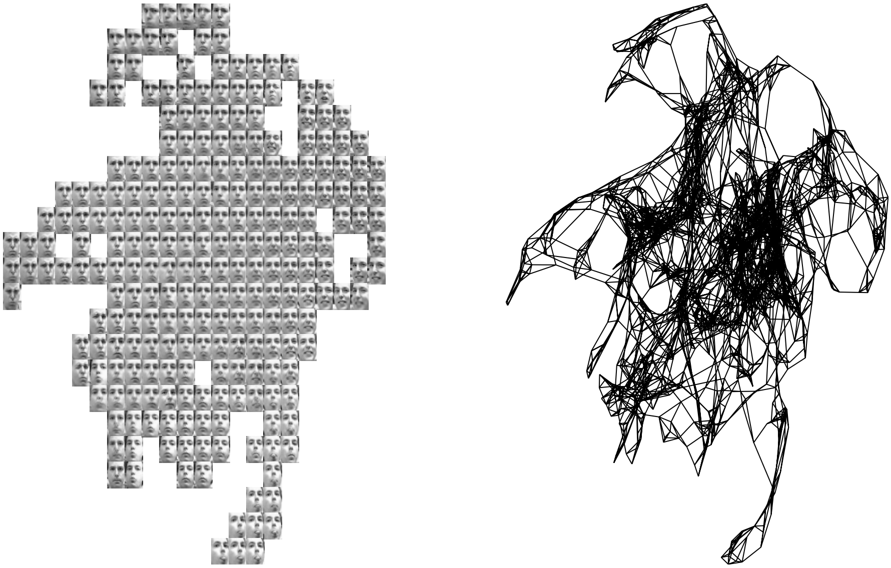


Fig. 6.59. Two-dimensional embedding of the 1965 real faces by Isotop.

initial shape of the data cloud. Although the SOM yields a visually satisfying embedding and reveals many details, some shortcomings must be remarked. First, similar faces may be distributed in two different places (different regions of the map can be folded close to each other in the data space). Second, the SOM is the only method that involves a mandatory vector quantization. Consequently, the points that are displayed as thumbnails are *not* in the data set: they are points (or prototypes) of the SOM grid.

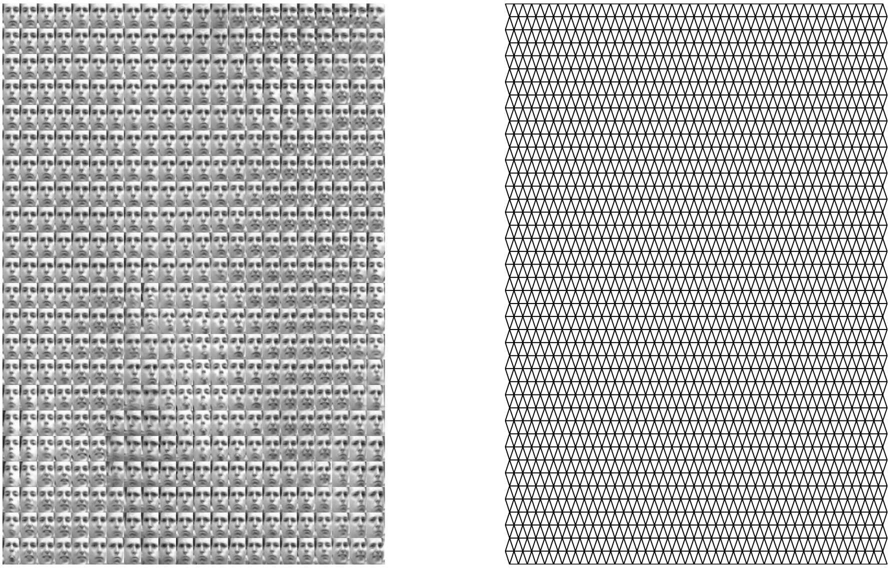


Fig. 6.60. Two-dimensional embedding of the 1965 real faces by a 44-by-44 SOM.

Until now, only the visual aspect of the embeddings has been assessed. In the same way as in Subsection 6.3.1, the mean relative rank errors (Eqs. (6.3) and (6.4)) can be computed in order to have a quantitative measure of the neighborhood preservation. The values reached by all reviewed methods are given in Fig. 6.61 for a number of neighbors ranging between 0 and 15. Metric MDS clearly performs the worst; this is also the only linear method. Performances of LLE are not really good either. The best results are achieved by Isotop and by methods working with graph distances (Isomap, GNLM, and CDA). GNLM reaches the best tradeoff between $\mathrm{MRRE}_{\mathbf{Y}\rightarrow\mathbf{X}}(K)$ and $\mathrm{MRRE}_{\mathbf{X}\rightarrow\mathbf{Y}}(K)$, followed by Isotop. On the other hand, CDA performs very well when looking at $\mathrm{MRRE}_{\mathbf{X}\rightarrow\mathbf{Y}}(K)$ only. Finally, the SOM cannot be compared directly to the other methods, since it is the only method involving a predefined lattice and a mandatory vector quantization (errors are computed on the prototypes coordinates). The first error $\mathrm{MRRE}_{\mathbf{X}\rightarrow\mathbf{Y}}(K)$ starts at low values but grows much faster than for other methods when K increases. This can be explained by the fact that the SOM can be folded on itself in the data space. On the other hand, $\mathrm{MRRE}_{\mathbf{Y}\rightarrow\mathbf{X}}(K)$ remains low because neighbors in the predefined lattice are usually close in the data space, too.

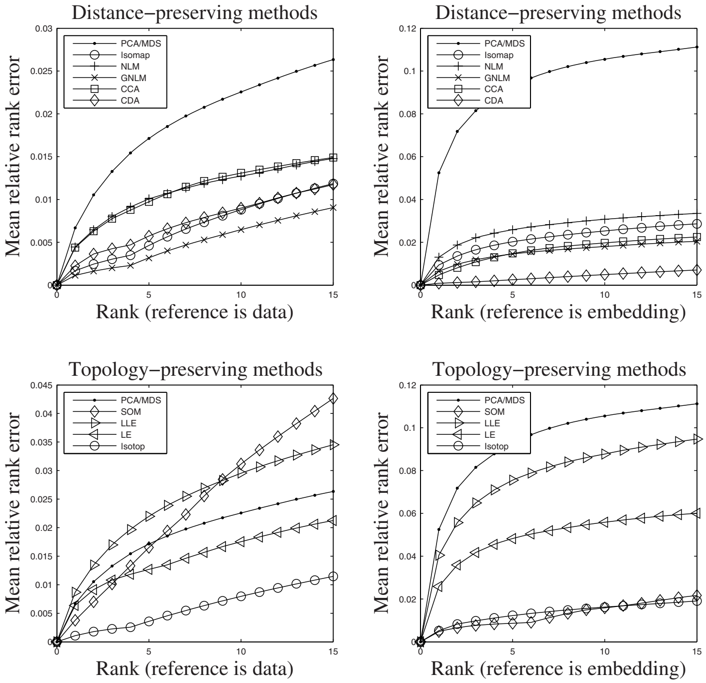


Fig. 6.61. Mean relative rank errors ($\mathrm{MRRE}_{\mathbf{Y}\rightarrow\mathbf{X}}(K)$ on the left and $\mathrm{MRRE}_{\mathbf{X}\rightarrow\mathbf{Y}}(K)$ on the right) of all NLDR methods for the real faces.

7

Conclusions

Overview. In addition to summarizing the key points of the book, this chapter attempts to establish a generic procedure (or "data flow") for the analysis of high-dimensional data. All methods used in the previous chapters are also classified from several points of view. Next, some guidelines are given for using the various methods. The chapter ends by presenting some perspectives for future developments in the field of nonlinear dimensionality reduction.

7.1 Summary of the book

The main motivations of this book are the analysis and comparison of various DR methods, with particular attention paid to nonlinear ones. Dimensionality reduction often plays an important role in the analysis, interpretation, and understanding of numerical data. In practice, dimensionality reduction can help one to extract some information from arrays of numbers that would otherwise remain useless because of their large size. To some extent, the goal consists of enhancing the readability of data. This can be achieved by visualizing data in charts, diagrams, plots, and other graphical representations.

7.1.1 The problem

As illustrated in Chapter 1, visualization becomes problematic once the dimensionality — the number of coordinates or simultaneous observations — goes beyond three or four. Usual projections and perspective techniques already reach their limits for only three dimensions ! This suggests using other methods to build low-dimensional representations of data. Beyond visualization, dimensionality reduction is also justified from a theoretical point of view by unexpected properties of high-dimensional spaces. In high dimensions, usual mathematical objects like spheres and cubes behave strangely and do not share the same nice properties as in the two- or three-dimensional cases. Other examples are the Euclidean norm, which is nearly useless in high-dimensional spaces, and the intrinsic sparsity of high-dimensional spaces (the "empty space phenomenon"). All those issues are usually called the "curse of dimensionality" and must be taken into account when processing high-dimensional data.

7.1.2 A basic solution

Historically, one of the first methods intended for the analysis of high-dimensional data was principal component analysis (PCA), introduced in Chapter 2. Starting from a data set in matrix form, and under some conditions, this method is able to perform three essential tasks:

- **Intrinsic dimensionality estimation.** This consists in estimating the (small) number of hidden parameters, called latent variables, that generated data.
- **Dimensionality reduction.** This consists in building a low-dimensional representation of data (a projection), according to the estimated dimensionality.
- **Latent variable separation.** This consists of a further transformation of the low-dimensional representation, such that the latent variables appear as mutually "independent" as possible.

Obviously, these are very desirable functionalities. Unfortunately, PCA remains a rather basic method and suffers from many shortcomings. For example, PCA assumes that observed variables are linear combinations of the latent ones. According to this data model, PCA just yields a linear projection of the observed variables. Additionally, the latent variable separation is achieved by simple decorrelation, explaining the quotes around the adjective "independent" in the above list.

For more than seven decades, the limitations of PCA have motivated the development of more powerful methods. Mainly two directions have been explored: namely, dimensionality reduction and latent variable separation.

7.1.3 Dimensionality reduction

Much work has been devoted to designing methods that are able to reduce the data dimensionality in a nonlinear way, instead of merely projecting data with a linear transformation. The first step in that direction was made by reformulating the PCA as a distance-preserving method. This yielded the classical metric multidimensional scaling (MDS) in the late 1930s (see Table 7.1). Although this method remains linear, like PCA, it is the basis of numerous nonlinear variants described in Chapter 4. The most widely known ones are undoubtedly nonmetric MDS and Sammon's nonlinear mapping (published in the late 1960s). Further optimizations are possible, by using stochastic techniques, for example, as in curvilinear component analysis (CCA), published in the early 1990s. Besides this evolution toward more and more complex algorithms, recent progress has been accomplished in the family of distance-preserving methods by replacing the usual Euclidean distance with another metric: the geodesic distance, introduced in the late 1990s. This particular distance measure is especially well suited for dimensionality reduction. The unfolding of nonlinear manifolds is made much easier with geodesic distances than with Euclidean ones.

Geodesic distances, however, cannot be used as such, because they hide a complex mathematical machinery that would create a heavy computational burden in practical cases. Fortunately, geodesic distances may be approximated in a very elegant way by graph distances. To this end, it suffices to connect neighboring points in the data set, in order to obtain a graph, and then to compute the graph distances with Dijkstra's algorithm [53], for instance.

A simple change allows us to use graph distances instead of Euclidean ones in classical distance-preserving methods. Doing so transforms metric MDS, Sammon's NLM and CCA in Isomap (1998), geodesic NLM (2002), and curvilinear distance analysis (2000), respectively. Comparisons on various examples in Chapter 6 clearly show that the graph distance outperforms the traditional Euclidean metric. Yet, in many cases and in spite of all its advantages, the graph distance is not the panacea: it broadens the set of manifolds that can easily be projected by distance preservation, but it does not help in all cases. For that reason, the algorithm that manages the preservation of distances fully keeps its importance in the dimensionality reduction process. This explains why the flexibility of GNLM and CDA is welcome in difficult cases where Isomap can fail.

Distance preservation is not the sole paradigm used for dimensionality reduction. Topology preservation, introduced in Chapter 5, is certainly more powerful and appealing but also more difficult to implement. Actually, in order to be usable, the concept of "topology" must be clearly defined; its translation from theory to practice does not prove as straightforward as measuring a distance. Because of that difficulty, topology-preserving methods like Kohonen's self-organizing maps appeared later (in the early 1980s) than distance-based methods. Other methods, like the generative topographic mapping (1995), may be viewed as principled reformulations of the SOM, within a probabilistic framework. More recent methods, like locally linear embedding (2000) and Isotop (2002), attempt to overcome some limitations of the SOM.

In Chapter 5, methods are classified according to the way they model the topology of the data set. Typically, this topology is encoded as neighborhood relations between points, using a graph that connects the points, for instance. The simplest solution consists of predefining those relations, without regards to the available data, as it is done in an SOM and GTM. If data are taken into account, the topology is said to be data-driven, like with LLE and Isotop. While data-driven methods generally outperform SOMs for dimensionality reduction purposes, the latter remains a reference tool for 2D visualization.

ANN	DR	Method	Author(s) & reference(s)
	1901	PCA	Pearson [149]
	1933	PCA	Hotelling [92]
	1938	classical metric MDS	Young & Householder [208]
1943		formal neuron	McCulloch & Pitts [137]
	1946	PCA	Karhunen [102]
	1948	PCA	Loève [128]
	1952	MDS	Torgerson [182]
1958		Perceptron	Rosenblatt [157]
1959		Shortest paths in a graph	Dijkstra [53]
	1962	nonmetric MDS	Shepard [171]
	1964	nonmetric MDS	Kruskal [108]
1965		K-means (VQ)	Forgy [61]
1967		K-means (VQ)	MacQueen [61]
		ISODATA (VQ)	Ball & Hall [8]
	1969	PP	Kruskal [109]
		NLM (nonlinear MDS)	Sammon [109]
1969		Perceptron	Minsky & Papert's paper [138]
	1972	PP	Kruskal [110]
	1973	SOM	von der Malsburg [191]
	1974	PP	Friedman & Tukey [67]
1974		Back-propagation	Werbos [201]
1980		LBG (VQ)	Linde, Buzo & Gray [124]
	1982	SOM (VQ & NLDR)	Kohonen [104]
1982		Hopfield network	Hopfield [91]
		LLoyd (VQ)	Lloyd [127]
	1984	Principal curves	Hastie & Stuetzle [79, 80]
1985		Competitive learning (VQ)	Rumelhart & Zipser [162, 163]
1986		Back-propagation & MLP	Rumelhart, Hinton & Williams [161, 160]
		BSS/ICA	Jutten [99, 98, 100]
	1991	Autoassociative MLP	Kramer [107, 144, 183]
	1992	"Neural" PCA	Oja [145]
	1993	VQP (NLM)	Demartines & Hérault [46]
		Autoassociative ANN	DeMers & Cottrell [49]
	1994	Local PCA	Kambhatla & Leen [101]
	1995	CCA (VQP)	Demartines & Hérault [47, 48]
		NLM with ANN	Mao & Jain [134]
	1996	KPCA	Schölkopf, Smola & Müller [167]
		GTM	Bishop, Svensén & Williams [22, 23, 24]
	1997	Normalized cut (spectral clustering)	Shi & Malik [172, 199]
	1998	Isomap	Tenenbaum [179, 180]
	2000	CDA (CCA)	Lee & Verleysen [116, 120]
		LLE	Roweis & Saul [158]
	2002	Isotop (MDS)	Lee [119, 114]
		LE	Belkin & Niyogi [12, 13]
		Spectral clustering	Ng, Jordan & Weiss [143]
		Coordination of local linear models	Roweis, Saul & Hinton [159]
	2003	HLLE	Donoho & Grimes [56, 55]
	2004	LPP	He & Niyogi [81]
		SDE (MDS)	Weinberger & Saul [196]
	2005	LMDS (CCA)	Venna & Kaski [186, 187]
	2006	Autoassociative ANN	Hinton & Salakhutdinov [89]

Table 7.1. Timeline of DR methods. Major steps in ANN history are given as milestones. Spectral clustering has been added because of its tight relationship with spectral DR methods.

7.1.4 Latent variable separation

Starting from PCA, the other direction that can be explored is latent variable separation. The first step in that direction was made with projection pursuit (PP; see Table 7.1) [109, 110, 67]. This technique, which is widely used in exploratory data analysis, aims at finding "interesting" (linear) one- or two-dimensional projections of a data set. Axes of these projections can then be interpreted as latent variables. A more recent approach, initiated in the late 1980s by Jutten and Hérault [99, 98, 100], led to the flourishing development of blind source separation (BSS) and independent component analysis (ICA). These fields propose more recent but also more principled ways to tackle the problem of latent variable separation. In contrast with PCA, BSS and ICA can go beyond variable decorrelation: most methods involve an objective function that can be related to statistical independence.

In spite of its appealing elegance, latent variable separation does not fit in the scope of this book. The reason is that most methods remain limited to linear data models. Only GTM can be cast within that framework: it is one of the rare NLDR methods that propose a latent variable model, i.e. one that considers the observed variables to be functions of the latent ones. Most other methods follow a more pragmatic strategy and work in the opposite direction, by finding any set of variables that give a suitable low-dimensional representation of the observed variables, regardless of the true latent variables. It is noteworthy, however, that GTM involves a nonlinear mapping and therefore offers no guarantee of recovering the true latent variables either, despite its more complex data model.

More information on projection pursuit can be found in [94, 66, 97]. For BSS and ICA, many details and references can be found in the excellent book by Hyvärinen, Karhunen, and Oja [95].

7.1.5 Intrinsic dimensionality estimation

Finally, an important key to the success of both dimensionality reduction and latent variable separation resides in the right estimation of the intrinsic dimensionality of data. This dimensionality indicates the minimal number of variables or free parameters that are needed to describe the data set without losing the information it conveys. The word "information" can be understood in many ways: it can be the variance in the context of PCA, for instance. Within the framework of manifold learning, it can also be the manifold "structure" or topology; finding the intrinsic dimensionality then amounts to determining the underlying manifold dimensionality. Chapter 3 reviews a couple of classical methods that can estimate the intrinsic dimensionality of a data set. A widely used approach consists of measuring the fractal dimension of the data set. Several fractal dimensions exist: the most-known ones are the correlation dimension and the box-counting dimension. These measures come from subdomains of physics, where they are used to study dynamical systems. Although they are often criticized in the physics literature, their claimed shortcomings do not really matter within the framework of dimensionality reduction. It is just useful to know that fractal dimensions tend to underestimate the true dimensionality [174] and that noise may pollute the estimation. But if the measure of the fractal dimension fails, then it is very likely that the data set is insufficient or too noisy and that any attempt to reduce the dimensionality will fail, too.

Other methods to estimate the intrinsic dimensionality are also reviewed in Chapter 3. For example, some DR methods can also be used to estimate the intrinsic dimensionality: they are run iteratively, with a decreasing target dimension, until they fail. The intrinsic dimensionality may then be assumed to be equal to the smallest target dimension before failure. This is a “trial-and-error” approach. Obviously, this way of estimating the dimensionality largely depends on the method used to reduce the dimensionality. The result may vary significantly just by changing the type of dimensionality reducer (distance- or topology-preserving method, Euclidean or graph distance, etc.). Moreover, the computational cost of repeating the dimensionality reduction to obtain *merely* the dimensionality may rapidly become prohibitive. From this point of view, methods that can build projections in an incremental way (see Subsection 2.5.7), such as PCA, Local PCA, Isomap, or SDE, appear as the best compromise because a single run suffices to determine the projections of all possible dimensionalities at once. In contrast with fractal dimensions, the trial-and-error technique tends to overestimate the true intrinsic dimensionality.

Section 3.4 compares various methods for the estimation of the intrinsic dimensionality. The correlation dimension (or another fractal dimension) and local PCA give the best results on the proposed data sets. Indeed, these methods are able to estimate the dimensionality on different scales (or resolutions) and thus yield more informative results.

7.2 Data flow

This section proposes a generic data flow for the analysis of high-dimensional data. Of course, the accent is put on the word “generic”: the proposed pattern must obviously be particularized to each application. However, it provides a basis that has been proven effective in many cases.

As a starting point, it is assumed that data consist of an unordered set of vectors. All vector entries are real numbers; there are no missing data.

7.2.1 Variable Selection

The aim of this first step is to make sure that all variables or signals in the data set convey useful information about the phenomenon of interest. Hence, if some variables or signals are zero or are related to another phenomenon, a variable selection must be achieved beforehand, in order to discard them. To some extent, this selection is a “binary” dimensionality reduction: each observed variable is kept or thrown away. Variable selection methods are beyond the scope of this book; this topic is covered in, e.g., [2, 96, 139].

7.2.2 Calibration

This second step aims at “standardizing” the variables. When this is required, the average of each variable is subtracted. Variables can also be scaled if needed. The division by the standard deviation is useful when the variables come from various origins. For example, meters do not compare with kilograms, and kilometers do not with grams. Scaling the variables helps to make them more comparable.

Sometimes, however, the standardization can make things worse. For example, an almost-silent signal becomes pure noise after standardization. Obviously, the knowledge that it was silent is important and should not be lost. In the ideal case, silent signals and other useless variables are eliminated by the above-mentioned variable selection. Otherwise, if no standardization has been performed, further processing methods can still remove almost-zero variables. (See Subsection 2.4.1 for a more thorough discussion.)

7.2.3 Linear dimensionality reduction

When data dimensionality is very high, linear dimensionality reduction by PCA may be very usesul to suppress a large number of useless dimensions. Indeed, PCA clearly remains one of the best techniques for “hard” dimensionality reduction. For this step, the strategy consists in elimating the largest number of variables while maintaining the reconstruction error very close to zero. This is achieved in order to make the operation as “transparent” as possible, i.e., nearly reversible. This also eases the work to be achieved by subsequent nonlinear methods (e.g., for a further dimensionality reduction). If the dimensionality is not too high, or if linear dimensionality causes a large reconstruction error, then PCA may be skipped.

In some cases, whitening can also be used [95]. Whitening, also known as sphering, is closely related to PCA. In the latter, the data space is merely rotated, using an orthogonal matrix, and the decorrelated variables having a variance close to zero are discarded. In whitening an additional step is used for scaling the decorrelated variables, in order to end up with unit-variance variables. This amounts to performing a standardization, just as described above, after PCA instead of before. Whereas the rotation involved in PCA does not change pairwise distances in the data set, the additional transformation achieved by whitening does, like the standardization. For zero-mean variables, Euclidean distances measured after whitening are equivalent to Mahalanobis distances measured in the raw data set (with the Mahalanobis matrix being the inverse of the data set covariance matrix).

7.2.4 Nonlinear dimensionality reduction

Nonlinear methods of dimensionality reduction may take over from PCA once the dimensionality is no longer too high, between a few tens and a few hundreds, depending on the chosen method. The use of PCA as preprocessing is justified by the fact that most nonlinear methods remain more sensitive to the curse of dimensionality than PCA due to their more complex model, which involves many parameters to identify.

Typically, nonlinear dimensionality reduction is the last step in the data flow. Indeed, the use of nonlinear methods transforms the data set in such a way that latent variable separation becomes difficult or impossible.

7.2.5 Latent variable separation

In the current state of the art, most methods for latent variable separation are incompatible with nonlinear dimensionality reduction. Hence, latent variable separation appears more as an alternative step to nonlinear dimensionality reduction than a subsequent one. The explanation is that most methods for latent variable separation like ICA assume in fact that the observed variables are linear combinations of the latent ones [95]. Without that assumption, these methods may not use statistical independence as a criterion to separate the variables. Only a few methods can cope with restricted forms of nonlinear mixtures like, such as, for example, postnonlinear mixtures [205, 177].

7.2.6 Further processing

Once dimensionality reduction or latent variable separation is achieved, the transformed data may be further processed, depending on the targeted application. This can range from simple visualization to automated classification or function approximation. In the two last cases, unsupervised learning is followed by supervised learning.

In summary, the proposed generic data flow for the analysis of high-dimensional data goes through the following steps:

1. **(Variable selection.)** This step allows the suppression of useless variables.
2. **Calibration.** This step gathers all preprocessings that must or may be applied to data (mean subtraction, scaling, or standardization, etc.).
3. **Linear dimensionality reduction.** This step usually consists of performing PCA (data may be whitened at the same time, if necessary).
4. **Nonlinear dimensionality reduction and/or latent variable separation.** These (often incompatible) steps are the main ones; they allow us to find "interesting" representations of data, by optimizing either the number of required variables (nonlinear dimensionality reduction) or their independence (latent variable separation).
5. **(Further processing.)** Visualization, classification, function approximation, etc.

Steps between paratheses are topics that are not covered in this book.

7.3 Model complexity

It is noteworthy that in the above-mentioned data flow, the model complexity grows at each step. For example, if N observations of D variables or signals are available, the calibration determines D means and D standard deviations; the time complexity to compute them is then $\mathcal{O}(DN)$. Next, for PCA, the covariance matrix contains $D(D-1)/2$ independent entries and the time complexity to compute them is $\mathcal{O}(D^2N)$. Obtaining a P-dimensional projection requires $\mathcal{O}(PDN)$ additional operations.

Things become worse for nonlinear dimensionality reduction. For example, a typical distance-preserving method requires $N(N-1)/2$ memory entries to store all pairwise distances. The time complexity to compute them is $\mathcal{O}(DN^2)$, at least for Euclidean distances. For graph distances indeed, the time complexity grows further to $\mathcal{O}(PN^2 \log N)$. In order to obtain a P-dimensional embedding, an NLDR method relying on a gradient descent such as— Sammon's NLM —requires $\mathcal{O}(PN^2)$ operations for a single iteration. On the other hand, a spectral method requires the same amount of operations per iteration, but the eigensolver has the advantage of converging much faster.

To some extent, progress of NLDR models and methods seems to be related not only to science breakthroughs but also to the continually increasing power of computers, which allows us to investigate directions that were previously out of reach from a practical point of view.

7.4 Taxonomy

Figure 7.1 presents a nonexhaustive hierarchy tree of some unsupervised data analysis methods, according to their purpose (latent variable separation or dimensionality reduction). This figure also gives an overview of all methods described in this book, which focuses on nonlinear dimensionality reduction based mainly on “geometrical” concepts (distances, topology, neighborhoods, manifolds, etc.).

Two classes of NLDR methods are distinguished in this book: those trying to preserve pairwise distances measured in the data set and those attempting to reproduce the data set topology. This distinction may seem quite arbitrary, and other ways to classify the methods exist. For instance, methods can be distinguished according to their algorithmic structure. In the latter case, spectral methods can be separated from those relying on iterative optimization schemes like (stochastic) gradient ascent/descent. Nevertheless, this last distinction seems to be less fundamental.

Actually, it can be observed that all distance-preserving methods involve pairwise distances either directly (metric MDS, Isomap) or with some kind of weighting (NLM, GNLM, CCA, CDA, SDE). In (G)NLM, this weighting is proportional to the inverse of the Euclidean (or geodesic) distances measured in the data space, whereas a decreasing function of the Euclidean distances in the embedding space is used in CCA and CDA. For SDE, only Euclidean distances to the K nearest neighbors are taken into account, while others are simply forgotten and replaced by those determined during the semidefinite programming step.

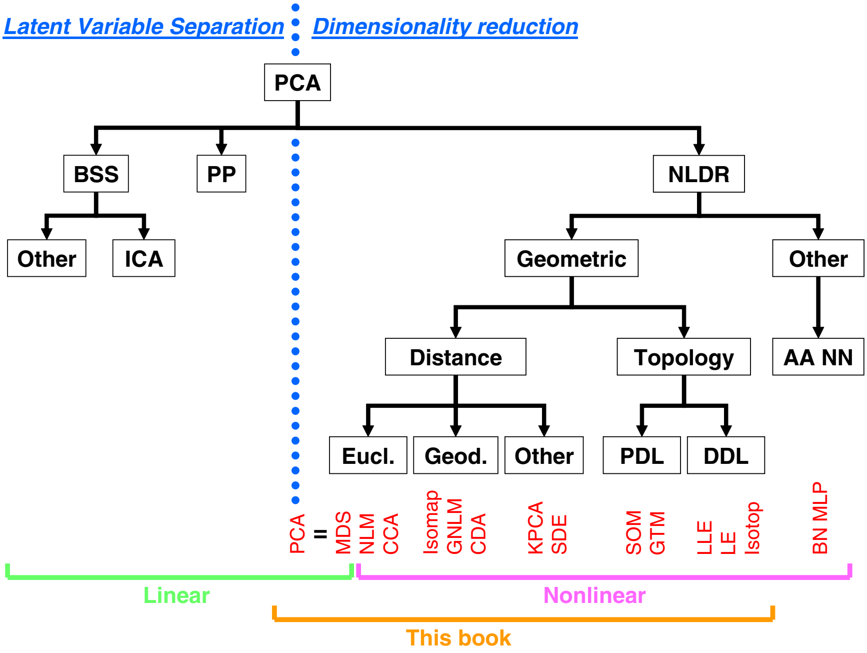


Fig. 7.1. Methods for latent variable separation and dimensionality reduction: a nonexhaustive hierarchy tree. Acronyms: PCA, principal component analysis; BSS, blind source separation; PP, projection pursuit; NLDR, nonlinear dimensionality reduction; ICA, independent component analysis; AA NN, auto-associative neural network; PDL, predefined lattice; DDL, data-driven lattice. Methods are shown as tree leaves.

On the other hand, in topology-preserving methods pairwise distances are never used directly. Instead they are replaced with some kind of similarity measure, which most of the time is a decreasing function of the pairwise distances. For instance, in LE only distances to the K nearest neighbors are involved; next the heat kernel is applied to them (possibly with an infinite temperature) and the Laplacian matrix is computed. This matrix is such that off-diagonal entries in a row or column are always lower than the corresponding entry on the diagonal. A similar reasoning leads to the same conclusion for LLE. In an SOM or Isotop, either grid distances (in the embedding space) or graph distances (in the data space) are used as the argument of a Gaussian kernel.

In the case of spectral methods, this distinction between distance and topology preservation has practical consequences. In all distance-preserving methods, the eigensolver is applied to a dense matrix, whose entries are either Euclidean distances (metric MDS), graph distances (Isomap), or distances optimized by means of semidefinite programming (SDE). In all these methods, the top eigenvectors are sought (those associated with the largest eigenvalues). To some extent, these eigenvectors form the solution of a maximization problem; in the considered case, the problem primarily consists of maximizing the variance in the embedding space, which is directly related to the associated eigenvalues.[1]

The situation gets reversed for topology-preserving methods. Most of the time, in this case, the eigensolver is applied to a sparse matrix and the bottom eigenvectors are sought, those associated with the eigenvalues of lowest (but nonzero) magnitude. These eigenvectors identify the solution of a minimization problem. The objective function generally corresponds to a local reconstruction error or distortion measure (see Subsections 5.3.1 and 5.3.2 about LLE and LE, respectively).

A duality relationship can be established between those maximizations and minimizations involving, respectively, dense and sparse Gram-like matrices, as sketched in [164] and more clearly stated in [204]. Hence to some extent distance and topology preservation are different aspects or ways to formulate the same problem. Obviously, it should be erroneous to conclude from the unifying theoretical framework described in [204] that all spectral methods are equivalent in practice ! This is not the case; experimental results in Chapter 6 show the large variety that can be observed among their results. For instance, following the reasoning in [164, 78], it can easily be demonstrated that under some conditions, the bottom eigenvectors of a Laplacian matrix (except the last one associated with a null eigenvalue) correspond to the leading eigenvectors obtained from a double-centered matrix of pairwise commute-time distances (CTDs). It can be shown quite easily that CTDs respect all axioms of a distance measure (Subsection 4.2.1). But unlike Euclidean distances, CTDs cannot be computed simply by knowing coordinates of two points. Instead, CTD distances are more closely related to graph distances, in the sense that other known points are involved the computation. Actually, it can be shown that the easiest way to obtain the matrix of pairwise CTDs consists of computing the pseudo-inverse of a Laplacian matrix. As a direct consequence, the leading eigenvectors of the CTD matrix precisely correspond to the bottom eigenvectors of the Laplacian matrix, just as stated above. Similarly, eigenvalues of the CTD matrix are inversely proportional to those of the Laplacian matrix. Therefore, even if distance- and topology-preserving spectral methods are equivalent from a theoretical viewpoint, it remains useful to keep both frameworks in practice, as the formulation of a particular method can be made easier or more natural in the one or the other.

[1] Of course, this maximization of the variance can easily be reformulated into a minimization of the reconstruction error, as observed in Subsection 2.4.2.

7.4.1 Distance preservation

Distance-preserving methods can be classified according to the distance and the algorithm they use. Several kinds of distances and kernels can be used. Similarly, the embedding can be obtained by three different types of algorithms. The first one is the spectral decomposition found in metric MDS. The second one is the quasi-Newton optimization procedure implemented in Sammon's NLM. The third one is the stochastic optimization procedure proposed in curvilinear component analysis. Table 7.2 shows the different combinations that have been described in the literature. The table also proposes an alternative naming convention. The first letter indicates the distance or kernel type (E for Euclidean, G for geodesic/graph, C for commute-time distance, K for fixed kernel, and O for optimized kernel), whereas the three next letters refer to the algorithm (MDS for spectral decomposition, NLM for quasi-Newton optimization, and CCA for CCA-like stochastic gradient descent).

Table 7.2. Classification of distance-preserving NLDR methods, according to the distance/kernel function and the algorithm. A unifying naming convention is proposed (the real name of each method is given between parentheses).

	MDS algorithm	NLM algorithm	CCA algorithm
Euclidean	EMDS (metric MDS)	ENLM (NLM)	ECCA (CCA)
Geodesic (Dijkstra)	GMDS (Isomap)	GNLM	GCCA (CDA)
Commute time	CMDS (LE)		
Fixed kernel	KMDS (KPCA)		
Optimized kernel (SDP)	OMDS (SDE/MVU)		

It is noteworthy that most methods have an intricate name that often gives few or no clues about their principle. For instance, KPCA is by far closer to metric MDS than to PCA. While SDE stands for semidefinite embedding, it should be remarked that all spectral methods compute an embedding from a positive semidefinite Gram-like matrix. The author of SDE renamed his method MVU, standing for maximum variance unfolding [197]; this new name does not shed any new light on the method: all MDS-based methods (like Isomap and KPCA, for instance) yield an embedding having maximal variance. The series can be continued, for instance, with PCA (pricipal component analysis), CCA (curvilinear component analysis), and CDA (curvilinear distances analysis), whose names seem to be designed to ensure a kind of filiation while remaining rather unclear about their principle. The name Isomap, which stands for Isometric feature mapping [179], is quite unclear too, since all distance-preserving NLDR methods attempt to yield an isometric embedding. Unfortunately, in most practical cases, perfect isometry is not reached.

Looking back at Table 7.2, the third and fourth rows contain methods that were initially not designed as distance-preserving methods. Regarding KPCA, the kernels listed in [167] are given for their theoretical properties, without any geometrical justification. However, the application of the kernel is equivalent to mapping data to a feature space in which a distance-preserving embedding is found by metric MDS. In the case of LE, the duality described in [204] and the connection with commute-time distances detailed in [164, 78] allow it to occupy a table entry.

Finally, it should be remarked that the bottom right corner of Table 7.2 contains many empty cells that could give rise to new methods with potentially good performances.

7.4.2 Topology preservation

As proposed in Chapter 5, topology-preserving methods fall into two classes. On one hand, methods like Kohonen's self-organizing maps and Svensén's GTM map data to a discrete lattice that is predefined by the user. On the other hand, more recent techniques like LLE and Isotop automatically build a data-driven lattice, meaning that the shape of the lattice depends on the data and is entirely determined by them. In most cases this lattice is a graph induced by k-ary or ϵ-ball neighborhoods, which provides a good discrete approximation of the underlying manifold topology. From that point of view, the corresponding methods can be qualified to be graph- or manifold-driven.

Table 7.3 is an attempt to classify topology-preserving methods as it was done in Table 7.2 for distance-preserving methods. If PCA is interpreted as a method that fits a plane in the data space in order to capture as much

Table 7.3. Classification of topology-preserving NLDR methods, according to the kind of lattice and algorithm. The acronyms ANN, MLE and EM in the first row stand for artificial neural network, maximum likelihood estimation and expectation-maximization respectively.

	ANN-like	MLE by EM	Spectral
Predefined lattice	SOM	GTM	
Data-driven lattice	Isotop		LLE, LE

variance as possible after projection on this plane, then to some extent running an SOM can then be seen as a way to fit a nonrigid (or articulated) piece of plane within the data cloud. Isotop, on the contrary, follows a similar strategy in the opposite direction: an SOM-like update rule is used for embedding a graph deduced from the data set in a low-dimensional space.

By essence, GTM solves the problem in a similar way as an SOM would. The approach is more principled, however, and the resulting algorithm works in a totally different way. A generative model is used, in which the latent space is fixed a priori and whose mapping parameters are identified by statistical inference.

Finally, topology-preserving spectral methods, like LLE and LE, develop a third approach to the problem. They build what can be called an “affinity” matrix [31], which is generally sparse; after double-centering or application of the Laplacian operator, some of its bottom eigenvectors form the embedding. A relationship between LLE and LE is established in [13], while the duality described in [204] allows us to relate both methods to distance-preserving spectral methods.

7.5 Spectral methods

Since 2000 most recent NLDR methods have been spectral, whereas methods based on sophisticated (stochastic) gradient ascent/descent were very popular during the previous decades. Most of these older methods were designed and developed in the community of artificial neural networks (ANNs). Methods like Kohonen’s SOM and Demartines’ VQP [46] (the precursor of CCA) share the same distributed structure and algorithmic scheme as other well-known ANNs like the multilayer perceptron.

Since the late 1990s, the ANN community has evolved and split into two groups. The first one has joined the fast-growing and descriptive field of neurosciences, whereas the second has been included in the wide field of machine learning, which can be seen as a fundamental and theory-oriented emanation of data mining. The way to tackle problems is indeed more formal in machine learning than it was in the ANN community and often resorts to concepts coming from statistics, optimization and graph theory, for example. The growing interest in spectral methods stems at least partly from this different perspective. Spectral algebra provides an appealing and elegant framework where NLDR but also clustering or other optimization problems encountered in data analysis can be cast within. From the theoretical viewpoint, the central problem of spectral algebra, i.e., finding eigenvalues and eigenvectors of a given matrix, can be translated into a convex objective function to be optimized. This guarantees the existence of a unique and global maximum; in addition, the solutions to the problem are also “orthogonal” in many cases, giving the opportunity to divide the problem into subproblems that can be solved successively, yielding the eigenvectors one by one. Another advantage of spectral algebra is to associate practice with theory: rather robust and efficient eigensolvers, with well-studied properties, are widely available.

This pretty picture hides some drawback however. The first to be mentioned is the "rigidity" of the framework provided by spectral algebra. In the case of NLDR, all spectral methods can be interpreted as performing metric MDS on a double-centered kernel matrix instead of a Gram matrix [31, 167, 203, 78, 15, 198, 17]. This kernel matrix is generally built in one of the following two ways:

- Apply a kernel function to the matrix of squared Euclidean pairwise distances or directly to the Gram matrix of dot products.
- Replace the Euclidean distance with some other distance function in the matrix of squared pairwise distances.

The first approach is followed in KPCA and SDE, and the second one in Isomap, for which graph distances are used instead of Euclidean ones. Eventually, the above-mentioned duality (Section 7.4) allows us to relate methods working with bottom eigenvectors of sparse matrices, like LLE, LE, and their variants, to the same scheme. Knowing also that metric MDS applied to a matrix of pairwise Euclidean distances yields a linear projection of the data set, it appears that the ability of all recent spectral methods to provide a nonlinear embedding actually relies merely on the chosen distance or kernel function. In other words, the first step of a spectral NLDR method consists of building a kernel matrix (under some conditions) or a distance matrix (with a non-Euclidean distance), which amounts to implicitly mapping data to a feature space in a nonlinear way. In the second step, metric MDS enables us to compute a linear projection from the feature space to an embedding space having the desired dimensionality. Hence, although the second step is purely linear, the succession of the two steps yields a nonlinear embedding.

A consequence of this two-step data processing is that for most methods the optimization (performed by the eigensolver) occurs in the second step only. This means that the nonlinear mapping involved in the first step (or equivalently the corresponding kernel) results from a more or less "arbitrary" user's choice. This can explain the large variety of spectral methods described in the literature: each of them describes a particular way to build a Gram-like matrix before computing its eigenvectors. A gradation of the "arbitrariness" of the data transformation can be established as follows.

The most arbitrary transformation is undoubtedly the one involved in KPCA. Based on kernel theory, KPCA requires the data transformation to be induced by a kernel function. Several kernel functions in agreement with the theory are proposed in [167], but there is no indication about which one to choose. Moreover, most of these kernels involve one or several metaparameters; again, no way to determine their optimal value is given.

In LLE, LE, Isomap, and their variants, the data transformation is based on geometric information extracted from data. In all cases, K-ary neighborhoods or ϵ-balls are used to induce a graph whose edges connect the data points. This graph, in turn, brings a discrete approximation of the underlying manifold topology/structure. In that perspective, it can be said that LLE, LE and Isomap are graph- or manifold-driven. They induce a “data-dependent” kernel function [15]. In practice, this means that the kernel value stored at row i and column j in the Gram-like matrix does not depend only on the ith and jth data points, but also on other points in the data set [78]. Since the Gram-like matrix built by those methods is an attempt to take into account the underlying manifold topology, it can be said the induced data mapping is less “arbitrary” than in the case of KPCA. Nevertheless, the Gram-like matrix still depends on K or ϵ; changing the value of these parameters modifies the induced data mapping, which may can lead to a completely different embedding. LLE seems to be particularly sensitive to these parameters, as witnessed in the literature [166, 90]. In this respect, the numerical stability of the eigenvectors computed by LLE can also be questioned [30] (the tail of the eigenspectrum is flat, i.e., the ratio of successive lowest-amplitude eigenvalues is close to one).

The highest level in the proposed gradation is eventually reached by SDE. In contrast with all other spectral methods, SDE is (currently) the only one that optimizes the Gram-like matrix before the metric MDS step. In Isomap, Euclidean distances within the K-ary neighborhoods are kept (by construction, graph distances in that case reduce to Euclidean distances), whereas distances between non-neighbors are replaced with graph distances, assuming that these distances subsequently lead to a better embedding. In SDE, distances between nonneighbors can be seen as parameters whose value is optimized by semidefinite programming. This optimization scheme is specifically designed to work in matrix spaces, accepts equality or inequality constraints on the matrix entries, and ensures that their properties are maintained (e.g., positive or negative semidefiniteness). An additional advantage of semidefinite programming is that the objective function is convex, ensuring the existence of a unique global maximum. In the case of a convex developable manifold, if the embedding provided by SDE is qualified to be optimal, then Isomap yields a good approximation of it although it is generally suboptimal [204] (see an example in Fig. 6.6). In the case of a nonconvex or nondevelopable manifold, the two methods behave very differently.

In summary, the advantages of spectral NLDR methods are as follows: they benefit from a strong and sound theoretical framework; eigensolvers are also efficient, leading to methods that are generally fast. Once the NLDR problem is cast within the framework, a global optimum of the objective function can be reached without implementing any “exotic” optimization procedure: it suffices to call an eigensolver, which can be found in many software toolboxes or libraries. However, it has been reported that Gram-like matrices built from sparse graphs (e.g., K-ary neighborhoods) can lead to ill-conditioned and/or ill-posed eigenproblems [30]. Moreover, the claim that spectral methods provide a global optimum is true but hides the fact that the actual nonlinear transformation of data is not optimized, except in SDE. In the last case, the kernel optimization unfortunately induces a heavy computational burden.

7.6 Nonspectral methods

Experimentally, nonspectral methods reach a better tradeoff between flexibility and computation time than spectral methods. The construction of methods like NLM or CCA/CDA consists of defining an objective function and then choosing an adequate optimization technique. This goes in the opposite direction of the thought process behind spectral methods, for which the objective function is chosen with the a priori idea to translate it easily into an eigenproblem.

Hence, more freedom is left in the design of nonspectral methods than in that of spectral ones. The price to pay is that although they are efficient in practice, they often give few theoretical guarantees. For instance, methods relying on a (stochastic) gradient ascent/decent can fall in local optima, unlike spectral methods. The optimization techniques can also involve metaparameters like step sizes, learning rates, stopping criteria, tolerances, and numbers of iterations. Although most of these parameters can be left to their default values, some of them can have a nonnegligible influence on the final embedding.

This is the case of the so-called neighborhood width involved in CCA/CDA, SOMs and Isotop. Coming from the field of artificial neural networks, these methods all rely on stochastic gradient ascent/descent or resort to update rules that are applied in the same way. As usual in stochastic update rules, the step size or learning rate is scheduled to slowly decrease from one iteration to the other, in order to reach convergence. In the above-mentioned algorithms, the neighborhood width is not kept constant and follows a similar schedule. This makes their dynamic behavior quite difficult to analyze, since the optimization process is applied to a varying objective function that depends on the current value of the neighborhood width. Things are even worse in the cases of SOMs and Isotop, since the update rules are empirically built: there exists no objective function from which the update rules can be derived. Since the objective function, when it exists, is nonconstant and depends on the neighborhood width, it can be expected that embeddings obtained with these methods will depend on the same parameter, independently from the fact that the method can get stuck in local optima.

The presence of metaparameters like the neighborhood width can also be considered an advantage, to some extent. It does provide additional flexibility to nonspectral methods in the sense that for a given method, the experienced user can obtain different behaviors just by changing the values of the metaparameters.

Finally, it is noteworthy that the respective strategies of spectral and nonspectral NLDR methods completely differ. Most spectral methods usually transform data (nonlinearly when building the Gram-like matrix, and then linearly when solving the eigenproblem) before pruning the unnecessary dimensions (eigenvectors are discarded). In contrast, most nonspectral methods start by initializing mapped data vectors in a low-dimensional space and then rearrange them to optimize some objective function.

7.7 Tentative methodology

Throughout this book, some examples and applications have demonstrated that the proposed analysis methods efficiently tackle the problems raised by high-dimensional data. This section is an attempt to guide the user through the large variety of NLDR methods described in the literature, according to characteristics of the available data.

A first list of guidelines can be given according to the shape, nature, or properties of the manifold to embed. In the case of ...

- **slightly curved manifolds.** Use a linear method like PCA or metric MDS; alternatively, NLM offers a good tradeoff between robustness and reproducibility and gives the ability to provide a nonlinear embedding.
- **convex developable manifolds.** Use methods relying on geodesic/graph distances (Isomap, GNLM, CDA) or SDE. Conditions to observe convex and developable manifolds in computer vision are discussed in [54].
- **nonconvex developable manifolds.** Do not use Isomap; use GNLM or CDA instead; SDE works well, too.
- **nearly developable manifolds.** Do not use Isomap or SDE; it is better use GNLM or CDA instead.
- **other manifolds.** Use GNLM or preferably CDA. Topology-preserving methods can be used too (LLE, LE, Isotop).
- **manifolds with essential loops.** Use CCA or CDA; these methods are able to tear the manifold, i.e., break the loop. The tearing procedure proposed in [121] can also break essential loops and make data easier to embed with graph-based methods (Isomap, SDE, GNLM, CDA, LLE, LE, Isotop).
- **manifolds with essential spheres.** Use CCA or CDA. The abovementioned tearing procedure is not able to "open" essential spheres.
- **disconnected manifolds.** This remains an open question. Most methods do not explicitly handle this case. The easiest solution is to build an adjacency graph, detect the disconnected components, and embed them separately. Of course, "neighborhood relationships" between the components are lost in the process.
- **clustered data.** In this case the existence of one or several underlying manifolds must be questioned. If the clusters do not have a low intrinsic dimension, the manifold assumption is probably wrong (or useless). Then use clustering algorithms, like spectral clustering or preferably other techniques like hierarchical clustering.

Guidelines can also be given according to the data set's size. In the case of ...

- **Large data set.** If several thousands of data points are available ($N > 2000$), most NLDR methods will generate a heavy computational burden because of their time and space complexities, which are generally proportional to N^2 (or even higher for the computation time). It is then useful to reduce the data set's size, at least to perform some preliminary steps. The easiest way to obtain a smaller data set consists of resampling the available one, i.e., drawing a subset of points at random. Obviously, this is not an optimal way, since it is possible, as ill luck would have it, for the drawn subsample not to be representative of the whole data set. Some examples throughout this book have shown that a representative subset can be determined using vector quantization techniques, like K-means and similar methods.
- **Medium-sized set.** If several hundreds of data points are available ($200 < N \leq 2000$), most NLDR methods can be applied directly to the data set, without any size reduction.
- **Small data set.** When fewer than 200 data points are available, the use of most NLDR methods becomes questionable, as the limited amount of data could be insufficient to identify the large number of parameters involved in many of these methods. Using PCA or classical metric MDS often proves to be a better option.

The dimensionality of data, along with the target dimension, can also be taken into account. In case of a ...

- **very high data dimensionality.** For more than 50 dimensions ($D > 50$), NLDR methods can suffer from the curse of dimensionality, get confused, and provide meaningless results. It can then be wise first to apply PCA or metric MDS in order to perform a hard dimensionality reduction. These two methods can considerably decrease the data dimensionality without losing much information (in terms of measured variance, for instance). Depending on the data set's characteristics, PCA or metric MDS can also help attenuate statistical noise in data. After PCA/MDS, a nonlinear method can be used with more confidence (see the two next cases) in order to further reduce the dimensionality.
- **high data dimensionality.** For a few tens of dimensions ($5 < D \leq 50$), NLDR methods should be used with care. The curse of dimensionality is already no longer negligible.
- **low data dimensionality.** For up to five dimensions, any NLDR method can be applied with full confidence.

Obviously, the choice of the target dimensionality should take into account the intrinsic dimensionality of data if it is known or can be estimated.

- If the target dimensionality is (much) higher than the intrinsic one, PCA or MDS performs very well. These two methods have numerous advantages:

they are simple, fast, do not fall in local optima, and involve no parameters. In this case, even the fact that they transform data in a linear way can be considered an advantage in many respects.
- If the target dimensionality is equal to or hardly higher than the intrinsic one, NLDR methods can yield very good results. Most spectral or nonspectral methods work quite well in this case. For highly curved manifolds, one or two supernumerary dimensions can improve the embedding quality. Most NLDR methods (and especially those based on distance preservation) have limited abilities to deform/distort manifolds. Some extra dimensions can then compensate for this lack of "flexibility." The same strategy can be followed to embed manifolds with essential loops or spheres.
- If the target dimensionality is lower than the intrinsic one, such as for visualization purposes, use NLDR methods at your own risk. It is likely that results will be meaningless since the embedding dimensionality is "forced." In this case, most nonspectral NLDR methods should be avoided. They simply fail to converge in an embedding space of insufficient dimensionality. On the other hand, spectral methods do not share this drawback since they solve an eigenproblem independently from the target dimensionality. This last parameter is involved only in the final selection of eigenvectors. Obviously, although an embedding dimensionality that is deliberately too low does not jeopardize the method convergence, this option does not guarantee that the obtained embedding is meaningful either. Its interpretation and/or subsequent use must be questioned.

Here is a list of additional advices related to the application's purpose and other considerations.

- Collect information about your data set prior to NLDR: estimate the intrinsic dimensionality and compute an adjacency graph in order to deduce the manifold connectivity.
- Never use any NLDR method without knowing the role and influence of all its parameters (true for any method, with a special emphasis on nonspectral methods).
- For 2D visualization and exploratory data analysis, Kohonen's SOM remains a reference tool.
- Never use KPCA for embedding purposes. The theoretical framework hidden behind KPCA is elegant and appealing; it paved the way toward a unified view of all spectral methods. However, in practice, the method lacks a geometrical interpretation that could help the user choose useful kernel functions. Use SDE instead; this method resembles KPCA in many respects, and the SDP step implicitly determines the optimal kernel function for distance preservation.
- Never use SDE with large data sets; this method generates a heavy computational burden and needs to run on much more powerful computers than alternative methods do.

- Avoid using GTM as much as possible; the method involves too many parameters and is restricted to 1D or 2D rectangular latent spaces; the mapping model proves to be not flexible enough to deal with highly curved manifolds.
- LLE is very sensitive to its parameter values (K or ϵ, and the regularization parameter Δ). Use it carefully, and do not hesitate to try different values, as is done in the literature [166].
- Most nonspectral methods can get stuck in local optima: depending on the initialization, different embeddings can be obtained.
- Finally, do not forget to assess the embedding quality using appropriate criteria [186, 185, 9, 74, 10, 190, 103] (see an example in Subsection 6.3.1).

The above recommendations leave the following question unanswered: given a data set, does one choose between distance and topology preservation? If the data set is small, the methods with the simplest models often suit the best (e.g., PCA, MDS, or NLM). With mid-sized data sets, more complex distance-preserving methods like Isomap or CCA/CDA often provide more meaningful results. Topology-preserving methods like LLE, LE, and Isotop should be applied to large data sets only. Actually, the final decision between distance and topology preservation should then be guided by the shape of the underlying manifold. Heavily crumpled manifolds are more easily embedded using topology preservation rather than distance preservation. The key point to know is that both strategies extract neither the same kind nor the same amount of information from data. Topology-preserving methods focus on local information (neighborhood relationships), whereas distance-preserving ones exploit both the local and global manifold structure.

7.8 Perspectives

During the 1900s, dimensionality reduction went through several eras. The first era mainly relied on spectral methods like PCA and then classical metric MDS. Next, the second era consisted of the generalization of MDS into nonlinear variants, many of them being based on distance or rank preservation and among which Sammon's NLM is probably the most emblematic representative. At the end of the century, the field of NLDR was deeply influenced by "neural" approaches; the autoassociative MLP and Kohonen's SOM are the most prominent examples of this stream. The beginning of the new century witnessed the rebirth of spectral approaches, starting with the discovery of KPCA.

So in which directions will the researchers orient their investigations in the coming years ? The paradigm of distance preservation can be counted among the classical NLDR tools, whereas no real breakthrough has happened in topology preservation since the SOM invention. It seems that the vein of spectral methods has now been largely exploited. Many recent papers dealing with that topic do not present new methods but are instead surveys that summarize the domain and explore fundamental aspects of the methods, like their connections or duality within a unifying frameworks. A recent publication in *Science* [89] describing a new training technique for auto-associative MLP could reorient the NLDR research toward artificial neural networks once again, in the same way as the publication of Isomap and LLE in the same journal in 2000 lead to the rapid development of many spectral methods. This renewed interest in ANNs could focus on issues that were barely addressed by spectral methods and distance preservation: large-scale NLDR problems (training samples with several thousands of items), "out-of-sample" generalization, bidirectional mapping, etc.

A last open question regards the curse of dimensionality. An important motivation behind (NL)DR aims at avoiding its harmful effects. Paradoxically, however, many NLDR methods do not bring a complete solution to the problem, but only dodge it. Many NLDR methods give poor results when the intrinsic dimensionality of the underlying manifold exceeds four or five. In such cases, the dimension of the embedding space becomes high enough to observe undesired effects related to the curse of dimensionality, such as the empty space phenomenon. The future will tell whether new techniques will be able to take up this ultimate challenge.

A

Matrix Calculus

A.1 Singular value decomposition

The *singular value decomposition* (SVD) of an M-by-N matrix $\mathbf{A}$ is written as

$$\mathbf{A} = \mathbf{V}\mathbf{\Sigma}\mathbf{U}^T \quad , \tag{A.1}$$

where

- $\mathbf{V}$ is an orthonormal (or unitary) M-by-M matrix such that $\mathbf{V}^T\mathbf{V} = \mathbf{I}_{M\times M}$.
- $\mathbf{\Sigma}$ is a pseudodiagonal matrix with the same size as $\mathbf{A}$; the M entries σ_m on the diagonal are called the singular values of $\mathbf{A}$.
- $\mathbf{U}$ is an orthonormal (or unitary) N-by-N matrix such that $\mathbf{U}^T\mathbf{U} = \mathbf{I}_{N\times N}$.

The number of singular values different from zero gives the rank of $\mathbf{A}$. When the rank is equal to P, the SVD can be used to compute the (pseudo-)inverse of $\mathbf{A}$:

$$\mathbf{A}^+ = \mathbf{U}\mathbf{\Sigma}^+\mathbf{V}^T \quad , \tag{A.2}$$

where the (pseudo-)inverse of $\mathbf{\Sigma}$ is trivially computed by transposing it and inverting its diagonal entries σ_m. The SVD is used in many other contexts and applications. For example, principal component analysis (see Section 2.4) can be carried out using an SVD.

By the way, it is noteworthy that PCA uses a slightly modified SVD. Assuming that $M < N$, $\mathbf{U}$ could become large when $M \ll N$, which often happens in PCA, and $\mathbf{\Sigma}$ contains many useless zeros. This motivates an alternative definition of the SVD, called the economy-size SVD, where only the first P columns of $\mathbf{U}$ are computed. Consequently, $\mathbf{U}^T$ has the same size as $\mathbf{A}$ and $\mathbf{\Sigma}$ becomes a square diagonal matrix. A similar definition is available when $M > N$.

For a square and symmetric matrix, the SVD is equivalent to the eigenvalue decomposition (EVD; see ahead).

A.2 Eigenvalue decomposition

The *eigenvalue decomposition* (EVD) of a square M-by-M matrix $\mathbf{A}$ is written as

$$\mathbf{A}\mathbf{V} = \mathbf{V}\mathbf{\Lambda} \ , \tag{A.3}$$

where

- $\mathbf{V}$ is a square M-by-M matrix whose columns $\mathbf{v}_m$ are unit norm vectors called *eigenvectors* of $\mathbf{A}$.
- $\mathbf{\Lambda}$ is a diagonal M-by-M matrix containing the M eigenvalues λ_m of $\mathbf{A}$.

The EVD is sometimes called the *spectral decomposition* of $\mathbf{A}$. Equation (A.3) translates the fact the eigenvectors keep their direction after left-multiplication by $\mathbf{A}$: $\mathbf{A}\mathbf{v}_m = \lambda_m \mathbf{v}_m$. Moreover, the scaling factor is equal to the associated eigenvalue. The number of eigenvalues different from zero gives the rank of $\mathbf{A}$, and the product of the eigenvalues is equal to the determinant of $\mathbf{A}$. On the other hand, the *trace* of $\mathbf{A}$, denoted $\operatorname{tr}(\mathbf{A})$ and defined as the sum of its diagonal entries, is equal to the sum of its eigenvalues:

$$\operatorname{tr}(\mathbf{A}) \triangleq \sum_{m=1}^{M} a_{m,m} = \sum_{m=1}^{M} \lambda_m \ . \tag{A.4}$$

In the general case, even if $\mathbf{A}$ contains only real entries, $\mathbf{V}$ and $\mathbf{\Lambda}$ can be complex. If $\mathbf{A}$ is symmetric ($\mathbf{A} = \mathbf{A}^T$), then $\mathbf{V}$ is orthonormal (the eigenvectors are orthogonal in addition to being normed); the EVD can then be rewritten as

$$\mathbf{A} = \mathbf{V}\mathbf{\Lambda}\mathbf{V}^T \ , \tag{A.5}$$

and the eigenvalues are all real numbers. Moreover, if $\mathbf{A}$ is positive definite (resp., negative definite), then all eigenvalues are positive (resp., negative). If $\mathbf{A}$ is positive semidefinite (resp., negative semidefinite), then all eigenvalues are nonnegative (resp., nonpositive). For instance, a covariance matrix is positive semidefinite.

A.3 Square root of a square matrix

The square root of a diagonal matrix is easily computed by applying the square root solely on the diagonal entries. By comparison, the square root of a nondiagonal square matrix may seem more difficult to compute. First of all, there are two different ways to define the square root of a square matrix $\mathbf{A}$.

The first definition assumes that

$$\mathbf{A} \triangleq (\mathbf{A}^{1/2})^T(\mathbf{A}^{1/2}) \ . \tag{A.6}$$

If $\mathbf{A}$ is symmetric, then the eigenvalue decomposition (EVD) of the matrix helps to return to the diagonal case. The eigenvalue decomposition (see Appendix A.2) of any symmetric matrix $\mathbf{A}$ is

$$\mathbf{A} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^T = (\mathbf{V}\boldsymbol{\Lambda}^{1/2})(\boldsymbol{\Lambda}^{1/2}\mathbf{V}^T) = (\mathbf{A}^{1/2})^T(\mathbf{A}^{1/2}) \ , \tag{A.7}$$

where $\boldsymbol{\Lambda}$ is diagonal. If $\mathbf{A}$ is also positive definite, then all eigenvalues are positive and the diagonal entries of $\boldsymbol{\Lambda}^{1/2}$ remain positive real numbers. (If $\mathbf{A}$ is only positive semidefinite, then the square root is no longer unique.)

The second and more general definition of the square root is written as

$$\mathbf{A} \triangleq (\mathbf{A}^{1/2})(\mathbf{A}^{1/2}) \ . \tag{A.8}$$

Again, the eigenvalue decomposition leads to the solution. The square root is then written as

$$\mathbf{A}^{1/2} = \mathbf{V}\boldsymbol{\Lambda}^{1/2}\mathbf{V}^{-1} \ , \tag{A.9}$$

and it is easy to check that

$$\begin{aligned} \mathbf{A}^{1/2}\mathbf{A}^{1/2} &= \mathbf{V}\boldsymbol{\Lambda}^{1/2}\mathbf{V}^{-1}\mathbf{V}\boldsymbol{\Lambda}^{1/2}\mathbf{V}^{-1} && \text{(A.10)} \\ &= \mathbf{V}\boldsymbol{\Lambda}^{1/2}\boldsymbol{\Lambda}^{1/2}\mathbf{V}^{-1} && \text{(A.11)} \\ &= \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^{-1} = \mathbf{A} \ . && \text{(A.12)} \end{aligned}$$

This is valid in the general case, i.e., $\mathbf{A}$ can be complex and/or nonsymmetric, yielding complex eigenvalues and eigenvectors. If $\mathbf{A}$ is symmetric, the last equation can be further simplified since the eigenvectors are real and orthonormal ($\mathbf{V}^{-1} = \mathbf{V}^T$).

It is notenorthy that the second definition of the matrix square root can be generalized to compute matrix powers:

$$\mathbf{A}^p = \mathbf{V}\boldsymbol{\Lambda}^p\mathbf{V}^{-1} \ . \tag{A.13}$$

B

Gaussian Variables

This appendix briefly introduces Gaussian random variables and some of their basic properties.

B.1 One-dimensional Gaussian distribution

Considering a single one-dimensional random variable x, it is said to be Gaussian if its probability density function $f_x(x)$ can be written as

$$f_x(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right) \quad , \tag{B.1}$$

where μ and σ^2 are the *mean* and the *variance*, respectively, and correspond to the first-order moment and second-order central moment. Figure B.1 shows a plot of a Gaussian probability density function. Visually, the mean μ indicates

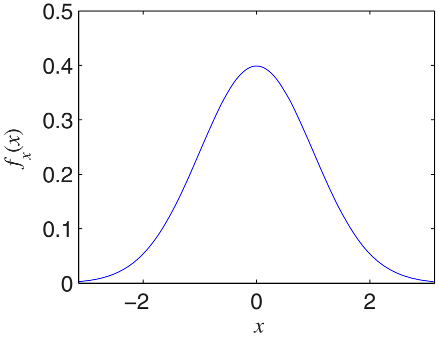


Fig. B.1. Probability density function associated with a one-dimensional Gaussian distribution.

the abscissa where the bump reaches its highest point, whereas σ is related to the spreading of the bump. For this reason, the standard deviation σ is often called the *width* in a geometrical context (see ahead).

Since only the mean and variance suffice to characterize a Gaussian variable, it is often denoted as $N(\mu, \sigma)$. The letter N recalls the alternative name of a Gaussian variable: a *normal* variable or *normally* distributed variable. This name simply reflects the fact that for real-valued variables, the Gaussian distribution is the most widely observed one in a great variety of phenomena.

Moreover, the central limit theorem states that a variable obtained as the sum of several independent identically distributed variables, regardless of their distribution, tends to be Gaussian if the number of terms in the sum tends to infinity. Thus, to some extent, the Gaussian distribution can be considered the “child” of all other distributions. On the other hand, the Gaussian distribution can also be interpreted as the “mother” of all other distributions. This is intuitively confirmed by the fact that any zero-mean unit variance pdf $f_y(y)$ can be modeled starting from a zero-mean and unit-variance Gaussian variable with pdf $f_x(x)$, by means of the Gram–Charlier or Edgeworth development:

$$f_y(y) = f_x(y) \left(1 + \frac{1}{6}\mu_3(y) H_3(y) + \frac{1}{24}(\mu_4(y) - 3) H_4(y) + \ldots\right) \quad , \tag{B.2}$$

where $H_i(y)$ is the ith-order Hermitte–Chebyshev polynomial and $\mu_i(y)$ the ith-order central moment. If the last equation is rewritten using the ith-order cumulants $\kappa_i(y)$ instead of central moments, one gets

$$f_y(y) = f_x(y) \left(1 + \frac{1}{6}\kappa_3(y) H_3(y) + \frac{1}{24}\kappa_4(y) H_4(y) + \ldots\right) \quad . \tag{B.3}$$

Visually, in the last development, a nonzero *skewness* $\kappa_3(y)$ makes the pdf $f_y(y)$ asymmetric, whereas a nonzero *kurtosis excess* $\kappa_4(y)$ makes its bump flatter or sharper. Even if the development does not go beyond the fourth order, it is easy to guess that the Gaussian distribution is the only one having zero cumulants for orders higher than two. This partly explains why Gaussian variables are said to be the “least interesting” ones in some contexts [95]. Actually, a Gaussian distribution has absolutely no salient characteristic:

- The support is unbounded, in contrast to a uniform distribution, for instance.
- The pdf is smooth, symmetric, and unimodal, without a sharp peak like the pdf of a Laplacian distribution.
- The distribution maximizes the differential entropy.

The function defined in Eq. (B.1) and plotted in Fig. B.1 is the sole function that both shows the above properties and respects the necessary conditions to be a probability density function. These conditions are set on the cumulative density function $F_x(x)$ of the random variable, defined as

$$F_x(x) = \int_{-\infty}^{x} f_x(u)du \ , \tag{B.4}$$

and requires that

- $F_x(-\infty) = 0$,
- $F_x(+\infty) = 1$,
- $F_x(x)$ is monotonic and nondecreasing,
- $F_x(b) - F_x(a)$ is the probability that $a < x \leq b$.

B.2 Multidimensional Gaussian distribution

A P-dimensional random variable $\mathbf{x}$ is jointly Gaussian if its joint probability density function can be written as

$$f_{\mathbf{x}}(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^P \det \mathbf{C}_{\mathbf{xx}}}} \exp\left(-\frac{1}{2}(\mathbf{x} - \mu_{\mathbf{x}})^T \mathbf{C}_{\mathbf{xx}}^{-1} (\mathbf{x} - \mu_{\mathbf{x}})\right) \ , \tag{B.5}$$

where $\mu_{\mathbf{x}}$ and $\mathbf{C}_{\mathbf{xx}}$ are, respectively, the mean vector and the covariance matrix. As the covariance matrix is symmetric and positive semidefinite, its determinant is nonnegative. The joint pdf of a two-dimensional Gaussian is drawn in Fig. B.2.

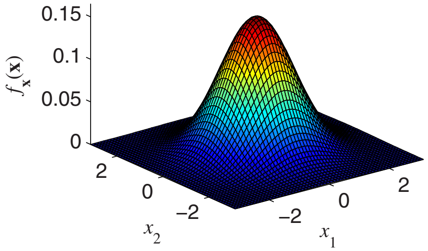


Fig. B.2. Probability density function associated with a two-dimensional Gaussian distribution.

It can be seen that in the argument of the exponential function, the factor $(\mathbf{x} - \mu_{\mathbf{x}})^T \mathbf{C}_{\mathbf{xx}}^{-1} (\mathbf{x} - \mu_{\mathbf{x}})$ is related to the square of the Mahalanobis distance between $\mathbf{x}$ and $\mu_{\mathbf{x}}$ (see Subsection 4.2.1).

B.2.1 Uncorrelated Gaussian variables

If a multidimensional Gaussian distribution is *uncorrelated*, i.e., if its covariance matrix is diagonal, then the last equation can be rewritten as

$$f_{\mathbf{x}}(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^P \prod_{p=1}^P c_{p,p}}} \exp\left(-\frac{1}{2}\sum_{p=1}^P \frac{(x_p-\mu_{x_p})^2}{c_{p,p}^2}\right) \tag{B.6}$$

$$= \prod_{p=1}^P \frac{1}{\sqrt{2\pi c_{p,p}}} \exp\left(-\frac{1}{2}\frac{(x_p-\mu_{x_p})^2}{c_{p,p}}\right) \tag{B.7}$$

$$= \prod_{p=1}^P \frac{1}{\sqrt{2\pi \sigma_{x_p}}} \exp\left(-\frac{1}{2}\frac{(x_p-\mu_{x_p})^2}{\sigma_{x_p}^2}\right) \tag{B.8}$$

$$= \prod_{p=1}^P f_{x_p}(x_p) \ , \tag{B.9}$$

showing that the joint pdf of uncorrelated Gaussian variables factors into the product of the marginal probability density functions. In other words, uncorrelated Gaussian variables are also statistically independent. Again, $f_{\mathbf{x}}(\mathbf{x})$ is the sole and unique probability density function that can satisfy this property. For other multivariate distributions, the fact of being uncorrelated does *not* imply the independence of the marginal densities. Nevertheless, the reverse implication is always true.

Geometrically, a multidimensional Gaussian distribution looks like a fuzzy ellipsoid, as shown in Fig. B.3. The axes of the ellipsoid correspond to coordinate axes.

B.2.2 Isotropic multivariate Gaussian distribution

A multidimensional Gaussian distribution is said to be *isotropic* if its covariance matrix can be written as a function of the single parameter σ:

$$\mathbf{C}_{\mathbf{xx}} = \sigma^2 \mathbf{I} \ . \tag{B.10}$$

As $\mathbf{C}_{\mathbf{xx}}$ is diagonal, the random variables in $\mathbf{x}$ are uncorrelated and independent, and σ is the common standard deviation shared by all marginal probability density functions $f_{x_p}(x_p)$. By comparison to the general case, the pdf of an isotropic Gaussian distribution can be rewritten more simply as

$$f_{\mathbf{x}}(\mathbf{x}) = \frac{1}{\sqrt{(2\pi)^P \det(\sigma^2 \mathbf{I})}} \exp\left(-\frac{1}{2}(\mathbf{x}-\mu_{\mathbf{x}})^T(\sigma^2\mathbf{I})^{-1}(\mathbf{x}-\mu_{\mathbf{x}})\right) \tag{B.11}$$

$$= \frac{1}{\sqrt{(2\pi\sigma^2)^P}} \exp\left(-\frac{1}{2}\frac{(\mathbf{x}-\mu_{\mathbf{x}})^T(\mathbf{x}-\mu_{\mathbf{x}})}{\sigma^2}\right) \tag{B.12}$$

$$= \frac{1}{\sqrt{(2\pi\sigma^2)^P}} \exp\left(-\frac{1}{2}\frac{\|\mathbf{x}-\mu_{\mathbf{x}}\|_2^2}{\sigma^2}\right) \ , \tag{B.13}$$

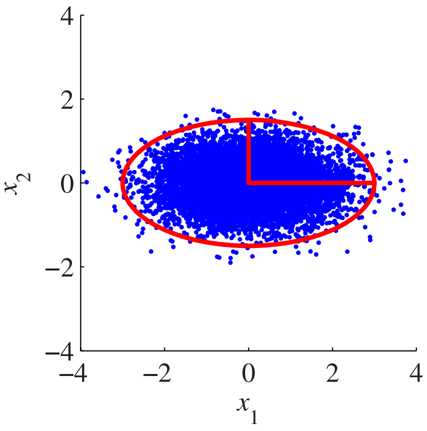


Fig. B.3. Sample joint distribution (10,000 realizations, in blue) of two uncorrelated Gaussian variables. The variances are proportional to the axis lengths (in red).

The appearance of the Euclidean distance (see Subsection 4.2.1) demonstrates that the value of the pdf depends only on the distance to the mean and not on the orientation. Consequently, $f_{\mathbf{x}}(\mathbf{x})$ is completely isotropic: no direction is privileged, as illustrated in Fig. B.4. For this reason, the standard deviation

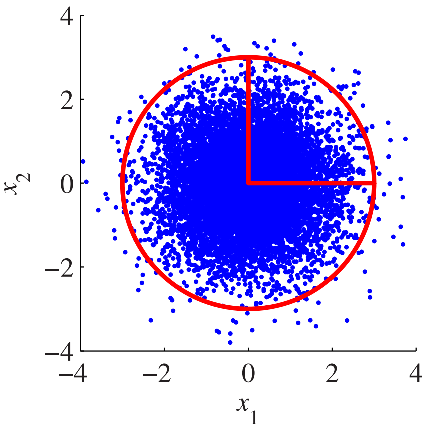


Fig. B.4. Sample (10,000 realizations) of a two-dimensional isotropic Gaussian distribution. The variances are proportional to the axis lengths (in red), which are equal. The variance is actually the same in all directions.

σ is often called the *width* or *radius* of the Gaussian distribution.

The function $f_{\mathbf{x}}(\mathbf{x})$ is often used outside the statistical framework, possibly without its normalization factor. In this case, $f_{\mathbf{x}}(\mathbf{x})$ is usually called a *radial basis function* or *Gaussian kernel*. In addition to be isotropic, such a function has very nice properties:

- It produces a single localized bump.
- Very few parameters have to be set (P means and one single variance, compared to $P(P+1)/2$ for a complete covariance matrix).
- It depends on the well-known and widely used Euclidean distance.

Gaussian kernels are omnipresent in applications like radial basis function networks [93] (RBFNs) and support vector machines (SVM) [27, 37, 42].

B.2.3 Linearly mixed Gaussian variables

What happens when several Gausian variables are linearly mixed ?

To answer the question, one assumes that the P-dimensional random vector $\mathbf{x}$ has a joint Gaussian distribution with zero means and identity covariance, without loss of generality. Then P linearly mixed variables can be written as $\mathbf{y} = \mathbf{A}\mathbf{x}$, where $\mathbf{A}$ is a square P-by-P *mixing matrix*. By "mixing matrix", it should be understood that $\mathbf{A}$ is of full column rank and has at least two nonzero entries per row. These conditions ensure that the initial random variables are well mixed and not simply copied or scaled. It is noteworthy that $\mathbf{y}$ has zero means like $\mathbf{x}$.

As matrix $\mathbf{A}$ defines a nonorthogonal change of the axes, the joint pdf of $\mathbf{y}$ can be written as

$$f_{\mathbf{y}}(\mathbf{y}) = \frac{1}{\sqrt{(2\pi)^P \det \mathbf{C}_{\mathbf{yy}}}} \exp\left(-\frac{1}{2}\mathbf{y}^T \mathbf{I}^{-1} \mathbf{y}\right) \tag{B.14}$$

$$= \frac{1}{\sqrt{(2\pi)^P \det \mathbf{C}_{\mathbf{yy}}}} \exp\left(-\frac{1}{2}(\mathbf{A}\mathbf{x})^T (\mathbf{A}\mathbf{x})\right) \tag{B.15}$$

$$= \frac{1}{\sqrt{(2\pi)^P \det \mathbf{C}_{\mathbf{yy}}}} \exp\left(-\frac{1}{2}\mathbf{x}^T (\mathbf{A}^T \mathbf{A})\, \mathbf{x}\right) \quad , \tag{B.16}$$

demonstrating that $\mathbf{C}_{\mathbf{yy}} = (\mathbf{A}^T\mathbf{A})^{-1}$. Unfortunately, as the covariance matrix is symmetric, it has only $P(P+1)/2$ free parameters, which is less than P^2, the number of entries in the mixing matrix $\mathbf{A}$. Consequently, starting from the mixed variables, it is impossible to retrieve the mixing matrix. A possible solution would have been to compute the matrix square root of $\mathbf{C}_{\mathbf{yy}}^{-1}$ (see Subsection A.3). However, this provides only a least-squares estimate of $\mathbf{A}$, on the basis of the incomplete information available in the covariance matrix. Geometrically, one sees in Fig. B.5 that the matrix square root always finds an orthogonal coordinate system, computed as the eigenvetors of $\mathbf{C}_{\mathbf{yy}}^{-1}$. This orthogonal system is shown in red, whereas the original coordinate system deformed by the mixing matrix is drawn in green. This explains why PCA

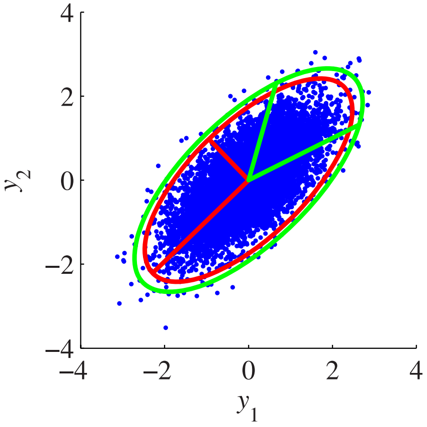


Fig. B.5. Sample joint distribution (10,000 realizations) of two isotropic Gaussian variables. The original coordinate system, in green, has been deformed by the mixing matrix. Any attempt to retrieve it leads to the orthogonal system shown in red.

(and ICA) is unable to separate mixtures of Gaussian variables: with two or more Gaussian variables, indeterminacies appear.

C

Optimization

Most nonspectral NLDR methods rely on the optimization of some objective function. This appendix describes a few classical optimization techniques.

C.1 Newton's method

The original Newton's method, also known as Newton–Raphson method, is an iterative procedure that finds a zero of a $\mathcal{C}_\infty$ function (infinitely differentiable function)

$$f : \mathbb{R} \to \mathbb{R} : x \mapsto f(x) \ . \tag{C.1}$$

Basically, Newton's method approximates the function f by its first-order Taylor's polynomial expansion:

$$f(x+\epsilon) = f(x) + f'(x)\epsilon + \mathcal{O}(\epsilon^2) \ . \tag{C.2}$$

Defining $x_{\text{old}} = x$ and $x_{\text{new}} = x + \epsilon$, omitting $\mathcal{O}(\epsilon^2)$, and assuming that $f(x_{\text{new}}) = 0$, one gets:

$$0 \approx f(x_{\text{old}}) + f'(x_{\text{old}})(x_{\text{new}} - x_{\text{old}}) \ . \tag{C.3}$$

Solving for x_{new} leads to

$$x_{\text{new}} \approx x_{\text{old}} - \frac{f(x_{\text{old}})}{f'(x_{\text{old}}} \ , \tag{C.4}$$

which can be rewritten as an iterative update rule:

$$x \leftarrow x - \frac{f(x)}{f'(x)} \ . \tag{C.5}$$

Intuitively, starting from a candidate solution x that is randomly initialized, the next candidate is the intersection of a tangent to $f(x)$ with the x-axis. It can be proven that the first-order approximation makes the convergence of the procedure very fast (quadratic convergence): after a few iterations, the solution remains almost constant, and the procedure may be stopped. However, it is easy to see that the method becomes unstable when $f'(x) \approx 0$.

C.1.1 Finding extrema

Recalling that function extremas are such that $f'(x) = 0$, a straight extension of Newton's procedure can be applied to find a local extremum of a twicely derivable function f:

$$x \leftarrow x - \frac{f'(x)}{f''(x)} \ , \tag{C.6}$$

where the first and second derivatives are assumed to be continuous. The last update rule, unfortunately, does not distinguish between a minimum and a maximum and yields either one of them. An extremum is a minimum only if the second derivative is positive, i.e., the function is concave in the neighborhood of the extremum. In order to avoid the convergence toward a maximum, a simple trick consists of forcing the second derivative to be positive:

$$x \leftarrow x - \frac{f'(x)}{|f''(x)|} \ . \tag{C.7}$$

C.1.2 Multivariate version

The generalization of Newton's optimization procedure (Eq. (C.6)) to multivariate functions $f(\mathbf{x}) : \mathbb{R}^P \to \mathbb{R} : \mathbf{x} \mapsto f(\mathbf{x})$ leads to

$$\mathbf{x} \leftarrow \mathbf{x} - \mathbf{H}^{-1} \nabla_{\mathbf{x}} f(\mathbf{x}) \ , \tag{C.8}$$

where $\nabla_{\mathbf{x}}$ is the differential operator with respect to the components of vector $\mathbf{x} = [x_1, \ldots, x_P]^T$:

$$\nabla_{\mathbf{x}} \triangleq \left[\frac{\partial}{\partial x_1}, \ldots, \frac{\partial}{\partial x_P} \right]^T \ , \tag{C.9}$$

and $\nabla_{\mathbf{x}} f(\mathbf{x})$ is the gradient of f, i.e., the vector of all partial derivatives:

$$\nabla_{\mathbf{x}} f(\mathbf{x}) = \frac{\partial f(\mathbf{x})}{\partial \mathbf{x}} = \left[\frac{\partial f(\mathbf{x})}{\partial x_1}, \ldots, \frac{\partial f(\mathbf{x})}{\partial x_P} \right]^T \ . \tag{C.10}$$

One step further, $\mathbf{H}^{-1}$ is the inverse of the Hessian matrix, defined as

$$\mathbf{H} \triangleq \nabla_{\mathbf{x}} \nabla_{\mathbf{x}}^T f(\mathbf{x}) = \left[\frac{\partial^2 f(\mathbf{x})}{\partial x_i \partial x_j} \right]_{ij} \ , \tag{C.11}$$

whose entries $h_{i,j}$ are the second-order partial derivatives.

Unfortunately, the application of Eq. (C.8) raises two practical difficulties. First, the trick of the absolute value is not easily generalized to multivariate functions, which can have minima, maxima, but also various kinds of saddle points. Second, the Hessian matrix is rarely available in practice. Moreover, its size grows proportionally to P^2, making its computation and storage problematic.

A solution to these two issues consists of assuming that the Hessian matrix is diagonal, although it is often a very crude hypothesis. This approximation, usually called quasi-Newton or diagonal Newton, can be written componentwise:

$$x_p \leftarrow x_p - \alpha \frac{\frac{\partial f(\mathbf{x})}{\partial x_p}}{\left|\frac{\partial^2 f(\mathbf{x})}{\partial x_p^2}\right|} \ , \tag{C.12}$$

where the coefficient α $(0 < \alpha \leq 1)$ slows down the update rule in order to avoid unstable behaviors due to the crude approximation of the Hessian matrix.

C.2 Gradient ascent/descent

The *gradient descent* (resp., ascent) is a generic minimization (resp., maximization) technique also known as the *steepest descent* (ascent) method. As Newton's method (see the previous section), the gradient descent is an iterative technique that finds the closest extremum starting from an initial guess. The method requires knowing only the gradient of the function to be optimized in closed form.

Actually, the gradient descent can be seen as a simplified version of Newton's method, in the case where the Hessian matrix is unknown. Hence, the gradient descent is a still rougher approximation than the pseudo-Newton method. More formally, the inverse $\mathbf{H}^{-1}$ of the unknown Hessian is simply replaced with the product $\alpha\mathbf{I}$, where α is a parameter usually called the *step size* or the *learning rate*. Thus, for a multivariate function $f(\mathbf{x})$, the iterative update rule can be written as

$$\mathbf{x} \leftarrow \mathbf{x} - \alpha \nabla_{\mathbf{x}} f(\mathbf{x}) \ . \tag{C.13}$$

As the Hessian is unknown, the local curvature of the function is unknown, and the choice of the step size may be critical. A value that is too large may jeopardize the convergence on an extremum, but, on the other hand, the convergence becomes very slow for small values.

C.2.1 Stochastic gradient descent

Within the framework of data analysis, it often happens that the function to be optimized is of the form

$$f(\mathbf{x}) = E_{\mathbf{y}}\{g(\mathbf{y}, \mathbf{x})\} \qquad \text{or} \qquad f(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^{N} g(\mathbf{y}(i), \mathbf{x}) \ , \tag{C.14}$$

where $\mathbf{y}$ is an observed variable and $\mathbf{Y} = [\ldots, \mathbf{y}(i), \ldots]_{1 \leq i \leq N}$ is an array of observations. Then the update rule for the gradient descent with respect to the unknown variables/parameters $\mathbf{x}$ can be written as

$$\mathbf{x} \leftarrow \mathbf{x} - \alpha \nabla_{\mathbf{x}} \frac{1}{N} \sum_{i=1}^{N} g(\mathbf{y}(i), \mathbf{x})\} \ . \tag{C.15}$$

This is the usual update rule for the classical gradient descent. In the framework of neural networks and other adaptive methods, the classical gradient descent is often replaced with the *stochastic gradient descent*. In the latter method, the update rule can be written in the same way as in the classical method, except that the mean (or expectation) operator disappears:

$$\mathbf{x} \leftarrow \mathbf{x} - \alpha \nabla_{\mathbf{x}} g(\mathbf{y}(i), \mathbf{x}) \ . \tag{C.16}$$

Because of the dangling index i, the update rule must be repeated N times, over all available observations $\mathbf{y}(i)$. From an algorithmic point of view, this means that two loops are needed. The first one corresponds to the iterations that are already performed in the classical gradient descent, whereas an inner loop traverses all vectors of the data set. A traversal of the data set is usually called an *epoch*.

Moreover, from a theoretical point of view, as the partial updates are no longer weighted and averaged, additional conditions must be fulfilled in order to attain convergence. Actually, the learning rate α must decrease as epochs go by and, assuming t is an index over the epochs, the following (in)equalities must hold [156]:

$$\sum_{t=1}^{\infty} \alpha(t) = \infty \qquad \text{and} \qquad \sum_{t=1}^{\infty} (\alpha(t))^2 < \infty \ . \tag{C.17}$$

Additionally, it is often advised to consider the available observations as an unordered set, i.e., to randomize the order of the updates at each epoch. This allows us to avoid any undesired bias due to a repeated order of the updates.

When a stochastic gradient descent is used in a truly adaptive context, i.e., the sequence of observations $\mathbf{y}(i)$ is infinite, then the procedure cannot be divided into successive epochs: there is only a single, very long epoch. In that case the learning rate is set to a small constant value (maybe after a short decrease starting from a larger value in order to initialize the process). Conditions to attain convergence are difficult to study in that case, but they are generally fulfilled if the sequence of observations is stationary.

D

Vector quantization

Like dimensionality reduction, vector quantization [77, 73] is a way to reduce the size of a data set. However, instead of lowering the *dimensionality* of the observations, vector quantization reduces the *number* of observations (see Fig. D.1). Therefore, vector quantization and dimensionality reduction are somewhat complementary techniques. By the way, it is noteworthy that several DR methods use vector quantization as preprocessing.

In practice, vector quantization is achieved by replacing the original data points with a smaller set of points called *units*, *centroids*, *prototypes* or *code vectors*. The ordered or indexed set of code vectors is sometimes called the *codebook*.

Intuitively, a good quantization must show several qualities. Classically, the user expects that the prototypes are *representative* of the original data they replace (see Fig. D.1). More formally, this goal is reached if the probability density function of the prototypes resembles the probability density function of the initial data. However, as probability density functions are difficult to estimate, especially when working with a finite set of data points, this idea will not work as such. An alternative way to capture approximately the original density consists of minimizing the quantization distortion. For a data set $\mathcal{Y} = \{\mathbf{y}(i) \in \mathbb{R}^D\}_{1 \leq i \leq N}$ and a codebook $\mathcal{C} = \{\mathbf{c}(j) \in \mathbb{R}^D\}_{1 \leq j \leq M}$, the distortion is a quadratic error function written as

$$E_{\mathrm{VQ}} = \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{y}(i) - \mathrm{dec}(\mathrm{cod}(\mathbf{y}(i)))\|^2 \ , \tag{D.1}$$

where the coding and decoding functions cod and dec are respectively defined as

$$\mathrm{cod} : \mathbb{R}^D \to \{1, \ldots, M\} : \mathbf{y} \mapsto \arg \min_{1 \leq j \leq M} \|\mathbf{y} - \mathbf{c}(j)\| \tag{D.2}$$

and

$$\mathrm{dec} : \{1, \ldots, M\} \to \mathcal{C} : j \mapsto \mathbf{c}(j) \ . \tag{D.3}$$

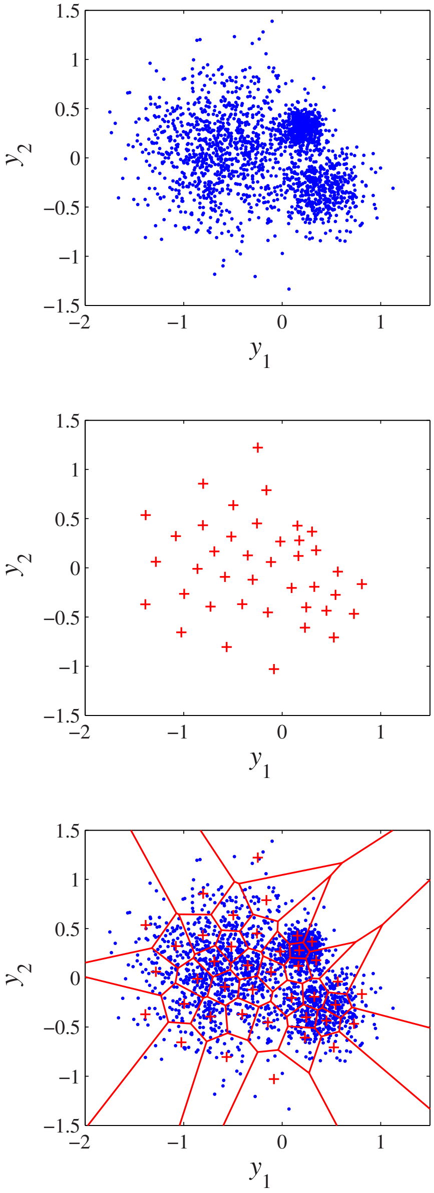


Fig. D.1. Principle of vector quantization. The first plot shows a data set (2000 points). As illustrated by the second plot, a vector quantization method can reduce the number of points by replacing the initial data set with a smaller set of representative points: the *prototypes*, *centroids*, or *code vectors*, which are stored in the codebook. The third plot shows simultaneously the initial data, the prototypes, and the boundaries of the corresponding Voronoï regions.

The application of the coding function to some vector $\mathbf{y}(i)$ of the data set gives the index j of the *best-matching unit* of $\mathbf{y}(i)$ (BMU in short), i.e., the closest prototype from $\mathbf{y}(i)$. Appendix F.2 explains how to compute the BMU efficiently. The application of the decoding function to j simply gives the coordinates $\mathbf{c}(j)$ of the corresponding prototype. The coding function induces a partition of $\mathbb{R}^D$: the open sets of all points in $\mathbb{R}^D$ that share the same BMU $\mathbf{c}(j)$ are called the *Voronoï regions* (see Fig. D.1). A discrete approximation of the Voronoï regions can be obtained by constituting the sets V_j of all data points $\mathbf{y}(i)$ having the same BMU $\mathbf{c}(j)$. Formally, these sets are written as

$$V_j = \{\mathbf{y}(i) | \mathrm{cod}(\mathbf{y}(i)) = j\} \tag{D.4}$$

and yield a partition of the data set.

D.1 Classical techniques

Numerous techniques exist to minimize E_{VQ}. The most prominent one is the LBG algorithm [124], which is basically an extension of the generalized Lloyd method [127]. Close relatives coming from the domain of cluster analysis are the ISODATA [8] and the K-means [61, 130] algorithms. In the case of the K-means, the codebook is built as follows. After the initialization (see ahead), the following two steps are repeated until the distortion decreases below a threshold fixed by the user:

1. Encode each point of the data set, and compute the discrete Voronoï regions V_j;
2. Update each $\mathbf{c}(j)$ by moving it to the barycenter of V_j, i.e.,

$$\mathbf{c}(j) \leftarrow \frac{1}{|V_j|} \sum_{\mathbf{y}(i) \in V_j} \mathbf{y}(i) \ . \tag{D.5}$$

This procedure monotonically decreases the quantization distortion until it reaches a local minimum. To prove the correctness of the procedure, one rewrites E_{VQ} as a sum of contributions coming from each discrete Voronoï region:

$$E_{\mathrm{VQ}} = \frac{1}{N} \sum_{j=1}^{M} E_{\mathrm{VQ}}^j \ , \tag{D.6}$$

where

$$E_{\mathrm{VQ}}^j = \sum_{\mathbf{y}(i) \in V_j} \|\mathbf{y}(i) - \mathbf{c}(j)\|^2 \ . \tag{D.7}$$

Trivially, the barycenter of some V_j minimizes the corresponding E_{VQ}^j. Therefore, step 2 decreases E_{VQ}. But, as a side effect, the update of the prototypes also modifies the results of the encoding function. So it must be shown that the re-encoding occurring in step 1 also decreases the distortion. The only terms that change in the quantization distortion defined in Eq. (D.1) are those corresponding to data points that change their BMU. By definition of the coding function, the distance $\|\mathbf{y}(i) - \mathbf{c}(j)\|$ is smaller for the new BMU than for the old one. Therefore, the error is lowered after re-encoding, which concludes the correctness proof of the K-means.

D.2 Competitive learning

Falling in local minima can be avoided to some extent by minimizing the quantization distortion by stochastic gradient descent (Robbins–Monro procedure [156]). For this purpose, the gradient of E_{VQ} with respect to prototype $\mathbf{c}(j)$ is written as

$$\nabla_{\mathbf{c}(j)} E_{\mathrm{VQ}} = \frac{1}{N}\sum_{i=1}^{N} 2\|\mathbf{y}(i) - \mathbf{c}(j)\| \frac{\partial\|\mathbf{y}(i) - \mathbf{c}(j)\|}{\partial \mathbf{c}(j)} \tag{D.8}$$

$$= \frac{1}{N}\sum_{i=1}^{N} 2\|\mathbf{y}(i) - \mathbf{c}(j)\| \frac{(\mathbf{c}(j) - \mathbf{y}(i))}{2\|\mathbf{y}(i) - \mathbf{c}(j)\|} \tag{D.9}$$

$$= \frac{1}{N}\sum_{i=1}^{N} (\mathbf{c}(j) - \mathbf{y}(i)) \ , \tag{D.10}$$

where the equality $j = \mathrm{cod}(\mathbf{y}(i))$ holds implicitly. In a classical gradient descent, all prototypes are updated simultaneously after the computation of their corresponding gradient. Instead, the Robbins–Monro procedure [156] (or stochastic gradient descent) separates the terms of the gradient and updates immediately the prototypes according to the simple rule:

$$\mathbf{c}(j) \leftarrow \mathbf{c}(j) - \alpha(\mathbf{c}(j) - \mathbf{x}_i) \ , \tag{D.11}$$

where α is a learning rate that decreases after each *epoch*, i.e., each sweeping of the data set. In fact, the decrease of the learning rate α must fulfill the conditions stated in [156] (see Section C.2).

Unlike the K-means algorithm, the stochastic gradient descent does not decrease the quantization distortion monotonically. Actually, the decrease occurs only on average. The stochastic gradient descent of E_{VQ} belongs to a wide class of vector quantization algorithms known as competitive learning methods [162, 163, 3].

D.3 Taxonomy

Quantization methods may be divided into *static*, *incremental*, and *dynamic* ones. This distinction refers to their capacity to increase or decrease the number of prototypes they update. Most methods, like competitive learning and

LBG, are static and manage a number of prototypes fixed in advance. Incremental methods (see, for instance, [68, 71, 11, 70]) are able to increase this predetermined number by inserting supplemental units when this is necessary (various criterions exist). Fully dynamic methods (see, for instance, [69]) can add new units and remove unnecessary ones.

In addition to the distinction between classical quantization and competitive learning, the latter can further be divided into two subcategories:

- **Winner take all (WTA).** Similarly to the stochastic method sketched just above, WTA methods update only one prototype (the BMU) at each presentation of a datum. WTA methods are the simplest ones and include the classical competitive learning [162, 163] and the frequency-sensitive competitive learning [51].
- **Winner take most (WTM).** WTM methods are more complex than WTA ones, because the prototypes interact at each presentation of a datum. In practice, *several* prototypes are updated at each presentation of a datum. In addition to the BMU, some other prototypes related to the BMU are also updated. Depending on the specific quantization method, these prototypes may be the second, third, and so forth closest prototypes in the data space, as in the neural gas [135] (NG). Otherwise, the neighborhood relationships with the BMU may also be predefined and data-independent, as in a self-organizing map [105, 154] (SOM; see also Subsection 5.2.1).

D.4 Initialization and "dead units"

For K-means as well as for competitive learning, the initialization of the codebook is very important. The worst case appears when the prototypes are initialized (far) outside the region of the space occupied by the data set. In this case, there is no guarantee that the algorithm will ever move these prototypes. Consequently, not only these prototypes are lost (they are often called *dead units*), but they may also mislead the user, since these dead units are obviously *not* representative of the original data. A good initialization consists of copying into the codebook the coordinates of m points chosen randomly in the data set.

The appearance of dead units may be circumvented in several ways. The most obvious way to deal with them consists of:

1. computing the cardinality of the Voronoï regions after quantization,
2. discarding the units associated with an empty region.

In order to keep the number of prototypes constant, lost units can be detected during convergence after each epoch of the Robbins–Monro procedure. But instead of removing them definitely as above, they may be reinitialized and reinserted, according to what is proposed in the previous paragraph. Such a recycling trick may be seen as a degenerate dynamic method that adds and removes units but always keeps the same total number of prototypes.

Outside the scope of its application to nonlinear dimensionality reduction, vector quantization belongs to classical techniques used, for example, in:

- statistical data analysis, mainly for the clustering of multivariate data; in that framework, the centroids and/or their discrete Voronoï regions are called *clusters*,
- telecommunications, for lossy data compression.

E

Graph Building

In the field of data analysis, graph building is an essential step in order to capture the neighborhood relationships between the points of a given data set. The resulting graph may also be used to obtain good approximations to geodesic distances with graph distances (see Subsection 4.3.1). The construction of graphs is also useful in many other domains, such as finite-element methods.

Briefly put, graph building in the context of NLDR aims at connecting neighboring points of the space. The set of available points is often finite and given as a matrix of coordinates. Obviously, the fact to be or not to be a neighbor must be defined in some way but essentially depends on the user's needs or preferences.

Several rules inspired by intuitive ideas allow building graphs starting from a set of points in space. Each rule shows specific properties and uses different types of information, depending mainly on whether the data are quantized or not (see Appendix D). A well-known method of building a graph between points is Delaunay triangulation [63]. Unfortunately, efficient implementations of Delaunay triangulation only exist in the two-dimensional case. Moreover, although Delaunay triangulation has many desirable properties, it is often not suited in the case of data analysis. A justification of this can be found in [5], along with a good discussion about the construction of graphs.

The five rules presented hereafter are illustrated by two examples. The two data sets consist of 3000 2-dimensional points drawn from two different 1-manifolds: a sine wave with an increasing frequency and a linear spiral. Their parametric equations are, respectively,

$$\mathbf{y} = \begin{bmatrix} x \\ \sin(\pi \exp(3x)) \end{bmatrix} \qquad \text{with} \qquad x \in [0,1] \tag{E.1}$$

and

$$\mathbf{y} = \begin{bmatrix} 2x\cos(6\pi x) \\ 2x\sin(6\pi x) \end{bmatrix} \qquad \text{with} \qquad x \in [0,1] \ . \tag{E.2}$$

In both cases, Gaussian noise is added (for the sine, on y_2 only, with standard deviation equal to 0.10; for the spiral, on both y_1 and y_2, with standard deviation equal to 0.05). Afterwards, the 3000 points of each data set are quantized with 120 and 25 prototypes respectively. Figure E.1 illustrates both data sets. Both manifolds have the property of being one-dimensional only in a local scale. For example, going from left to right along the sine wave makes it appear "more and more two-dimensional" (see Chapter 3). For the rules that build a graph without assuming that the available points are prototypes resulting from a vector quantization, the prototypes are given as such. On the other hand, the rules relying on the fact that some data set has been quantized are given both the prototypes and the original data sets.

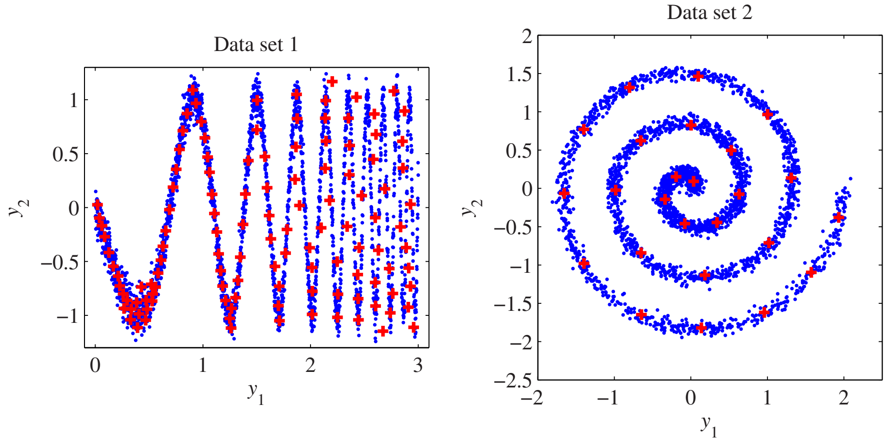


Fig. E.1. Two data sets used to test graph-building rules: the sine with exponentially increasing frequency and the linear spiral. Data points are displayed as points whereas prototypes are circles.

E.1 Without vector quantization

All the rules described ahead just assume a set of data points $\{\mathbf{y}(i)\}_{1\leq i\leq N}$.

E.1.1 *K*-rule

Also known as the rule of K-ary neighborhoods, this rule is actually very simple: each point $\mathbf{y}(i)$ is connected with the K closest other points. As a direct consequence, if the graph is undirected (see Section 4.3), then each point is connected with at least K other points. Indeed, each point elects exactly K

neighbors but can also be elected by points that do not belong to this set of K neighbors. This phenomenum typically happens with an isolated point: it elects as neighbors faraway points while those points find their K neighbors within a much smaller distance. Another consequence of this rule is that no (nontrivial) upper bound can be easily given for the longest distance between a point and its neighbors. The practical determination of the K closest points is detailed in Section F.2.

Assuming the data set $\{\mathbf{y}(i)\}_{1\leq i\leq N}$ is actually the set of 120 or 25 prototypes mentioned above, the K-rule gives the results displayed in the first row of Fig. E.2. Knowing that the manifolds are one-dimensional, the value of K is set to 2. For the sine wave, the result is good but gets progressively worse as the frequency increases. For the spiral, the obtained graph is totally wrong, mainly because the available points do not sample the spiral correctly (i.e., the distances between points on different whorls may be smaller than those of points lying on the same whorl).

E.1.2 ϵ-rule

By comparison with the K-rule, the ϵ-rule works almost conversely: each point $\mathbf{y}(i)$ is connected with all other points lying inside an ϵ-ball centered on $\mathbf{y}(i)$. Consequently, ϵ is by construction the upper bound for the longest distance between a point and its neighbors. But as a counterpart, no (nontrivial) lower bound can easily be given for the smallest number of neighbors that are connected with each point. Consequently, it may happen that isolated points have no neighbors. As extensively demonstrated in [19], the ϵ-rule shows better properties for the approximation of geodesic distances with graph distances. However, the choice of ϵ appears more difficult in practice than the one of K. The practical determination of the points lying closer than a fixed distance ϵ from another point is detailed in Section F.2.

Assuming the data set $\{\mathbf{y}(i)\}_{1\leq i\leq N}$ is actually the set of 120 or 25 prototypes mentioned above, the ϵ-rule gives the results displayed in the second row of Fig. E.2. The parameter ϵ is given the values 0.3 and 0.8 for, respectively, the sine and the spiral. As can be seen, the ϵ-rule gives good results only if the distribution of points remains approximately uniform; this assumption is not exactly true for the two proposed data sets. Consequently, the graph includes too many edges in dense regions like the first minimum of the sine wave and the center of the spiral. On the other hand, points lying in sparsely populated regions often remain disconnected.

E.1.3 τ-rule

This more complex rule connects two points $\mathbf{y}(i)$ and $\mathbf{y}(j)$ if they satisfy two conditions. The first condition states that the distances $d_i = \min_j \|\mathbf{y}(i)-\mathbf{y}(j)\|$ and $d_j = \min_i \|\mathbf{y}(j) - \mathbf{y}(i)\|$ from the two points to their respective closest neighbor is

$$d_i \leq \tau d_j \text{ and } d_j \leq \tau d_i \quad \text{(similarity cond.)} \ , \tag{E.3}$$

where τ is a tolerance greater than 1 (for example, $\tau = 1.5$ or 2.0). The second condition requires that the two points are neighbors in the sense:

$$\|\mathbf{y}(i) - \mathbf{y}(j)\| \leq \tau d_i \text{ or } \|\mathbf{y}(j) - \mathbf{y}(i)\| \leq \tau d_j \quad \text{(neighborhood cond.)} \ . \tag{E.4}$$

This rule behaves almost like the ϵ-rule, except that the radius is implicit, is hidden in τ, and adapts to the local data distribution. The mean radius increases in sparse regions and decreases in dense regions.

Assuming the data set $\{\mathbf{y}(i)\}_{1 \leq i \leq N}$ is actually the set of 120 or 25 prototypes mentioned above, the τ-rule gives the results displayed in the third row of Fig. E.2. For the sine as well as for the spiral, the parameter τ equals 1.5. As expected, the τ-rule behaves a little better than the ϵ-rule when the density of points varies. This is especially visible in dense regions: the numerous unnecessary edges produced by the ϵ-rule disappear. Moreover, it is noteworthy that the τ-rule better extracts the shape of the spiral than the two previous rules.

E.2 With vector quantization

When it is known that the points to be connected result from a vector quantization, the construction of the graph may exploit that additional information. Such information may be given, for example, by the local distribution of data points between prototypes that are close to each other. Some algorithms, called topology representing networks [136] (TRN) perform this task concurrently with vector quantization. Typical examples are the neural gas [135] and its numerous variants (such as [71]). The two simple rules detailed ahead, applied after vector quantization, yield similar or even better results.

E.2.1 Data rule

For each data point $\mathbf{y}(i)$, this rule computes the set containing the K closest prototypes, written as $\{\mathbf{c}(j_1), \ldots, \mathbf{c}(j_K)\}$. Then each possible pair $\{\mathbf{c}(j_s), \mathbf{c}(j_t)\}$ in this set is analyzed. If the point fulfills the following two conditions, then the prototypes of the considered pair are connected, and a graph edge is created between their associated vertices. The first one is the condition of the ellipse, written as

$$d(\mathbf{y}(i), \mathbf{c}(j_r)) + d(\mathbf{y}(i), \mathbf{c}(j_s)) < C_1 d(\mathbf{c}(j_r), \mathbf{c}(j_s)) \ , \tag{E.5}$$

and the second one is the condition of the circle

$$d(\mathbf{y}(i), \mathbf{c}(j_r)) < C_2 d(\mathbf{y}(i), \mathbf{c}(j_d)) \text{ and } d(\mathbf{y}(i), \mathbf{c}(j_s)) < C_2 d(\mathbf{y}(i), \mathbf{c}(j_r)) \ , \tag{E.6}$$

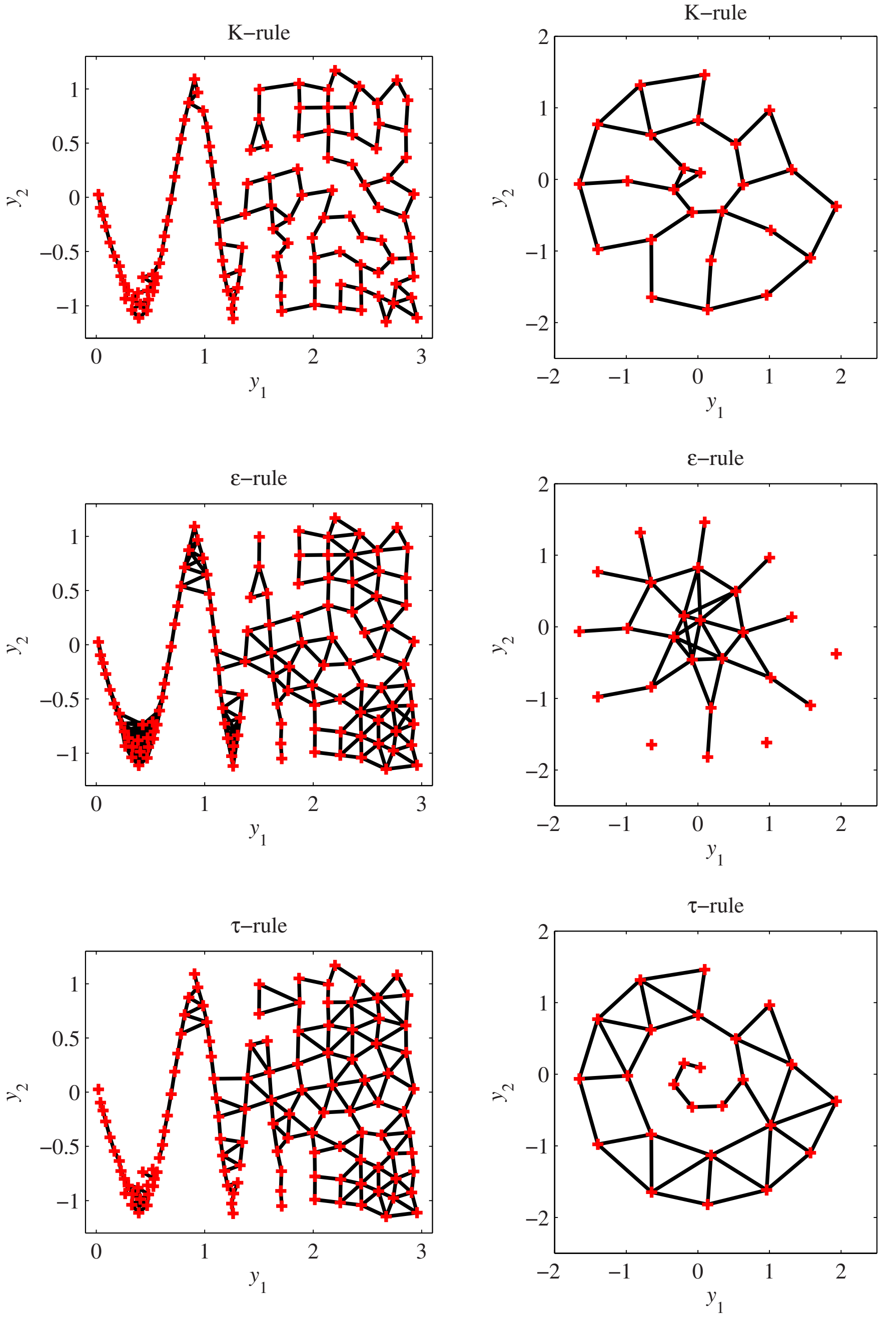


Fig. E.2. Results of the K-rule, ϵ-rule, and τ-rule on the data sets proposed in Fig. E.1.

where C_1 and C_2 are constants defined as

$$C_1 = \sqrt{S^2+1} \ , \tag{E.7}$$

$$C_2 = \frac{1+S}{1-S} \ , \tag{E.8}$$

The value of the constant S is chosen by the user, with values between 0.2 and 0.6. In this range, S can be seen as the edge of an almost cubic hypervolume (see Fig. E.3). Indeed, the first condition holds if $\mathbf{y}(i)$ lies inside a hyperellipsoid with foci at $\mathbf{c}(j_r)$ and $\mathbf{c}(j_s)$, whereas the second condition holds if $\mathbf{y}(i)$ lies outside two hyperballs including the two foci.

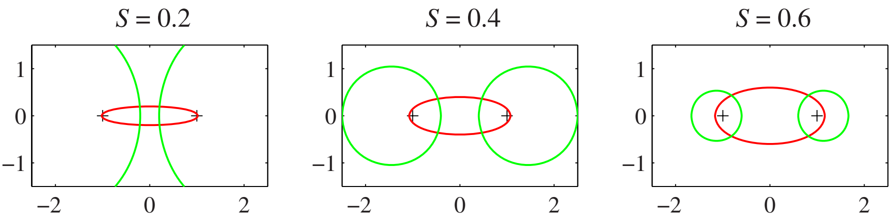


Fig. E.3. The two conditions to fulfill in order to create a graph edge between the vertices associated to two prototypes (black crosses). The points that create the edge must be inside the ellipse and outside both circles. The ellipse and circles are shown for different values of S.

Assuming the data set $\{\mathbf{y}(i)\}_{1\leq i\leq N}$ is actually the set of 120 or 25 prototypes mentioned in the introductory section, the data rule gives the results displayed in the first row of Fig. E.4. The parameter K equals 2, and S is assigned the values 0.5 and 0.3 for the sine and the spiral, respectively. As can be seen, the exploitation of the information provided by the data set before vector quantization allows one to extract the shape of both data sets in a much better way than the three previous rules did.

E.2.2 Histogram rule

This rule, described in [5], shares some common ideas with the previous one. In a similar way, the goal consists of connecting prototypes $\mathbf{c}(j_r)$ and $\mathbf{c}(j_s)$ only if data points lie between them, i.e., around the midpoint of a segment with $\mathbf{c}(j_r)$ and $\mathbf{c}(j_s)$ at both ends. In contrast with the data rule, however, the histogram rule is probabilistic.

In practice, for each data point $\mathbf{y}(i)$, the two closest prototypes $\mathbf{c}(j_r)$ and $\mathbf{c}(j_s)$ are computed and memorized as possible end vertices for a new edge of the graph. Obviously, several data points can lead to considering the same possible edge. These data points may be binned in a histogram according to the normalized scalar product

$$h(i) = \frac{\langle (\mathbf{y}(i) - \mathbf{c}(j_r)) \cdot (\mathbf{c}(j_s) - \mathbf{c}(j_r)) \rangle}{\langle (\mathbf{c}(j_s) - \mathbf{c}(j_r)) \cdot (\mathbf{c}(j_s) - \mathbf{c}(j_r)) \rangle} \quad .$$

The normalized scalar product can also be expressed with distances:

$$h(i) = \frac{\|\mathbf{y}(i) - \mathbf{c}(j_r)\|_2^2 - \|\mathbf{y}(i) - \mathbf{c}(j_s)\|_2^2 + \|\mathbf{c}(j_r) - \mathbf{c}(j_s)\|_2^2}{2\|\mathbf{c}(j_r) - \mathbf{c}(j_s)\|_2} \quad .$$

The histogram may contain only three bins:

- $h(i) < \frac{1}{2}(1 - H)$,
- $\frac{1}{2}(1 - H) \leq h(i) \leq \frac{1}{2}(1 + H)$,
- $\frac{1}{2}(1 + H) < h(i)$,

where $0 \leq H \leq 1$ is a parameter that determines the width of the central bin. The final decision to establish the edge between $\mathbf{c}(j_r)$ and $\mathbf{c}(j_s)$ is taken if the central bin is higher than the other two bins. In two dimensions, the histogram rule produces a subset of the Delaunay triangulation.

Assuming the data set $\{\mathbf{y}(i)\}_{1 \leq i \leq N}$ is actually the set of 120 or 25 prototypes mentioned in the introductory section, the histogram rule gives the results displayed in the second row of Fig. E.4. The results are similar to those of the data rule. It is noteworthy that the histogram rule behaves much better than the data rule when outliers pollute the data (in the artificial examples of Fig. E.1, only Gaussian noise is added).

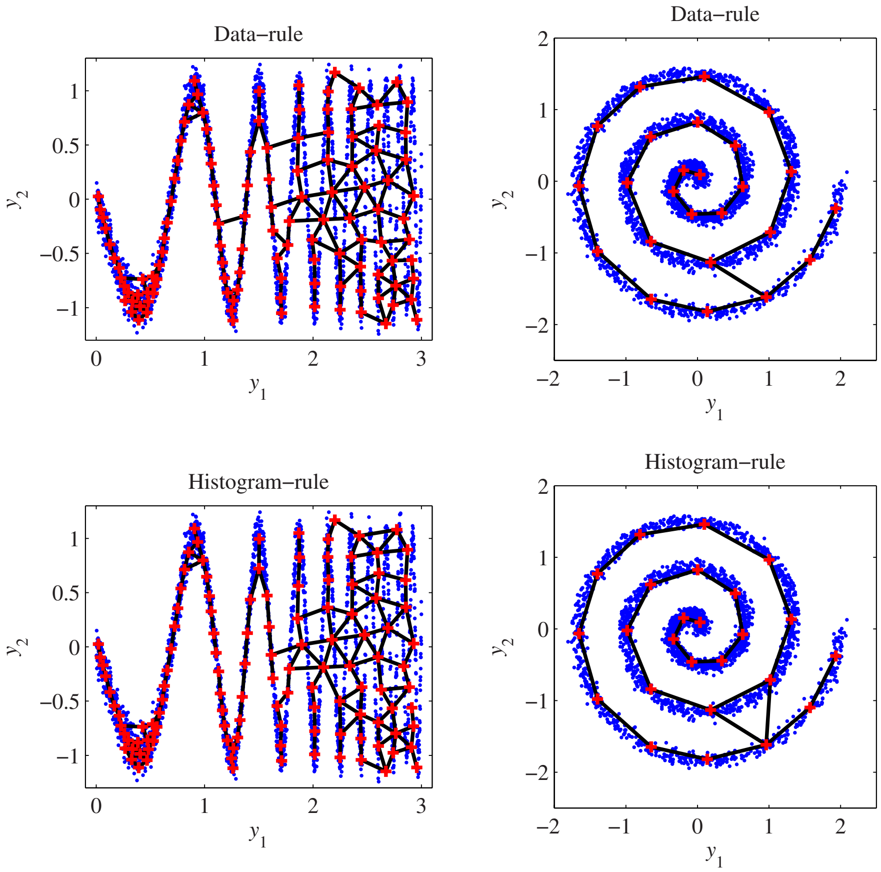


Fig. E.4. Results of the data rule and histogram rule on the data sets proposed in Fig. E.1.

F

Implementation Issues

This appendix gathers some hints in order to implement efficiently the methods and algorithms described in the main chapters.

F.1 Dimension estimation

This first section refers to the methods described in Chapter 3 to estimate the intrinsic dimensionality of a data set.

F.1.1 Capacity dimension

Some ideas to build efficient implementations of the box-counting dimension are given in [45]. Briefly put, the idea consists of assuming the presence of a D-dimensional grid that surrounds the region of the space occupied by the D-dimensional data. It is not a good idea to list all hypercubes in the grid and to count the number of points lying in each of them. Indeed, the number of hypercubes grows exponentially, and many of these "boxes" are likely to be empty if the intrinsic dimensionality is low, as expected. A better solution consists in listing only the nonempty hypercubes. This can be done by giving a label or code number to each box and assigning to each point the label of the box it is lying in.

F.1.2 Correlation dimension

By comparison with the capacity dimension, the correlation dimension is much easier to implement. In order to compute it and to draw the figures of Chapter 3, several pieces of code are needed.

The core of the algorithm consists of computing the pairwise distances involved in the correlation sum (Eq. (3.18)). For N points or observations,

$N(N-1)/2$ distances have to be computed. If N is large, the number of distances to store in memory becomes untractable. Fortunately, distances themselves are not interesting: only their cumulative distribution matters for the correlation sum. An idea to approximate that distribution consists of building a histogram. Hence, once a distance is computed, it is placed in the right bin and may be forgotten afterwards. This trick avoids storing all distances: only the histogram stands in the memory.

The histogram itself can be implemented quite simply by a simple vector. Each entry of the vector is a bin. For a given value, the index of the corresponding bin is computed in $\mathcal{O}(1)$ time. If the histogram h contains B bins, and if the first and last bins are located at values $h(1)$ and $h(B)$, then the index of the bin corresponding to value v is written as

$$b = \max\left(h(1), \min\left(h(B), \mathrm{rnd}\left(\frac{v - h(1)}{h(B) - h(1)}\right)\right)\right) \quad , \tag{F.1}$$

where the function $\mathrm{rnd}(x)$ rounds its argument x. The min and max prevent b from getting out of bounds. The choice of the number of bins B depends on the user's needs. On the other hand, the bounds of the histogram $h(1)$ and $h(B)$ are important in order to avoid wasted bins. The first thing to remember is that the correlation sum is always displayed in log-log plots. Therefore, the values to be binned in the histogram should be the logarithm of the distances, in order to yield an equal resolution (i.e., bin width) in the plot. For optimal efficiency, the choice of the histogram bounds is straightforward: $h(1)$ and $h(B)$ must equal the logarithm of, respectively, the minimal and maximal distances measured in the data set. Unfortunately, these distances are not known for sure until all $N(N-1)/2$ distances have been computed. This means that the distances should be either stored or computed twice. The first solution is the worst: the benefit of time is small, but the required amount of memory dramatically increases. A third solution would be to approximate the minimal and maximal distances. For example, a rather good approximation of the maximum:

$$\max_{i,j} \|\mathbf{y}(i) - \mathbf{y}(j)\|^2 \approx \max_{i} \|\mathbf{y}(i) - E\{\mathbf{y}\}\|^2 \quad , \tag{F.2}$$

where the expectation of $\mathbf{y}$ can be approximated by the sample mean, computed beforehand. This approximation reduces the computation time from $\mathcal{O}(N^2)$ to $\mathcal{O}(N)$. Although the exact value of the maximal distance is not so important, things are unfortunately much different for the minimal distance. Indeed, as the bounds of the histogram are the logarithms of these values, an error on a small value is much more dangerous than on a larger one. By the way, the knowledge of the exact minimal distance prevents us from having empty bins and causing a $\log 0 = -\infty$. In summary, it is better to compute the minimal and maximal distances exactly.

After all distances have been computed, the histogram bins may be cumulated in order to obtain the correlation sum $\hat{C}_2(\epsilon)$. Next, the logarithm is applied to the bin heights, yielding the necessary log-log curves. Finally, the numerical derivative is computed as in Eqs. (3.30) and (3.31). The second-order estimate is replaced with a first-order one for the first and last bins.

F.2 Computation of the closest point(s)

When searching in a set of points $\{\mathbf{y}(i)\}_{1\leq i\leq N}$ that are the K closest ones from given coordinates $\mathbf{y}$, one has to distinguish four cases:

1. $0 < K \ll N/2$;
2. $0 \ll K < N/2$;
3. $N/2 < K \ll N$;
4. $N/2 \ll K < N$.

The four cases are solved by the following procedure:

1. Compute all distances $d(\mathbf{y}(i), \mathbf{y})$ and store them in a vector.
2. Sort the vector containing distances.
3. Keep the first K entries of the sorted vector.

The most time-consuming step is the sort ($\mathcal{O}(N \log(N))$). If K is small (second case), then a partial sort can dramatically decrease the computation time. Moreover, the ordering of the K closest points often does not matter at all: the user just needs to know whether or not a given $\mathbf{y}(i)$ belongs to the K closest ones. Consequently, the last two cases in the above list can be elegantly solved just by computing the $N-K$ farthest points with a partial sort. At this point, the use of a partial sort algorithm leads to acceptable solutions for the last three cases. Unfortunately, if K is very small (first case), then the sorting time becomes negligible. Even a stupid partial sort like traversing K times the data set and looking for the minimum can be negligible (the complexity is only $\mathcal{O}(KN)$). Therefore, in the first case, thanks to the partial sort, the most time-consuming step becomes the computation of all distances ($\mathcal{O}(DN)$ where D, the dimensionality of the data, is assumed to be larger than K). Would it be possible to avoid the computation of all distances and get some additional speed-up ?

A positive answer to the above question is given by partial distance search (PDS in short). This technique exploits the structure of the Minkowski norm (see Subsection 4.2.1), which is computed as a sum of positive terms, elevated to some power:

$$\|\mathbf{y}(i) - \mathbf{y}\|_p = \sqrt[p]{\sum_{d=1}^{D} |y_d(i) - y_d|^p} \ . \tag{F.3}$$

As the pth root is a monotonic function, it does not change the ordering of the distance and may be omitted.

For the sake of simplicity, it is assumed that a sorted list of K canditates closest points is available. This list can be initialized by randomly choosing K

points in the data set, computing their distances (without applying the pth root) to the source point $\mathbf{x}$, and sorting the distances. Let ϵ be assigned with the value of the last and longest distance in the list. From this state, PDS has to traverse the $N-K$ remaining points in order to update the list. For each of these points $\mathbf{y}(i)$, PDS starts to compute the distance from the source $\mathbf{y}$, by cumulating the terms $|y_d(i) - y_d|^p$ for increasing values of index d. While the sum of those terms remains lower than ϵ, it is worth carrying on the additions, because the point remains a possible candidate to enter the list. Otherwise, if the sum grows beyond ϵ, it can be definitely deduced that the point is not a good candidate, and the PDS may simply stop the distance computation between $\mathbf{y}(i)$ and $\mathbf{y}$. If the point finally enters the list, the largest distance in the list is thrown away and the new point replaces it at the right position in the list.

The denomination ϵ is not given arbitrarily to the longest distance in the list. A small change in the PDS algorithm can perform another task very efficiently: the detection of the points $\mathbf{y}(i)$ lying in a hyperball of radius ϵ. In this variant, ϵ is fixed and the list has a dynamic size; the sorting of the list is unnecessary. The original and modified PDS algorithms find direct applications in the K-rule and ϵ-rule for graph building (see Appendix E).

Other techniques for an even faster determination of the closest points exist in the literature, but they either need special data structures built around the set of data points [207] or provide only an approximation of the exact result [26].

F.3 Graph distances

Within the framework of dimensionality reduction, graph distances are used to approximate geodesic distances on a manifold, i.e., distances that are measured *along* a manifold, like a man walking on it, and not as a bird flies, as does the Euclidean distance. Whereas Section 4.3 justfies and details the use of graph distances in the context of NLDR, this subsection focuses on some implementation issues.

Compared to Euclidean distances, graph distances are much more difficult to compute efficiently. The explanation of such a difference comes from the fact that computing the Euclidean distance between two points requires knowing the coordinates of these two points. More precisely, computing $d(\mathbf{y}(i), \mathbf{y}(j)) = \|\mathbf{y}(i) - \mathbf{y}(j)\|_2$, where $\mathbf{y}(i), \mathbf{y}(j) \in \mathbb{R}^D$, requires $\mathcal{O}(D)$ operations. (See Section 4.2 for more details about the Euclidean distance and the L_2-norm $\|\mathbf{y}\|_2$.)

Hence, when dealing with a set of N points, computing the Euclidean distances from one point, called the *source*, to all others trivially requires $\mathcal{O}(DN)$ operations. Similarly, the measurement of all pairwise distances (from all possible sources to all other points) demands $\mathcal{O}(DN^2)$; as the Euclidean distance is symmetric (see Subsection 4.2.1), only $N(N-1)/2$ pairwise distances need to be effectively computed. For the Euclidean distance, this distinction among three tasks may seem quite artificial. Indeed, computing the distance for one pair (SPED task; single-pair Euclidean distance), one source (SSED task; single-source Euclidean distances), or all sources (APED task; all-pairs Euclidean distances) simply demands repeating 1, N, or $N(N-1)/2$ times a single basic procedure that measures the distance between two points.

On the other hand, when determining graph distances, the distinction among the three tasks is essential. The first point to remark is that computing the graph distance $\delta(v_i, v_j)$ between two vertices of a weighted graph is wasteful. Intuitively, this fact can be justified by remarking that a graph distance depends on more than the two vertices between which it is computed. Indeed, computing the distance between v_i and v_j requires computing paths between v_i and v_j and finding the shortest one. If this shortest path is written $\{v_i, \ldots, v_k, \ldots, v_j\}$, then $\{v_i, \ldots, v_j\}$ is the shortest path between v_i and v_k; but, of course, v_k and all other intermediate vertices are not known in advance. Therefore, computing the shortest path from v_i to v_j (i.e., graph distance $\delta(\mathbf{y}(i), \mathbf{y}(j)))$ requires computing all intermediate shortest paths from v_i to any v_k (i.e. the distances $\delta(\mathbf{y}(i), \mathbf{y}(k)))$, subject to length$(\{v_i, \ldots, v_k\}) \leq$ length$(\{v_i, \ldots, v_j\})$ (resp., $\delta(\mathbf{y}(i), \mathbf{y}(k)) \leq \delta(\mathbf{y}(i), \mathbf{y}(j)))$. (For an undirected graph, as is often the case in applications to dimensionality reduction, a slightly cheaper solution consists of computing simultaneously shortest paths starting from both vertices v_i and v_i and stopping when they merge, instead of computing all shortest paths from v_i until v_j is reached.) As a consequence of the previous observations, it appears that the most basic procedure for computing graph distances should be the one that already measures all distances from one source to all other vertices. In the literature, this task is called the SSSP problem (single-source shortest paths). Similarly, the computation for all pairs is known as the APSP problem (all-pairs shortest paths).

SSSP (single-source shortest paths)

The problem of computing the shortest paths from one source vertex to all other vertices is usually solved by Dijkstra's [53] algorithm. Dijkstra's algorithm has already been sketched in Subsection 4.3.1; its time complexity is $\mathcal{O}(D|E| + N \log N)$, where $|E|$ is the number of edges in the graph. The main idea of the algorithm consists of computing the graph distances in ascending order. This way implicitly ensures that each distance is computed along the shortest path. This also means that distances have to be sorted in some way. Actually, at the beginning of the algorithm, all distances are initialized to $+\infty$, except for the distance to the source, which is trivially zero. Other distances are updated and output one by one, as the algorithm is running and discovering the shortest paths. Hence, distances are not really sorted, but their intermediate values are stored in a priority queue, which allows us to store a set of values and extract the smallest one. An essential property of a priority queue is the possibility to decrease the stored values. In the case of Dijkstra's algorithm, this functionality is very important since distances are not known in advance and may be lowered every time a shorter path is discovered. Priority queues with efficient operations for extracting the minimum and updating the stored values are implemented by data structures called *heaps*. In particular, Fibonacci heaps are exceptionally well fitted to Dijkstra's algorithm [65].

Dijkstra's algorithm computes not only the length of the shortest paths (i.e., the graph distances) but also the shortest path itself. Actually, associated with each vertex v_j is a pointer to the vertex v_k, which is its predecessor in the shortest path that goes from the source v_i to v_j. If the graph is connected, the shortest paths computed by Dijkstra's algorithm form a spanning tree of the graph.

As a byproduct, Dijkstra's algorithm outputs the graph distances in ascending order. They are implicitly sorted by the algorithm.

APSP (all-pairs shortest paths)

Usually, the problem of computing all pairwise graph distances in a weighted graph is solved by repeating an SSSP algorithm (like Dijkstra's one) for each vertex [210]. This procedure requires $\mathcal{O}(ND|E| + N^2 \log N)$ operations, which is far more expensive than the $\mathcal{O}(DN^2)$ operations required to achieve the equivalent task for Euclidean distances. Hopefully, it can be seen that repeating an SSSP procedure generates a lot of redundancy. More precisely, in the perspective of the APSP, a single execution of an SSSP procedure gives more than the shortest paths from the specified source v_s to all other vertices. In fact, determining the shortest path $\{v_i, \ldots, v_k, \ldots, v_j\}$ from v_i to some v_j not only requires computing all intermediate paths $\{v_i, \ldots, v_k\}$ but also gives the shortest path $\{v_k, \ldots, v_j\}$ for free. Moreover, if the graph is undirected, then the shortest paths $\{v_j, \ldots, v_k, \ldots, v_i\}$, $\{v_j, \ldots v_k\}$, and $\{v_k, \ldots, v_i\}$ are known in advance, too. Unfortunately, to the authors' knowledge, no existing algorithm succeeds in exploiting this redundancy. References to improved APSP algorithms, performing faster in particular situations (e.g., integer weight, approximated solution) can be found in [210].

References

1. C.C. Aggarwal, A. Hinneburg, and D.A. Keim. On the surprising behavior of distance metrics in high dimensional space. In J. Van den Bussche and V. Vianu, editors, *Proceedings of the Eighth International Conference on Database Theory*, volume 1973 of *Lecture Notes in Computer Science*, pages 420–434. Springer, London, 2001.
2. D.W. Aha and R.L. Bankert. A comparative evaluation of sequential feature selection algorithms. In D. Fisher and H.J. Lenz, editors, *Learning from Data: AI and Statistics*, pages 199–206. Springer-Verlag, New York, NY, 1996. Also published in the *Proc. 5th Int. Workshop on AI and Statistics*.
3. A. Ahalt, A.K. Krishnamurthy, P. Chen, and D.E. Melton. Competitive learning algorithms for vector quantization. *Neural Networks*, 3:277–290, 1990.
4. H. Akaike. Information theory and an extension of the maximum likelihood principle. In B.N. Petrov and F. Csaki, editors, *Proceedings of the 2nd International Symposium on Information Theory*, pages 267–281. Akademia Kiado, Budapest, 1973.
5. M. Aupetit. Robust topology representing networks. In M. Verleysen, editor, *Proceedings of ESANN 2003, 11th European Symposium on Artificial Neural Networks*, pages 45–50. d-side, Bruges, Belgium, April 2003.
6. C. Baker. *The Numerical Treatment of Integral Equations.* Clarendon Press, Oxford, 1977.
7. K. Balász. *Principal curves: learning, design, and applications.* PhD thesis, Concordia University, Montréal, Canada, 1999.
8. G.B. Ball and D.J. Hall. A clustering technique for summarizing multivariate data. *Behavioral Science*, 12:153–155, 1967.
9. H.-U. Bauer, M. Herrmann, and T. Villmann. Neural maps and topographic vector quantization. *Neural Networks*, 12:659–676, 1999.
10. H.-U. Bauer and K.R. Pawelzik. Quantifying the neighborhood preservation of self-organizing maps. *IEEE Transactions on Neural Networks*, 3:570–579, 1992.
11. H.-U. Bauer and T. Villmann. Growing a hypercubical output space in a self-organizing feature map. Technical Report TR-95-030, International Computer Science Institute, Berkeley, 1995.
12. M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In T.G. Dietterich, S. Becker, and Z. Ghahramani,

editors, *Advances in Neural Information Processing Systems (NIPS 2001)*, volume 14. MIT Press, Cambridge, MA, 2002.

13. M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, June 2003.
14. R. Bellman. *Adaptative Control Processes: A Guided Tour.* Princeton University Press, Princeton, NJ, 1961.
15. Y. Bengio, O. Delalleau, N. Le Roux, J.-F. Paiement, P. Vincent, and M. Ouimet. Learning eigenfunctions links spectral embedding and kernel PCA. *Neural Computation*, 16(10):2197–2219, 2004.
16. Y. Bengio, J.-F. Paiement, P. Vincent, O. Delalleau, N. Le Roux, and M. Ouimet. Out-of-sample extensions for LLE, isomap, MDS, eigenmaps, and spectral clustering. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems (NIPS 2003)*, volume 16. MIT Press, Cambridge, MA, 2004.
17. Y. Bengio, P. Vincent, J.-F. Paiement, O. Delalleau, M. Ouimet, and N. Le Roux. Spectral clustering and kernel PCA are learning eigenfunctions. Technical Report 1239, Département d'Informatique et Recherche Opérationnelle, Université de Montréal, Montréal, July 2003.
18. N. Benoudjit, C. Archambeau, A. Lendasse, J. Lee, and M. Verleysen. Width optimization of the Gaussian kernels in radial basis function networks. In M. Verleysen, editor, *Proceedings of ESANN 2002, 10th European Symposium on Artificial Neural Networks*, pages 425–432. d-side, Bruges, Belgium, April 2002.
19. M. Bernstein, V. de Silva, J.C. Langford, and J.B. Tenenbaum. Graph approximations to geodesics on embedded manifolds. Technical report, Stanford University, Palo Alto, CA, December 2000.
20. K.S. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In *Seventh International Conference on Database Theory*, volume 1540 of *Lecture Notes in Computer Science*, pages 217–235. Springer-Verlag, Jerusalem, Israel, 1999.
21. C.M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, Oxford, 1995.
22. C.M. Bishop, M. Svensén, and C.K.I. Williams. EM optimization of latent-variables density models. In D.S. Touretzky, M.C. Mozer, and M.E. Hasselmo, editors, *Advances in Neural Information Processing Systems (NIPS 1995)*, volume 8, pages 465–471. MIT Press, Cambridge, MA, 1996.
23. C.M. Bishop, M. Svensén, and C.K.I. Williams. GTM: A principled alternative to the self-organizing map. In M.C. Mozer, M.I. Jordan, and T. Petsche, editors, *Advances in Neural Information Processing Systems (NIPS 1996)*, volume 9, pages 354–360. MIT Press, Cambridge, MA, 1997.
24. C.M. Bishop, M. Svensén, and K.I. Williams. GTM: A principled alternative to the self-organizing map. *Neural Computation*, 10(1):215–234, 1998.
25. I. Borg and P. Groenen. *Modern Multidimensional Scaling: Theory and Applications.* Springer-Verlag, New York, 1997.
26. A. Borodin, R. Ostrovsky, and Y. Rabani. Lower bounds for high dimensional nearest neighbor search and related problems. In *Proceedings of the thirty-first annual ACM symposium on Theory of Computing, Atlanta, GA*, pages 312–321. ACM Press, New York, 1999.

27. B.E. Boser, I.M. Guyon, and V. Vapnik. A training algorithm for optimal margin classifiers. In *Fifth Annual Workshop on Computational Learning Theory.* ACM, Pittsburg, PA, 1992.
28. C. Bouveyron. Dépliage du ruban cortical à partir d'images obtenues en IRMf, mémoire de DEA de mathématiques appliquées. Master's thesis, Unité Mixte Inserm – UJF 594, Université Grenoble 1, France, June 2003.
29. M. Brand. Charting a manifold. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems (NIPS 2002)*, volume 15. MIT Press, Cambridge, MA, 2003.
30. M. Brand. Minimax embeddings. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems (NIPS 2003)*, volume 16. MIT Press, Cambridge, MA, 2004.
31. M. Brand and K. Huang. A unifying theorem for spectral embedding and clustering. In C.M. Bishop and B.J. Frey, editors, *Proceedings of International Workshop on Artificial Intelligence and Statistics (AISTATS'03).* Key West, FL, January 2003. Also presented at NIPS 2002 workshop on spectral methods and available as Technical Report TR2002-042.
32. J. Bruske and G. Sommer. Intrinsic dimensionality estimation with optimally topology preserving maps. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(5):572–575, 1998.
33. M.A. Carreira-Perpiñán. A review of dimension reduction techniques. Technical report, University of Sheffield, Sheffield, January 1997.
34. A. Cichocki and S. Amari. *Adaptative Blind Signal and Image Processing.* John Wiley & Sons, New York, 2002.
35. R.R. Coifman, S. Lafon, A.B. Lee, M. Maggioni, B. Nadler, F. Warner, and S. Zucker. Geometric diffusion as a tool for harmonic analysis and structure definition of data, part i: Diffusion maps. *Proceedings of the National Academy of Sciences*, 102(21):7426–7431, 2005.
36. R.R. Coifman, S. Lafon, A.B. Lee, M. Maggioni, B. Nadler, F. Warner, and S. Zucker. Geometric diffusion as a tool for harmonic analysis and structure definition of data, part i: Multiscale methods. *Proceedings of the National Academy of Sciences*, 102(21):7432–7437, 2005.
37. C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.
38. M. Cottrell, J.-C. Fort, and G. Pagès. Two or three things that we know about the Kohonen algorithm. In M. Verleysen, editor, *Proceedings ESANN'94, 2nd European Symposium on Artificial Neural Networks*, pages 235–244. D-Facto conference services, Brussels, Belgium, 1994.
39. R. Courant and D. Hilbert. *Methods of Mathematical Physics*, volume 1. Interscience Publishers, Inc., New York, 1953.
40. T.M. Cover and J.A. Thomas. *Elements of Information Theory.* John Wiley, New York, 1991.
41. T.F. Cox and M.A.A. Cox. *Multidimensional Scaling.* Chapman & Hall, London, 1995.
42. N. Cristianini and J. Shawe-Taylor. *An Introduction to Support Vector Machines (and Other Kernel-Based Learning Methods).* Cambridge University Press, 2000.
43. J. de Leeuw and W. Heiser. Theory of multidimensional scaling. In *Handbook of Statistics*, chapter 13, pages 285–316. North-Holland Publishing Company, Amsterdam, 1982.

44. D. de Ridder and R.P.W. Duin. Sammon's mapping using neural networks: A comparison. *Pattern Recognition Letters*, 18(11–13):1307–1316, 1997.
45. P. Demartines. *Analyse de données par réseaux de neurones auto-organisés.* PhD thesis, Institut National Polytechnique de Grenoble (INPG), Grenoble, France, 1994.
46. P. Demartines and J. Hérault. Vector quantization and projection neural network. volume 686 of *Lecture Notes in Computer Science*, pages 328–333. Springer-Verlag, New York, 1993.
47. P. Demartines and J. Hérault. CCA: Curvilinear component analysis. In *15th Workshop GRETSI*, Juan-les-Pins (France), September 1995.
48. P. Demartines and J. Hérault. Curvilinear component analysis: A self-organizing neural network for nonlinear mapping of data sets. *IEEE Transactions on Neural Networks*, 8(1):148–154, January 1997.
49. D. DeMers and G.W. Cottrell. Nonlinear dimensionality reduction. In D. Hanson, J. Cowan, and L. Giles, editors, *Advances in Neural Information Processing Systems (NIPS 1992)*, volume 5, pages 580–587. Morgan Kaufmann, San Mateo, CA, 1993.
50. A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of Royal Statistical Society, B*, 39(1):1–38, 1977.
51. D. DeSieno. Adding a conscience to competitive learning. In *Proceedings of ICNN'88 (International Conference on Neural Networks)*, pages 117–124. IEEE Service Center, Piscataway, NJ, 1988.
52. G. Di Battista, P. Eades, R. Tamassia, and I.G. Tollis. Algorithms for drawing graphs: An annotated bibliography. Technical report, Brown University, June 1994.
53. E.W. Dijkstra. A note on two problems in connection with graphs. *Numerical Mathematics*, 1:269–271, 1959.
54. D. Donoho and C. Grimes. When does geodesic distance recover the true parametrization of families of articulated images? In M. Verleysen, editor, *Proceedings of ESANN 2002, 10th European Symposium on Artificial Neural Networks*, pages 199–204, Bruges, Belgium, April 2002. d-side.
55. D.L. Donoho and C. Grimes. Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data. In *Proceedings of the National Academy of Arts and Sciences*, volume 100, pages 5591–5596, 2003.
56. D.L. Donoho and C. Grimes. Hessian eigenmaps: New locally linear techniques for high-dimensional data. Technical Report TR03-08, Department of Statistics, Stanford University, Palo Alto, CA, 2003.
57. E. Erwin, K. Obermayer, and K. Schulten. Self-organizing maps: ordering, convergence properties and energy functions. *Biological Cybernetics*, 67:47–55, 1992.
58. P.A. Estévez and A.M. Chong. Geodesic nonlinear mapping using the neural gas network. In *Proceedings of IJCNN 2006*. 2006. In press.
59. P.A. Estévez and C.J. Figueroa. Online data visualization using the neural gas network. *Neural Networks*, 19:923–934, 2006.
60. B.S. Everitt. *An Introduction to Latent Variable Models*. Monographs on Statistics and Applied Probability. Chapman & Hall, London, New York, 1984.
61. E. Forgy. Cluster analysis of multivariate data: efficiency versus interpretability of classifications. *Biometrics*, 21:768, 1965.

62. J.-C. Fort. SOM's mathematics. *Neural Networks*, 19:812–816, 2006.
63. S. Fortune. Voronoi diagrams and delaunay triangulations. In D.Z. Du and F. Hwang, editors, *Computing in Euclidean geometry*, pages 193–233. World Scientific, Singapore, 1992.
64. D. François. *High-dimensional data analysis: optimal metrics and feature selection.* PhD thesis, Université catholique de Louvain, Département d'Ingénierie Mathématique, Louvain-la-Neuve, Belgium, September 2006.
65. M.L. Fredman and R.E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34:596–615, 1987.
66. J.H. Friedman. Exploratory projection pursuit. *Journal of the American Statistical Association*, 82(397):249–266, March 1987.
67. J.H. Friedman and J.W. Tukey. A projection pursuit algorithm for exploratory data analysis. *IEEE Transactions on Computers*, C23(9):881–890, 1974.
68. B. Fritzke. Let it grow – self-organizing feature maps with problem dependent cell structure. In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas, editors, *Artificial Neural Networks*, volume 1, pages 403–408. Elsevier, Amsterdam, 1991.
69. B. Fritzke. Growing cell structures – a self-organizing network for unsupervised and supervised learning. *Neural Networks*, 7(9):1441–1460, 1994. Also available as technical report TR-93-026 at the International Computer Science Institute (Berkeley, CA), May 1993.
70. B. Fritzke. Growing grid – a self-organizing network with constant neighborhood range and adaptation strength. *Neural Processing Letters*, 2(5):9–13, 1995.
71. B. Fritzke. A growing neural gas network learns topologies. In G. Tesauro, D.S. Touretzky, and T.K. Leen, editors, *Advances in Neural Information Processing Systems (NIPS 1994)*, volume 7, pages 625–632. MIT Press, Cambridge, MA, 1995.
72. K. Fukunaga and D.R. Olsen. An algorithm for finding intrinsic dimensionality of data. *IEEE Transactions on Computers*, C-20(2):176–193, 1971.
73. A. Gersho and R.M. Gray. *Vector Quantization and Signal Processing.* Kluwer Academic Publisher, Boston, 1992.
74. G.J. Goodhill and T.J. Sejnowski. Quantifying neighbourhood preservation in topographic mappings. In *Proceedings of the Third Joint Symposium on Neural Computation*, pages 61–82. California Institute of Technology, University of California, Pasadena, CA, 1996.
75. J. Göppert. Topology-preserving interpolation in self-organizing maps. In *Proceedings of NeuroNîmes 1993*, pages 425–434. Nanterre, France, October 1993.
76. P. Grassberger and I. Procaccia. Measuring the strangeness of strange attractors. *Physica*, D9:189–208, 1983.
77. R.M. Gray. Vector quantization. *IEEE Acoustics, Speech and Signal Processing Magazine*, 1:4–29, April 1984.
78. J. Ham, D.D. Lee, S. Mika, and B. Schölkopf. A kernel view of the dimensionality reduction of manifolds. In *21th International Conference on Machine Learning (ICML-04)*, pages 369–376, 2004. Also available as technical report TR-102 at Max Planck Institute for Biological Cybernetics, Tübingen, Germany, 2003.
79. T. Hastie. *Principal curves and surfaces.* PhD thesis, Stanford University, Palo Alto, CA, 1984.

80. T. Hastie and W. Stuetzle. Principal curves. *Journal of the American Statistical Association*, 84(406):502–516, 1989.
81. X.F. He and P. Niyogi. Locality preserving projections. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Advances in Neural Information Processing Systems (NIPS 2003)*, volume 16. MIT Press, Cambridge, MA, 2004.
82. X.F. He, S.C. Yan, Y.X. Hu, H.G. Niyogi, and H.J. Zhang. Face recognition using Laplacianfaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(3):328–340, 2005.
83. H.G.E. Hentschel and I. Procaccia. The infinite number of generalized dimensions of fractals and strange attractors. *Physica*, D8:435–444, 1983.
84. J. Hérault, C. Jaussions-Picaud, and A. Guérin-Dugué. Curvilinear component analysis for high dimensional data representation: I. Theoretical aspects and practical use in the presence of noise. In J. Mira and J.V. Sánchez, editors, *Proceedings of IWANN'99*, volume II, pages 635–644. Springer, Alicante, Spain, June 1999.
85. M. Herrmann and H.H. Yang. Perspectives and limitations of self-organizing maps in blind separation of source signals. In S. Amari, L. Xu, L.-W. Chan, I. King, and K.-S. Leung, editors, *Progress in Neural Information Processing, Proceedings of ICONIP'96*, volume 2, pages 1211–1216. Springer-Verlag, 1996.
86. D. Hilbert. Über die stetige Abbildung einer Linie auf ein Flachenstück. *Math. Ann.*, 38:459–460, 1891.
87. G. Hinton and S.T. Roweis. Stochastic neighbor embedding. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems (NIPS 2002)*, volume 15, pages 833–840. MIT Press, Cambridge, MA, 2003.
88. G.E. Hinton. Learning distributed representations of concepts. In *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*, Amherst, MA, 1986. Reprinted in R.G.M. Morris, editor, *Parallel Distributed Processing: Implications for Psychology and Neurobiology*, Oxford University Press, USA, 1990.
89. G.E. Hinton and R.R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 313(5786):504–507, July 2006.
90. G.E. Hinton and R.R. Salakhutdinov. Supporting online material for "reducing the dimensionality of data with neural networks". *Science*, 313(5786):504–507, July 2006. Available at `www.sciencemag.org/cgi/content/full/313/5786/502/DC1`.
91. J.J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. In *Proc. Natl. Acad. Sci. USA 79*, pages 2554–2558. 1982.
92. H. Hotelling. Analysis of a complex of statistical variables into principal components. *Journal of Educational Psychology*, 24:417–441, 1933.
93. J.R. Howlett and L.C. Jain. *Radial Basis Function Networks 1: Recent Developments in Theory and Applications*. Physica Verlag, Heidelberg, 2001.
94. P.J. Huber. Projection pursuit. *Annals of Statistics*, 13(2):435–475, 1985.
95. A. Hyvärinen, J. Karhunen, and E. Oja. *Independent Component Analysis*. Wiley-Interscience, 2001.
96. A.K. Jain and D. Zongker. Feature selection: Evaluation, application and small sample performance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(2):153–158, 1997.

97. M.C. Jones and R. Sibson. What is projection pursuit? *Journal of the Royal Statistical Society, Series A*, 150:1–36, 1987.
98. C. Jutten. *Calcul neuromimétique et traitement du signal, analyse en composantes indépendantes.* PhD thesis, Institut National Polytechnique de Grenoble, 1987.
99. C. Jutten and J. Hérault. Space or time adaptive signal processing by neural network models. In *Neural Networks for Computing, AIP Conference Proceedings*, volume 151, pages 206–211. Snowbird, UT, 1986.
100. C. Jutten and J. Hérault. Blind separation of sources, part I: An adaptative algorithm based on neuromimetic architecture. *Signal Processing*, 24:1–10, 1991.
101. N. Kambhatla and T.K. Leen. Dimension reduction by local principal component analysis. *Neural Computation*, 9(7):1493–1516, October 1994.
102. K. Karhunen. Zur Spektraltheorie stochastischer Prozesse. *Ann. Acad. Sci. Fennicae*, 34, 1946.
103. K. Kiviluoto. Topology preservation in self-organizing maps. In IEEE Neural Networks Council, editor, *Proc. Int. Conf. on Neural Networks, ICNN'96*, volume 1, pages 294–299, Piscataway, NJ, 1996. Also available as technical report A29 of the Helsinki University of Technology.
104. T. Kohonen. Self-organization of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.
105. T. Kohonen. *Self-Organizing Maps.* Springer, Heidelberg, 2nd edition, 1995.
106. A. König. Interactive visualization and analysis of hierarchical neural projections for data mining. *IEEE Transactions on Neural Networks*, 11(3):615–624, 2000.
107. M. Kramer. Nonlinear principal component analysis using autoassociative neural networks. *AIChE Journal*, 37:233, 1991.
108. J.B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29:1–28, 1964.
109. J.B. Kruskal. Toward a practical method which helps uncover the structure of a set of multivariate observations by finding the linear transformation which optimizes a new index of condensation. In R.C. Milton and J.A. Nelder, editors, *Statistical Computation.* Academic Press, New York, 1969.
110. J.B. Kruskal. Linear transformations of multivariate data to reveal clustering. In *Multidimensional Scaling: Theory and Application in the Behavioural Sciences, I, Theory.* Seminar Press, New York and London, 1972.
111. W. Kühnel. *Differential Geometry Curves – Surfaces – Manifolds.* Amer. Math. Soc., Providence, RI, 2002.
112. M. Laurent and F. Rendl. Semidefinite programming and integer programming. Technical Report PNA-R0210, CWI, Amsterdam, April 2002.
113. M.H. C. Law, N. Zhang, and A.K. Jain. Nonlinear manifold learning for data stream. In *Proceedings of SIAM Data Mining*, pages 33–44. Orlando, FL, 2004.
114. J.A. Lee, C. Archambeau, and M. Verleysen. Locally linear embedding versus Isotop. In M. Verleysen, editor, *Proceedings of ESANN 2003, 11th European Symposium on Artificial Neural Networks*, pages 527–534. d-side, Bruges, Belgium, April 2003.
115. J.A. Lee, C. Jutten, and M. Verleysen. Non-linear ICA by using isometric dimensionality reduction. In C.G. Puntonet and A. Prieto, editors, *Independent Component Analysis and Blind Signal Separation*, Lecture Notes in Computer Science, pages 710–717, Granada, Spain, 2004. Springer-Verlag.

116. J.A. Lee, A. Lendasse, N. Donckers, and M. Verleysen. A robust nonlinear projection method. In M. Verleysen, editor, *Proceedings of ESANN 2000, 8th European Symposium on Artificial Neural Networks*, pages 13–20. D-Facto public., Bruges, Belgium, April 2000.
117. J.A. Lee, A. Lendasse, and M. Verleysen. Curvilinear distances analysis versus isomap. In M. Verleysen, editor, *Proceedings of ESANN 2002, 10th European Symposium on Artificial Neural Networks*, pages 185–192. d-side, Bruges, Belgium, April 2002.
118. J.A. Lee and M. Vereleysen. How to project "circular" manifolds using geodesic distances. In M. Verleysen, editor, *Proceedings of ESANN 2004, 12th European Symposium on Artificial Neural Networks*, pages 223–230. d-side, April 2004.
119. J.A. Lee and M. Verleysen. Nonlinear projection with the Isotop method. In J.R. Dorronsoro, editor, *LNCS 2415: Artificial Neural Networks, Proceedings of ICANN 2002*, pages 933–938. Springer, Madrid (Spain), August 2002.
120. J.A. Lee and M. Verleysen. Curvilinear distance analysis versus isomap. *Neurocomputing*, 57:49–76, March 2004.
121. J.A. Lee and M. Verleysen. Nonlinear dimensionality reduction of data manifolds with essential loops. *Neurocomputing*, 67:29–53, 2005.
122. A. Lendasse, J. Lee, and M. Verleysen. Forecasting electricity consumption using nonlinear projection and self-organizing maps. *Neurocomputing*, 48:299–311, October 2002.
123. A. Lendasse, J.A. Lee, V. Wertz, and M. Verleysen. Time series forecasting using CCA and Kohonen maps – application to electricity consumption. In M. Verleysen, editor, *Proceedings of ESANN 2000, 8th European Symposium on Artificial Neural Networks*, pages 329–334. D-Facto public., Bruges, Belgium, April 2000.
124. Y. Linde, A. Buzo, and R.M. Gray. An algorithm for vector quantizer design. *IEEE Transactions on Communications*, 28:84–95, 1980.
125. N. Linial, E. London, and Y. Rabinovich. The geometry of graphs and some of its algorithmic applications. *Combinatorica*, 15(2):215–245, 1995.
126. L. Ljung. *System Identification: Theory for the User*. Prentice Hall Information and System Sciences Series. Prentice Hall, Englewood Cliffs, NJ, 2nd edition, 1999.
127. S.P. Lloyd. Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28:129–137, 1982. Unpublished memorandum, 1957, Bell Laboratories.
128. M. Loève. Fonctions aléatoire du second ordre. In P. Lévy, editor, *Processus stochastiques et mouvement Brownien*, page 299. Gauthier-Villars, Paris, 1948.
129. D.J.C. MacKay. Bayesian neural networks and density networks. *Nuclear Instruments and Methods in Physics Research, Section A*, 354(1):73–80, 1995.
130. J.B. MacQueen. Some methods for classification and analysis of multivariate observations. In L.M. Le Cam and J. Neyman, editors, *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability. Volume I: Statistics*, pages 281–297. University of California Press, Berkeley and Los Angeles, CA, 1967.
131. B.B. Mandelbrot. How long is the coast of Britain? *Science*, 155:636–638, 1967.
132. B.B. Mandelbrot. *How long is the coast of Britain?* Freeman, San Francisco, 1982.

133. B.B. Mandelbrot. *Les objets fractals: forme, hasard et dimension.* Flammarion, Paris, 1984.
134. J. Mao and A.K. Jain. Artificial neural networks for feature extraction and multivariate data projection. *IEEE Transactions on Neural Networks*, 6(2):296–317, March 1995.
135. T. Martinetz and K. Schulten. A "neural-gas" network learns topologies. In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas, editors, *Artificial Neural Networks*, volume 1, pages 397–402. Elsevier, Amsterdam, 1991.
136. T. Martinetz and K. Schulten. Topology representing networks. *Neural Networks*, 7(3):507–522, 1994.
137. W. McCulloch and W. Pitts. A logical calculus of ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943.
138. M. Minsky and S. Papert. *Perceptrons: An Introduction to Computational Geometry.* MIT Press, Cambridge, MA, 1969.
139. L.C. Molina, L. Belanche, and À. Nebot. Feature selection algorithms: A survey and experimental evaluation. In *Proceedings of 2002 IEEE International Conference on Data Mining (ICDM'02)*, pages 306–313. December 2002. Also available as technical report LSI-02-62-R at the Departament de Lleguatges i Sistemes Informàtics of the Universitat Politècnica de Catalunya, Spain.
140. J.R. Munkres. *Topology: A First Course.* Prentice-Hall, Englewood Cliffs, NJ, 1975.
141. B. Nadler, S. Lafon, R.R. Coifman, and I.G. Kevrekidis. Diffusion maps, spectral clustering and eigenfunction of fokker-planck operators. In Y. Weiss, B. Schölkopf, and J. Platt, editors, *Advances in Neural Information Processing Systems (NIPS 2005)*, volume 18. MIT Press, Cambridge, MA, 2006.
142. R.M. Neal. *Bayesian Learning for Neural Networks.* Springer Series in Statistics. Springer-Verlag, Berlin, 1996.
143. A.Y. Ng, M.I. Jordan, and Y. Weiss. On spectral clustering: analysis and an algorithm. In T.G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems (NIPS 2001)*, volume 14. MIT Press, Cambridge, MA, 2002.
144. E. Oja. Data compression, feature extraction, and autoassociation in feedforward neural networks. In T. Kohonen, K. Mäkisara, O. Simula, and J. Kangas, editors, *Artificial Neural Networks*, volume 1, pages 737–745. Elsevier Science Publishers, B.V., North-Holland, 1991.
145. E. Oja. Principal components, minor components, and linear neural networks. *Neural Networks*, 5:927–935, 1992.
146. E. Ott. Measure and spectrum of d_q dimensions. In *Chaos in Dynamical Systems*, pages 78–81. Cambridge University Press, New York, 1993.
147. P. Pajunen. Nonlinear independent component analysis by self-organizing maps. In C. von der Malsburg, W. von Seelen, J.C. Vorbruggen, and B. Sendhoff, editors, *Artificial Neural Networks, Proceedings of ICANN'96*, pages 815–820. Springer-Verlag, Bochum, Germany, 1996.
148. E. Pękalska, D. de Ridder, R.P.W. Duin, and M.A. Kraaijveld. A new method of generalizing Sammon mapping with application to algorithm speed-up. In M. Boasson, J.A. Kaandorp, J.F.M. Tonino, and M.G. Vosselman, editors, *Proceedings of ASCI'99, 5th Annual Conference of the Advanced School for Computing and Imaging*, pages 221–228. ASCI, Delft, The Netherlands, June 1999.

149. K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2:559–572, 1901.
150. J. Peltonen, A. Klami, and S. Kaski. Learning metrics for information visualisation. In *Proceedings of the 4th Workshop on Self-Organizing Maps (WSOM'03)*, pages 213–218. Hibikino, Kitakyushu, Japan, September 2003.
151. Y.B. Pesin. On rigorous mathematical definition of the correlation dimension and generalized spectrum for dimension. *J. Stat. Phys.*, 71(3/4):529–547, 1993.
152. Y.B. Pesin. *Dimension Theory in Dynamical Systems: Contemporary Views and Applications*. The University of Chicago Press, Chicago, 1998.
153. J. Rissanen. Modelling by shortest data description. *Automatica*, 10:465–471, 1978.
154. H. Ritter, T. Martinetz, and K. Schulten. *Neural Computation and Self-Organizing Maps*. Addison-Wesley, Reading, MA, 1992.
155. H. Ritter and K. Schulten. On the stationary state of Kohonen's self-organizing sensory mapping. *Biological Cybernetics*, 54:99–106, 1986.
156. H. Robbins and S. Monro. A stochastic approximation method. *Annals of Mathematical Statistics*, 22:400–407, 1951.
157. F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.
158. S.T. Roweis and L.K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.
159. S.T. Roweis, L.K. Saul, and G.E. Hinton. Global coordination of local linear models. In T.G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems (NIPS 2001)*, volume 14. MIT Press, Cambridge, MA, 2002.
160. D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning internal representations by error propagation. In D.E. Rumelhart and J.L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1: Foundations. MIT Press, Cambridge, MA, 1986.
161. D.E. Rumelhart, G.E. Hinton, and R.J. Williams. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.
162. D.E. Rumelhart and D. Zipser. Feature discovery by competitive learning. *Cognitive Science*, 9:75–112, 1985.
163. D.E. Rumelhart and D. Zipser. Feature discovery by competitive learning. In D.E. Rumelhart and J.L. McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, pages 151–193. MIT Press, Cambridge, MA, 1986.
164. M. Saerens, F. Fouss, L. Yen, and P. Dupont. The principal components analysis of a graph, and its relationships to spectral clustering. In *Proceddings of the 15th European Conference on Machine Learning (ECML 2004)*, volume 3201 of *Lecture notes in Artificial Intelligence*, pages 371–383, Pisa, Italy, 2004.
165. J.W. Sammon. A nonlinear mapping algorithm for data structure analysis. *IEEE Transactions on Computers*, CC-18(5):401–409, 1969.
166. L.K. Saul and S.T. Roweis. Think globally, fit locally: Unsupervised learning of nonlinear manifolds. *Journal of Machine Learning Research*, 4:119–155, June 2003.
167. B. Schölkopf, A. Smola, and K.-R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10:1299–1319, 1998. Also available as technical report 44 at the Max Planck Institute for Biological Cybernetics, Tübingen, Germany, December 1996.

168. G. Schwartz. Estimating the dimension of a model. *Annals of Statistics*, 6:497–511, 1978.
169. D.W. Scott. *Multivariate Density Estimation: Theory, Practice and Visualization.* Wiley Series in Probability and Mathematical Statistics. John Wiley & Sons, New York, 1992.
170. D.W. Scott and J.R. Thompson. Probability density estimation in higher dimensions. In J.R. Gentle, editor, *Proceedings of the Fifteenth Symposium on the Interface*, pages 173–179. Elsevier Science Publishers, B.V., North-Holland, 1983.
171. R.N. Shepard. The analysis of proximities: Multidimensional scaling with an unknown distance function (parts 1 and 2). *Psychometrika*, 27:125–140, 219–249, 1962.
172. J. Shi and J. Malik. Normalized cuts and image segmentation. In *Proceedings IEEE International Conference on Computer Vision and Pattern Recognition*, pages 731–737, 1997.
173. J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8):888–905, 2000.
174. A.L. Smith. The maintenance of uncertainty. Manuscript for the Fermi Summer School, October 1996.
175. M. Spivak. *Calculus on Manifolds: A Modern Approach to Classical Theorems of Advanced Calculus.* Addison-Wesley, Reading, MA, 1965.
176. J.F.M. Svensén. *GTM: The generative topographic mapping.* PhD thesis, Aston University, Aston, UK, April 1998.
177. A. Taleb and C. Jutten. Source separation in postnonlinear mixtures. *IEEE Transactions on Signal Processing*, 47(10):2807–2820, 1999.
178. Y.W. Teh and S.T. Roweis. Automatic alignment of hidden representations. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems (NIPS 2002)*, volume 15. MIT Press, Cambridge, MA, 2003.
179. J.B. Tenenbaum. Mapping a manifold of perceptual observations. In M. Jordan, M. Kearns, and S. Solla, editors, *Advances in Neural Information Processing Systems (NIPS 1997)*, volume 10, pages 682–688. MIT Press, Cambridge, MA, 1998.
180. J.B. Tenenbaum, V. de Silva, and J.C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, December 2000.
181. P.C. Teo, G. Sapiro, and B.A. Wandell. Creating connected representations of cortical gray matter for functional MRI visualization. *IEEE Transactions on Medical Imaging*, 16(6):852–863, 1997.
182. W.S. Torgerson. Multidimensional scaling, I: Theory and method. *Psychometrika*, 17:401–419, 1952.
183. S. Usui, S. Nakauchi, and M. Nakano. Internal colour representation acquired by a five-layer neural network. In T. Kohonen, K. Makisara, O. Simula, and J. Kangas, editors, *Artificial Neural Networks.* Elsevier Science Publishers, B.V., North-Holland, 1991.
184. L. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38:49–95, March 1996.
185. J. Venna and S. Kaski. Neighborhood preservation in nonlinear projection methods: An experimental study. In G. Dorffner, H. Bischof, and K. Hornik,

editors, *Proceedings of ICANN 2001, International Conference on Artificial Neural Networks*, pages 485–491. Springer, Berlin, 2001.

186. J. Venna and S. Kaski. Local multidimensional scaling with controlled tradeoff between trustworthiness and continuity. In *Proceedings of the 5th Workshop on Self-Organizing Maps (WSOM'05)*, pages 695–702. Paris, September 2005.
187. J. Venna and S. Kaski. Local multidimensional scaling. *Neural Networks*, 19:889–899, 2006.
188. J. Venna and S. Kaski. Visualizing gene interaction graphs with local multidimensional scaling. In M. Verleysen, editor, *Proceedings of ESANN 2006, 14th European Symposium on Artificial Neural Networks*, pages 557–562. d-side, Bruges, Belgium, April 2006.
189. J.J. Verbeek, N. Vlassis, and B. Kröse. Coordinating mixtures of probabilistic principal component analyzers. Technical Report IAS-UVA-02-01, Computer Science Institute, University of Amsterdam, Amsterdam, February 2002.
190. T. Villman, R. Der, M. Hermann, and T.M. Martinetz. Topology preservation in self-organizing maps: exact definition and measurement. *IEEE Transactions on Neural Networks*, 8:256–266, 1997.
191. C. von der Malsburg. Self-organization of orientation sensitive cells in the striate cortex. *Kybernetik*, 14:85–100, 1973.
192. B.A. Wandell, S. Chial, and B.T. Backus. Visualization and measurement of the cortical surface. *Journal of Cognitive Neuroscience*, 12:739–752, 2000.
193. B.A. Wandell and R.F. Dougherty. Computational neuroimaging: Maps and tracts in the human brain. In B.E. Rogowitz, T.N. Pappas, and S.J. Daly, editors, *Human Vision and Electronic Imaging XI*, volume 6057 of *Proceedings of the SPIE*, pages 1–12, February 2006.
194. E.J. Wegman. Hyperdimensional data analysis using parallel coordinates. *Journal of the American Statistical Association*, 85(411):664–675, September 1990.
195. K.Q. Weinberger, B.D. Packer, and L.K. Saul. Unsupervised learning of image manifolds by semidefinite programming. In *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, Barbados, January 2005.
196. K.Q. Weinberger and L.K. Saul. Unsupervised learning of image manifolds by semidefinite programming. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR-04)*, volume 2, pages 988–995, Washington, DC, 2004.
197. K.Q. Weinberger and L.K. Saul. Unsupervised learning of image manifolds by semidefinite programming. *International Journal of Computer Vision*, 70(1):77–90, 2006. In Special Issue: Computer Vision and Pattern Recognition-CVPR 2004 Guest Editor(s): A. Bobick, R. Chellappa, L. Davis.
198. K.Q. Weinberger, F. Sha, and L.K. Saul. Learning a kernel matrix for nonlinear dimensionality reduction. In *Proceedings of the Twenty-First International Conference on Machine Learning (ICML-04)*, pages 839–846, Banff, Canada, 2004.
199. Y. Weiss. Segmentation using eigenvectors: A unifying view. In *Proceedings IEEE International Conference on Computer Vision*, pages 975–982, 1999.
200. E.W. Weisstein. Mathworld. A Wolfram web-resource – `http://mathworld.wolfram.com/`.
201. P.J. Werbos. *Beyond regression: New tools for prediction and analysis in the behavioral sciences.* PhD thesis, Harvard University, 1974.

202. H. Whitney. Differentiable manifolds. *Annals of Mathematics*, 37(3):645–680, 1936.
203. C.K.I. Williams. On a connection between Kernel PCA and metric multidimensional scaling. In T.K. Leen, T.G. Diettrich, and V. Tresp, editors, *Advances in Neural Information Processing Systems (NIPS 2000)*, volume 13, pages 675–681. MIT Press, Cambridge, MA, 2001.
204. L. Xiao, J. Sun, and S. Boyd. A duality view of spectral methods for dimensionality reduction. In *Proceedings of the 23rd International Conference on Machine Learning*. Pittsburg, PA, 2006.
205. H.H. Yang, S. Amari, and A. Cichocki. Information-theoretic approach to blind separation of sources in non-linear mixtures. *Signal Processing*, 64(3):291–300, 1998.
206. L. Yen, D. Vanvyve, F. Wouters, F. Fouss, M. Verleysen, and M. Saerens. Clustering using a random-walk based distance measure. In M. Verleysen, editor, *Proceedings of ESANN 2005, 13th European Symposium on Artificial Neural Networks*, pages 317–324, Bruges, Belgium, April 2005. d-side.
207. P.N. Yianilos. Data structures and algorithms for nearest neighbor search in general metric spaces. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, 1993.
208. G. Young and A.S. Householder. Discussion of a set of points in terms of their mutual distances. *Psychometrika*, 3:19–22, 1938.
209. H. Zha and Z. Zhang. Isometric embedding and continuum ISOMAP. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003)*. Washington DC, August 2003.
210. U. Zwick. Exact and approximate distances in graphs — a survey. In *Proceedings of the 9th ESA*, pages 33–48. 2001. Updated version available on Internet.

Index