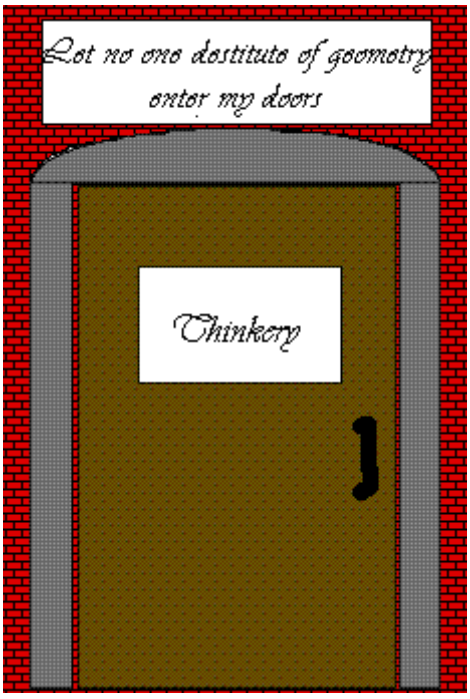
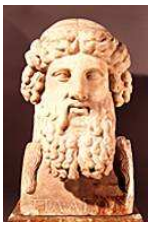
Basics of Geometrical Data-Analysis

Algorithms for Vectorial Pattern-Analysis

N. Laskaris



''A circle, for instance,
is defined as a plane figure
composed of a series of points,
all of which are equidistant
from a given point.
No one has ever actually seen
such a figure, however''
Plato, 427-347 BC



Introduction

The term ‘‘pattern’’, currently, encompasses the notion of a variety of data-forms the machines have to tackle with. Despite the fact that in early days it was used mostly for pictorial information, i.e. 2D-signals, now the same term stands almost for any output from a data-source. For instance, any digital-signal can be considered as an 1D-pattern, a grey-scale image as a 2D-patterm, a video-sequence as a (temporal) multi-dimensional pattern etc.

Here we will present a general purpose framework for dealing with patterns and discuss simple algorithms with a wide-range of applications (from novelty-detection and prototyping in databases to the full organization of a library of patterns). The main characteristic of the framework and simultaneously its great benefit is its Geometrical character. This enables the direct conceptualization of the employed ideas and promotes the easy understanding of the described algorithmic steps.

Given an ensemble of N (general character) patterns, a p-dimensional vector $\mathbf{x_i}$, i=1,2,…N

$$\mathbf{x_i} = [\, x_i(1) \;\; x_i(2) \;\ldots\ldots\; x_i(p) \;]$$

is extracted from each one. With this step, which is known as the ***feature-extraction*** step, the set of patterns is represented by a set of row-vectors.

The N vectors are gathered in the so-called Data-Matrix $\mathbf{X^{data}}$

$$X^{data}_{[N\,x\,p]} = \begin{pmatrix} \mathbf{x_1} \\ \mathbf{x_2} \\ \cdot \\ \cdot \\ \cdot \\ \mathbf{x_N} \end{pmatrix} = \begin{pmatrix} [x_1(1), x_1(2) \ldots\ldots, x_1(p)] \\ [x_2(1), x_2(2) \ldots\ldots, x_2(p)] \\ \cdot \\ \cdot \\ \cdot \\ [x_N(1), x_N(2) \ldots\ldots, x_N(p)] \end{pmatrix} = \begin{pmatrix} x_{11}, x_{12} \ldots\ldots, x_{1p} \\ x_{21}, x_{22} \ldots\ldots, x_{2p} \\ \cdot \\ \cdot \\ \cdot \\ x_{N1}, x_{N2} \ldots\ldots, x_{Np} \end{pmatrix} = \begin{matrix} \text{Pattern number} \\ \end{matrix} \begin{pmatrix} x_{11}, x_{21} \ldots\ldots, x_{1p} \\ x_{21}, x_{22} \ldots\ldots, x_{2p} \\ \cdot \\ \cdot \\ \cdot \\ x_{N1}, x_{N2} \ldots\ldots, x_{Np} \end{pmatrix}$$

Two simple transformations of the above matrix are usually employed:

(i) <u>standardization</u> of each one of the p variates (after subtraction of its mean) is performed via a normalization step in which the variate is divided with its *standard deviation* (*std*). This is useful when the variates have different scales: e.g. one variate is amplitude and the other is time.

- ***Whitening*** based on PCA is a more advanced standardization procedure that aims, also, at decorrelating the variates simultaneously.

(ii) <u>normalization</u> of each one of the N vectors, by dividing with its norm,
i.e. replacement of $\mathbf{x_i}$ with $\mathbf{X_i = x_i / ||x_i||}$

where $||\mathbf{x_i}|| = [\, x_i(1)^2 + x_i(2)^2 + \ldots . x_i(p)^2 \,]^{1/2}$

This transformation is useful in order to highlight shape similarities during the subsequent computation of Euclidean distances between patterns (see below).

The geometrical consideration, according to which the patterns are represented by points (i.e. the end-tails of the corresponding vectors) in a multidimensional space, is very useful in order to conceptualize morphological relationships between patterns, to search for natural groupings inside the sample of patterns, etc. The key idea is that similar patterns are mapped onto nearby points.

Distance matrix

The very first thing, that one can do is to measure the geometrical distance between two vectors in order to quantify (inversely) the similarity between the corresponding two patterns. A small distance means great similarity between the two patterns and this can be interpreted as common signal/information content.

The most common (but not always the most efficient) way to measure dissimilarity is through the Euclidean distance:

$$\mathbf{d(x_1,x_2)} = ||\mathbf{x_1 - x_2}|| = [\, (x_1(1)- x_2(1))^2 + (x_1(2) - x_2(2))^2 + \ldots . + (x_1(p)-x_2(p))^2 \,]^{1/2}$$

For computational considerations, usually its squared form is utilized, i.e.

$$\mathbf{d(x_1,x_2)} = ||\mathbf{x_1 - x_2}||^2$$

For an ensemble of N patterns $\{\mathbf{x_i}\}_{i=1:N}$, all the pairwise distances are gathered in the so-called (NxN) ***distance matrix*** $\mathbf{D_{[NxN]}}$

$$\mathbf{D}_{[N\,x\,p]} = \begin{pmatrix} 0 & d(1,2) & d(1,3) & \cdots & d(1,N) \\ d(2,1) & 0 & d(2,3) & \cdots & d(2,N) \\ d(3,1) & d(3,2) & 0 & \cdots & d(3,N) \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ d(N,1) & d(N,2) & d(N,3) & \cdots & 0 \end{pmatrix} = \begin{array}{c} \\ \text{Pattern number} \end{array} \overset{\text{Pattern number}}{\begin{pmatrix} 0 & D_{12} & D_{13} & \cdots & D_{1N} \\ D_{21} & 0 & D_{23} & \cdots & D_{2N} \\ D_{31} & D_{32} & 0 & \cdots & D_{3N} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ D_{N1} & D_{N2} & D_{N3} & \cdots & 0 \end{pmatrix}}$$

A **fast computation** of this (symmetric) matrix, enabling e.g. *optical implementation*, is given via the following matrix operations:

$$\mathbf{D} = \text{diag}(\mathbf{A})\,\mathbf{E} + \mathbf{E}\,\text{diag}(\mathbf{A}) - 2\mathbf{A} \qquad \mathbf{(1)}$$

where

$$\mathbf{A} = \mathbf{X}\mathbf{X}^{\mathbf{T}}, \quad \mathbf{E}_{(N\,x\,N)} = \begin{pmatrix} 11\ldots1 \\ 11\ldots1 \end{pmatrix}, \quad \mathbf{X}_{(N\,x\,p)} = \mathbf{X}^{\mathbf{data}} = [\,\mathbf{x}_1 \vdots \mathbf{x}_2 \vdots \ldots \mathbf{x}_N\,]$$

Notice

If the normalized versions (X_i) of the patterns x_i has been used in the Data matrix, then the corresponding pairwise Euclidean distances becomes

$$d(X_i,X_j) = 2\,(1 - \rho(x_i, x_j))$$

where $\rho(x_i, x_j)$ is the ***correlation coefficient*** between the two vectors, which is defined as:

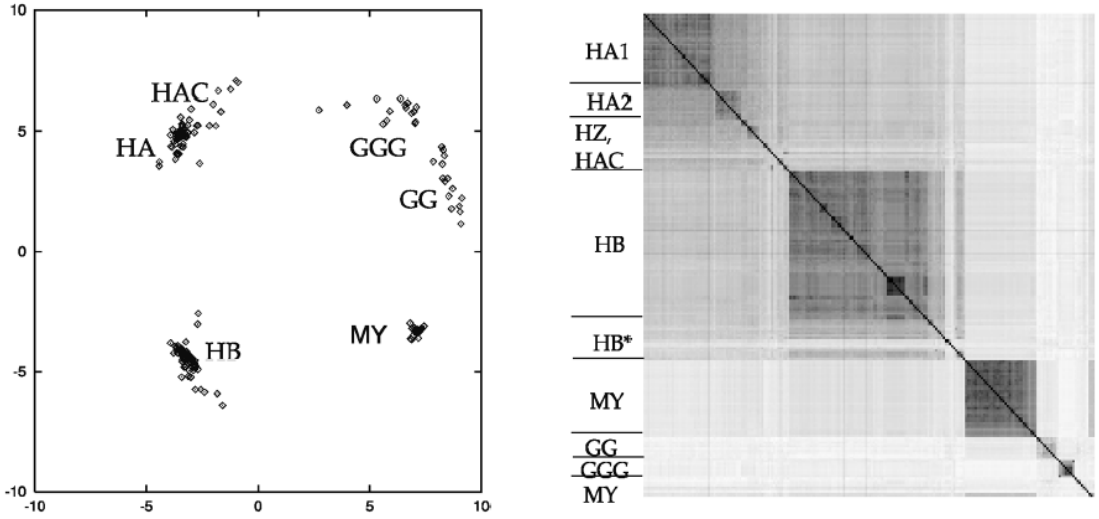
$$\rho(\mathbf{x_i}, \mathbf{x_j}) = \mathbf{x_i} \bullet \mathbf{x_j} \;/\; (||\mathbf{x_1}||^2\,||\mathbf{x_2}||^2) = \mathbf{X_i} \bullet \mathbf{X_j}$$

where $\bullet$ defines inner product, i.e. in form of matrix operations : $\mathbf{x_i}\,\mathbf{x_j}^{\mathbf{T}}$ (with the superscript ‘‘T” denoting the transpose operation).

Remark:

The Correlation coefficient between time series-waveforms (when considering 1-D signals as patterns), usually referred to as a “shape similarity”, is known to expresses the synchronization between them.

An insight to the structural information contained by the Distance-matrix can be obtained via a simple visualization-scheme. The entries of matrix **D** are treated as grey-valued pixels and the layout produced this way is indicative of the presence of any structure in the data. This procedure is an easily-implemented technique for unmasking possible outliers (the corresponding rows/columns are white stripes in the produced layout). An example of a point-sample and the corresponding distance-matrix can be seen below (taken from actual protein-sequence data).



Relating topological descriptors of point sets with the data.

The description of a set of patterns, through the topology of their representing points can lead to simple descriptors that have a ready geometric interpretation without loosing the connection with conventional approach for studying the data (e.g. statistical analysis). Geometrical concepts like the 'local point-density' or the outline/skeleton of a point-swarm can be utilized in building tools for understanding and handling the multidimensional data.

In the sequel, we are –first– considering the interpoint distances and the gravitational centre of a point set.

A simple geometrical descriptor of a point set is its ***dispersion J***, which expresses the compactness of the point set as the average distance from the geometrical mean.

$$J(\{\mathbf{x_i}\}_{i=1:N}) = 1/(N-1)\cdot\sum_{i=1}^{N}\|\mathbf{x_i}-\mathbf{x_{ave}}\|^2 \quad , \qquad \mathbf{x_{ave}} = {}^{1}\!/_{N}\cdot\sum_{1=1}^{N}\mathbf{x_i}$$

note: it is the p-dimensional analogous of the standard deviation of a set of scalars (a set with unidimensional members, i.e. numbers).

It can be shown that dispersion can be expressed as a summation of pairwise distances (a trick that will be justified later) :

$$J(\{\mathbf{x_i}\}_{i=1:N}) = 1/2N(N-1) \cdot \sum_{i=1}^{N}\sum_{j=1}^{N} \left\|\mathbf{x_i} - \mathbf{x_j}\right\|^2$$

and therefore estimated via the following simple matrix operation:

$$J(\{\mathbf{x_i}\}_{i=1:N}) = \frac{1}{2N(N-1)} \cdot \quad \mathbf{u} \cdot \mathbf{D} \cdot \mathbf{u^T} \quad , \qquad u_{[1xN]} = [1,1,.....1]$$

The following “rules” are motivated by the geometrical interpretation of the computed quantities:

(i) Between two sets of patterns, and assuming common underlying signal-source, the more reliable set is the one of smallest dispersion. In other words, the dispersion is a measure of ‘noise’ in the data.

(ii) The contribution of the i-th vector to the total dispersion (and correspondingly to the ‘noise’ of the data) is the sum of its distance to the rest of the points:

$$dist(x_i) = d(i,1) + d(i,2) + \ldots + d(i,N)$$

This can serve as a simple gauge for unmasking outlying points, i.e points that lie far away from the majority of them and therefore correspond to unusual patterns.

(iii) Conversely, the notion of **Vector Median** can be introduced. This is the vector with the shortest aggregate distance (from the rest vectors in the point-sample).

Unmasking Outliers

Using simple functionals with arguments the pairwise distances, we can built mappings that are informative about the ‘‘distinctiveness’’ of the corresponding patterns. The idea inherent in many ***vector-ordering*** schemes is to map each vector to a scalar, to locate the vectors with images lying at the extremes of the obtained scalar distributions, to identify the corresponding vectors and make a final judgment about the corresponding patters.
The simplest mapping can be built using the aggregate distance, i.e. using the elements from the corresponding row of the distance matrix

$$x_i \mapsto dist(x_i) = d(i,1) + d(i,2) + \ldots + d(i,N)$$

This can serve as a simple gauge for unmasking outlying points, i.e points that can correspond to extremely noisy patterns or extremely interesting patterns. In the latter case the task is known as ***novelty detection*** and used in many quality-control tasks.

Usually, the estimated scalars are ordered

$$\left[\ \mathrm{dist}_1\ \mathrm{dist}_2\ \ \mathrm{dist}_3\ \ ....\ \ \mathrm{dist}_N\ \right] \xrightarrow{\text{ordering}} \left[\ \mathrm{dist}_{[1]}\ \mathrm{dist}_{[2]}\ \ \mathrm{dist}_{[3]}\ \ ....\ \ \mathrm{dist}_{[N]}\ \right]$$

and this ordering defines the ordering of the corresponding vectors (and consequently of the patterns they are associated with)

$$\left[x_1\, x_2\, x_3 .... x_N\ \right] \xrightarrow{\text{Reducedordering}} \left[x_{[1]}\, x_{[2]}\, x_{[3]} .... x_{[N]}\ \right]$$

In this way, a ranked list of patterns has been identified in which the elements that deserves further consideration (due to their non-typicality) lie at one of the two ends.

- As an alternative for unmasking outlying points, the following measure, can be utilized:

$$\mathrm{dist}(x_i) = \min_j\left(\left\{ d(i,j) \right\}_{j=1:N,\, j\neq i} \right)$$

While the above described ordering procedure can easily spot non-typical patterns, the identification of the most typical ones require more delicate procedures, the majority of which fall in the mainstream of Clustering literature.

Cluster Analysis

Cluster Analysis (*CA*) deals with the identification of natural groupings in an ensemble of objects. In the case of a point set, CA searches for homogeneous subsets. The most common categorization of CA algorithms classifies them into *partition*, *hierarchical* and *graph-theoretic* ones. In the following, we discuss a few prototypical algorithms which are belonging to the first two categories and postpone the discussion of graph-theoretic approaches for a later part of these notes in which ***exploratory-data*** analysis is treated in some details.

Hierarchical Clustering

The main characteristics of these algorithms are that they work with a dissimilarity matrix without using the patterns themselves and that they have a deterministic character (in the sense that they produce always the same output, in contrast to the partition algorithm that the resulting grouping depends on their initialization). In the sequel, we outline the most common among them, known as the **Single-linkage algorithm :**

Given the dissimilarity matrix (here, the matrix **D**), the process begin by pairing the two points k and l with the smallest distance. The rows and columns in D corresponding to k and l points are deleted. A new row (and the corresponding column) is inserted. It contains the distances of the first cluster (k,l) to the remaining N-2 points.

These distances are found from the rule:

$$D_{(kl)i} = \min (D_{ki} , D_{li}), \; i \neq k,l$$

Using the new [N-1 x N-1] dissimilarity matrix, we identify the next two points with the smallest pairwise distance. During this procedure the pair (k,l) is treated a single point and can be paired with one of the N-2 points. Next, a new [N-2 x N-2] distance matrix is derived and the procedure continues until all points have been grouped into a hierarchy of clusters. This hierarchy is a sequence of nested point sets and is represented as a function of the pairing distance. The visualization of this hierarchy through *dendrograms* enables the final user-depended grouping.

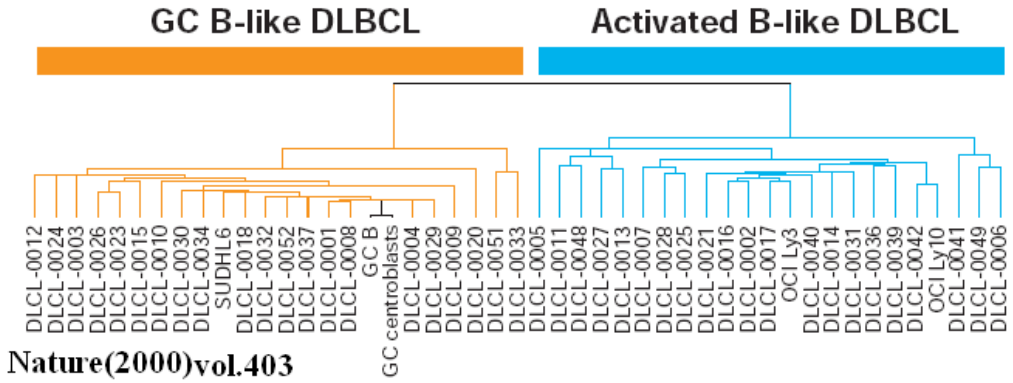


Fig 3 Discovery of DLBCL subtypes by gene expression profiling.

Partitioning Clustering Algorithms

The search for clusters -in the case of ***partitioning algorithms***- includes some algorithmic steps that are directed to the minimization (maximization) of an objective (cost) function that expresses the separability (compactness) of the produced groups.

The **partition matrix U** is used to tabulate the results of the CA. It's a [CxN] matrix, with each row devoted to one of the C produced clusters.

$$U_{[CxN]} = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_C \end{bmatrix} = \begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1N} \\ u_{21} & u_{22} & \cdots & u_{21} \\ \vdots & \vdots & \vdots & \vdots \\ u_{C1} & u_{C2} & & u_{CN} \end{bmatrix}, \qquad \sum_{j=1:C} u_{ij} = 1, \quad \sum_{\substack{j=1:C \\ i=1:N}} u_{ij} = N$$

The *indicator function* u_{ji} takes the value **1** if the point $\mathbf{x_i}$ belongs to the j-th cluster; otherwise is set to **0** (crisp clustering; in fuzzy counterpart u_{ji} simply takes a value in the range [0,1]).

In the case of ***C-means*** algorithm, the objective function that is minimized is the total intra-cluster dispersion:

$$E = \sum_{j=1:C} \sum_{i=1:N} u_{ji} \left\| \mathrm{x}_i - \mathrm{o}_j \right\|^2 , \quad \mathrm{o}_j = \frac{1}{\sum_{i=1:N} u_{ji}} \sum_{i=1:N} u_{ji} \mathrm{x}_i$$

The cores of the C clusters are the corresponding geometrical centers (means); this explains the name of the algorithm.

It easy to see that, since this objective is the sum of the individual subset dispersions, the algorithm "works" at a splitting direction so at to reduce the initially estimated noise power of the overall set.

In matrix operation, the above cost function reads:

$$E = \sum_{j=1:C} \frac{1}{2 \cdot pop_j} \, \mathrm{u}_j \cdot \mathrm{D} \cdot \mathrm{u}_j^T , \quad pop_j = \sum_{i=1:N} u_{ji}$$

where **D** is the ensemble interpoint distance and pop_j is the population of each cluster.

This equation can be written in an even more compact form, after proper scaling of the u_j with the corresponding population : E= trace($\mathbf{UDU^T}$)

Note: *Which calls for a "physical interpretation" of the off diagonal elements of the matrices product inside the trace operation: the summation of the off-diagonal elements of this product expresses the average inter-cluster separability*.

Remarks:

i) Since CA algorithms always result to grouped data, a critical issue that always arises is if their function really contributes to the understanding of the true point distribution. A way to justify this is the comparison of measure E with the corresponding dispersion for the overall point set dispersion.

ii) To alleviate the problem of initialization and not sufficient convergence, usually the iterative algorithms (like the C-means) are applied a few times and the best partition matrix is the final outcome.

iii) An intriguing aspect is "how many clusters there are" in the point set. A simple strategy for estimating the number of clusters C, is to apply the algorithm for increasing value of C, and by plotting the corresponding values of E as function of C to decide the critical number C_0. Notice that E is by default a monotonically decreasing function of C, with absolute minimum C=N, i.e each point to its one cluster.

iv) Outlying points tend to obscure the convergence and the accuracy of the resulting partition. It is suggested to be isolated from the beginning.

v) The objective function has been modified many times in the Pattern Recognition literature, e.g. so as to bias the creation of highly populated clusters.

Subtractive Clustering

An efficient technique known as ***mountain-clustering*** has been introduced recently [Yager, 1994] for delineating cores in a multimodal point distribution. It is an iterative scheme that employs ***detection*** of the most significant mode and ***subtraction*** of the subset of points that are coming from the certain mode.

In our case this technique has been modified as follows:

(i) For the detection of the dominant mode the technique of *Potential Function* is used so as to construct a mountain, the height of which is proportional to the local point density. An estimate of the local point density is assigned to each point $\mathbf{x_i}$, through the relation

$$PD(\mathbf{x}_i) = \frac{1}{(2\pi)^{P/2} r_o^P N} \sum_{j=1}^{N} exp[\frac{-\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2r_o^2}]$$

where r_o, known as radius of influence, shapes the influence of each point on the rest and has to be estimated from "noisy conditions".

Remark: **Notice that the PD- quantity can be estimated, in a straightforward manner, using the elements of distance matrix.*

The point of the set that lies closer to the dominant mode is identified as the point $\mathbf{x_{max}}$ of maximum local point density.

(ii) For the refinement of the dominant-mode estimation, the points in the vicinity of $\mathbf{x_{max}}$ are averaged. To this end, each point $\mathbf{x_i}$ in the point-set is ordered according to its distance $d(\mathbf{x_i}, \mathbf{x_{max}})$, i.e. the closer to the $\mathbf{x_{max}}$ the point is, the lower its rank [i] will be. A portion of the lower ranked points will be averaged

$$\bar{\mathbf{x}}_{\mathbf{sel}} = \frac{1}{\mathbf{j_0}} \sum_{\mathbf{i=[1]}}^{\mathbf{[j_0]}} \mathbf{x_i}$$

The definition of the number j_o can be adaptive and provides the optimal number of nearest neighbours of $\mathbf{x_{max}}$ that have to be subtracted. This subset is removed and the procedure is repeated from the detection step.

Multidimesional Scaling

Many questions that arise during the execution of an CA algorithm, like “how many clusters ?” or “are there any outliers in the sample”, have -by far- an easy answer in the case of univariate / bivariate observations, i.e. points on the real line/plane. It is remarkable the human gift for pattern recognition-tasks like determining modes in a point distribution and recognizing trends (e.g. attractors, abrupt changes) in the data when these are presented in the form of point-diagrams.

Motivated by this gift, many dimensionality reduction techniques have been introduced as a preprocessing step to (or crude-approximation of) Cluster Analysis. These techniques aim at “projecting” the original p-dimensional point-sample onto a low dimensional space (e.g. PCA, projection pursuit algorithms etc). These techniques work with the original set of variates, trying to extract linear/nonlinear combinations of them that the further analysis could focus on. This turns to be the main disadvantage of them, since in many cases (e.g. data from psychophysics behavioral experiments) the only available information comes in the form of a similarity matrix (i.e. the inverse of a distance matrix), that describes the mutual relationships between the patterns we want to analyze. This led to the development of an important branch of Multivariate Analysis, known as Multi-Dimensional Scaling (MDS).

The definition of MDS is –currently– any procedure that, given a dissimilarity matrix corresponding to a set of patterns, configures points in a low dimensional space (usually 2-D) as images of the patterns in a way that the interpoint distances approximate as much as possible the original pairwise dissimilarities. This results in a 2-D “projection” of the objects, where neighboring relationships/clustering trends are prominent.

An early categorization of MDS algorithms used to classify them into two categories: *metric* and *nonmetric MDS*. As a metric MDS algorithm is referred one that is akin to PCA, i.e. it is applied via eigenvectors analysis and has analytical expression. On the contrary the nonmetric MDS algorithms are iterative in nature and computational demanding, but usually (slightly to moderately) superior to the metric ones.

In the following the classic metric algorithm [Torgerson; 1952,1958 (see [Morrison,1990])] is presented. The output of this algorithm has been proposed as a very good initialization for the nonmetric ones.
The algorithm starts with a transformation of the original dissimilarity matrix; in our case this matrix is the interpoint distance matrix **D** computed for a set of N p-dimensional points**.**

(i) $\mathbf{A_{[NxN]}} = -\mathbf{D_{[NxN]}}$

(ii) the elements of **A** are doubly centered about their row and column means resulting to matrix $\mathbf{B_{[NxN]}}$ with elements:

$B_{ij} = A_{ij} - \underline{A}_{i.} - \underline{A}_{.j} + \underline{A}_{..}$

(iii) The first r characteristic roots $l_1, l_2, \ldots l_r$ and their associated vectors $\mathbf{v_{1\,[Nx1]}}, \mathbf{v_2}, \ldots, \mathbf{v_r}$

are extracted from **B**.

(iv) The vectors are normalized so that $\mathbf{v_i}^T \mathbf{v_i} = l_i$ and gathered in a [N x r] matrix

$$V_{[Nxr]}=[\ \mathbf{v_1}\ \mathbf{v_2}\ \ldots..\ \mathbf{v_r}\ ]$$

(v) The i-th row of this matrix contains the coordinates of the i-th point in the new r-dimensional space (r is usually, but not necessary, 2) :

$$\mathbf{V}_{[N\,x\,r]} = X^{\mathbf{data}}_{[N\,x\,r]} = \begin{pmatrix} \chi_1 \\ \chi_2 \\ \cdot \\ \cdot \\ \cdot \\ \chi_N \end{pmatrix} = \begin{pmatrix} \chi_{11},\chi_{12}\cdots\cdots,\chi_{1r} \\ \chi_{21},\chi_{22}\cdots\cdots,\chi_{2r} \\ \cdot \\ \cdot \\ \cdot \\ \chi_{N1},\chi_{N2}\cdots\cdots,\chi_{Nr} \end{pmatrix} = \begin{matrix} \text{pattern number} \\ \\ \\ \\ \\ \\ \end{matrix} \begin{pmatrix} \chi_{11},\chi_{12}\cdots\cdots,\chi_{1r} \\ \chi_{21},\chi_{22}\cdots\cdots,\chi_{2r} \\ \cdot \\ \cdot \\ \cdot \\ \chi_{N1},\chi_{N2}\cdots\cdots,\chi_{Nr} \end{pmatrix}$$

(vi) A measure of map credibility, regarding its ability to reflect the original structure is given by the normalised total discrepancy

$$\text{Streess} = \sum_{i<j} \left| \sqrt{D_{ij}} - \sqrt{\Delta_{ij}} \right| \Big/ \sum_{i<j} \sqrt{D_{ij}}$$

where $\boldsymbol{\Delta}$ is the matrix of interpoint distances $\Delta_{ij} = ||\boldsymbol{\chi_i} - \boldsymbol{\chi_j}||^2$ in the new space (computed from eq.(1) using as data matrix the matrix $\mathbf{V_{[Nxr]}}$)

Remarks:

Possible outliers in the set tend to “dominate” the projection. A refined image can be obtained after their isolation and removal.

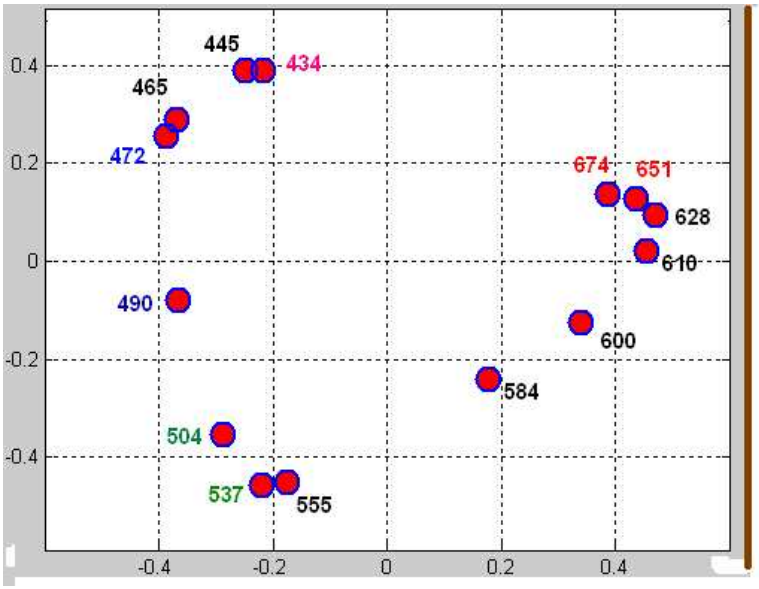
A classical example:

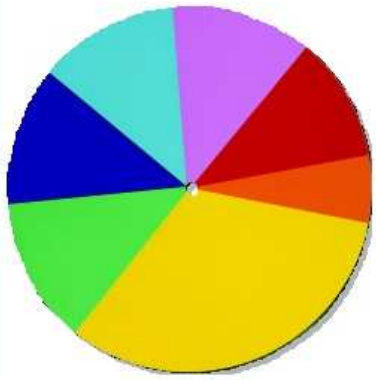
With standard psychophysical experimental procedures, the perceptual similarity (PS) between colors was estimated and tabulated as follows. The 14 entries correspond to 14 different 'hues' with wavelengths :

Wavelength = [434 445 465 472 490 504 537 555 584 600 610 628 651 674]

Grossly speaking a reddish hue corresponds to the wavelength of 674, while a bluish hue to 472, etc…

0	0.8600	0.4200	0.4200	0.1800	0.0600	0.0700	0.0400	0.0200	0.0700	0.0900	0.1200	0.1300	0.1600
0.8600	0	0.5000	0.4400	0.2200	0.0900	0.0700	0.0700	0.0200	0.0400	0.0700	0.1100	0.1300	0.1400
0.4200	0.5000	0	0.8100	0.4700	0.1700	0.1000	0.0800	0.0200	0.0100	0.0200	0.0100	0.0500	0.0300
0.4200	0.4400	0.8100	0	0.5400	0.2500	0.1000	0.0900	0.0200	0.0100	0	0.0100	0.0200	0.0400
0.1800	0.2200	0.4700	0.5400	0	0.6100	0.3100	0.2600	0.0700	0.0200	0.0200	0.0100	0.0200	0
0.0600	0.0900	0.1700	0.2500	0.6100	0	0.6200	0.4500	0.1400	0.0800	0.0200	0.0200	0.0200	0.0100
0.0700	0.0700	0.1000	0.1000	0.3100	0.6200	0	0.7300	0.2200	0.1400	0.0500	0.0200	0.0200	0
0.0400	0.0700	0.0800	0.0900	0.2600	0.4500	0.7300	0	0.3300	0.1900	0.0400	0.0300	0.0200	0.0200
0.0200	0.0200	0.0200	0.0200	0.0700	0.1400	0.2200	0.3300	0	0.5800	0.3700	0.2700	0.2000	0.2300
0.0700	0.0400	0.0100	0.0100	0.0200	0.0800	0.1400	0.1900	0.5800	0	0.7400	0.5000	0.4100	0.2800
0.0900	0.0700	0.0200	0	0.0200	0.0200	0.0500	0.0400	0.3700	0.7400	0	0.7600	0.6200	0.5500
0.1200	0.1100	0.0100	0.0100	0.0100	0.0200	0.0200	0.0300	0.2700	0.5000	0.7600	0	0.8500	0.6800
0.1300	0.1300	0.0500	0.0200	0.0200	0.0200	0.0200	0.0200	0.2000	0.4100	0.6200	0.8500	0	0.7600
0.1600	0.1400	0.0300	0.0400	0	0.0100	0	0.0200	0.2300	0.2800	0.5500	0.6800	0.7600	0





The above point diagram was produced by applying the classical-MDS algorithm to the distance matrix with entries d(i,j)=1-PS(i,j), i,j=1:14. The 'homeomorphism' of this plot with the well-known color-disk shown on the right is remarkable.

(Data) Manifold Learning

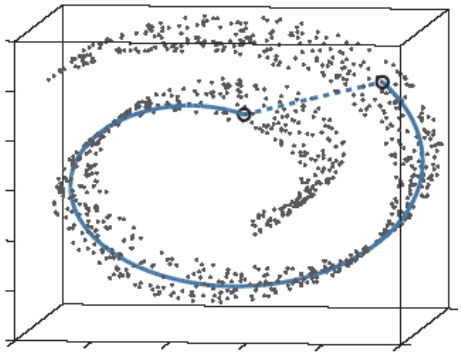
The last three years -and especially after the appearance of two publications (listed below) in the same issue of Science magazine in Dec,2000- the interest about manifolds has been renewed and extended well beyond the mathematicians' community (e.g. Riemannian manifold). Nowadays, ***Manifold-Learning*** has become an individual scientific branch in which data-analysts, from different research directions, contribute and interact.

A well-informed Web-site is : http://www.cse.msu.edu/~lawhiu/manifold/

where the two Science-paper can be found and downloaded.

(1) J.B. Tenenbaum, V. de Silva and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction . Science, vol. 290, pp. 2319--2323, 2000
(2) S. T. Roweis and L. K. Saul. Nonlinear Dimensionality Reduction by Locally Linear Embedding . Science, vol. 290, pp. 2323--2326, 2000

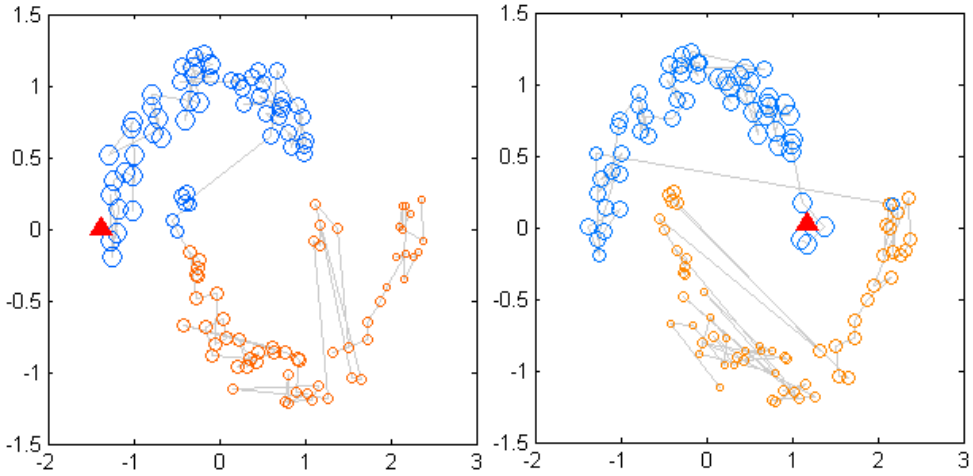
A simple definition of a manifold –well aligned with the spirit of these notes- is this of 'a ***constrained (multidimensional) surface***'. This implies the existence of an ***ambient*** (vector) ***space*** in which the available data lie in a restricted way. The following figure shows the famous 'Swiss-Roll'' which is 2D-surface in a 3D-space (the ambient space).



In the most usual case, the available data are multivariate observations from a high dimensional space (for instance: individual video-frames can be considered as points in a space of num_of_pixels^2 dimensions). This high-dimensionality usually obscures the useful information in the data, and constitutes one of the major component of the 'curse of dimensionality'. ''Less is Better'' is a popular motto within Data-Analysts, who are currently interested in efficient techniques for handling high-dimensional data and the development of methods for data-abstraction and summarization. Visualization-schemes are the most popular, since some insight into the data can be gained, immediately, by the user through low-dimensional plots and graphs.

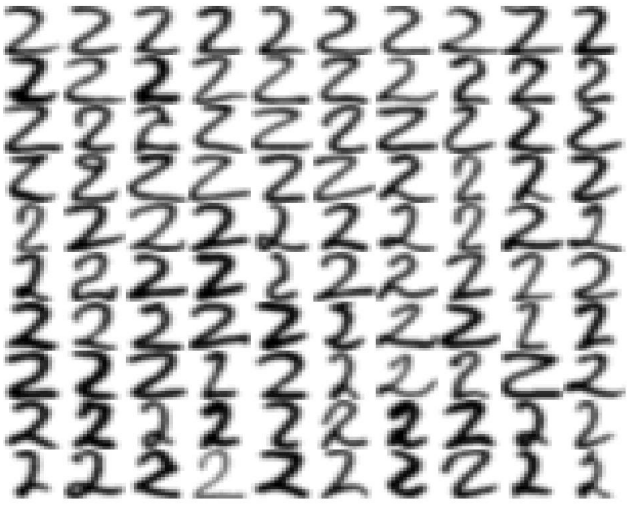
To underline further the need for Manifold-Learning we are including the two 'classical'-examples (borrowed from the above mentioned web-site). This of searching in 'a pair of moon-datasets' and retrieval from a database containing hand-written digits.

These figures show the results of our ranking algorithm on the toy "two moons" data set. In each case the query point is marked by the red triangle. The size of the other points indicates their ranking score, and the connecting lines join the points in the order of their ranking score. Intuitively the ideal solution is to rank all the points in the same "moon" as the query point higher than the points in the other moon.

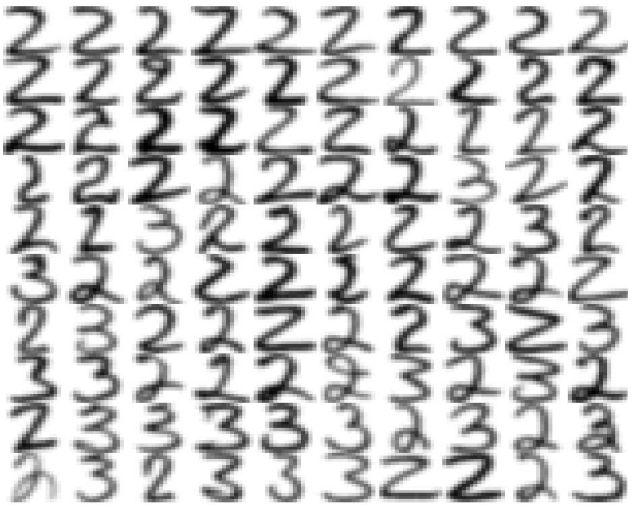


Below are the results on a subset of the USPS data set. In each case the top left-hand image is the query, and the unlabelled data set consisted of 5424 exemplars of the digits 1-6. The 99 top ranked images are shown for (a) data-manifold ranking algorithm and (b) Euclidean distance ranking. Note that (b) contains a larger number of 3's and 2's with knots, subjectively somewhat dissimilar to the query.

(a) Top 100 by data manifold ranking



(b) Top 100 by Euclidean distance



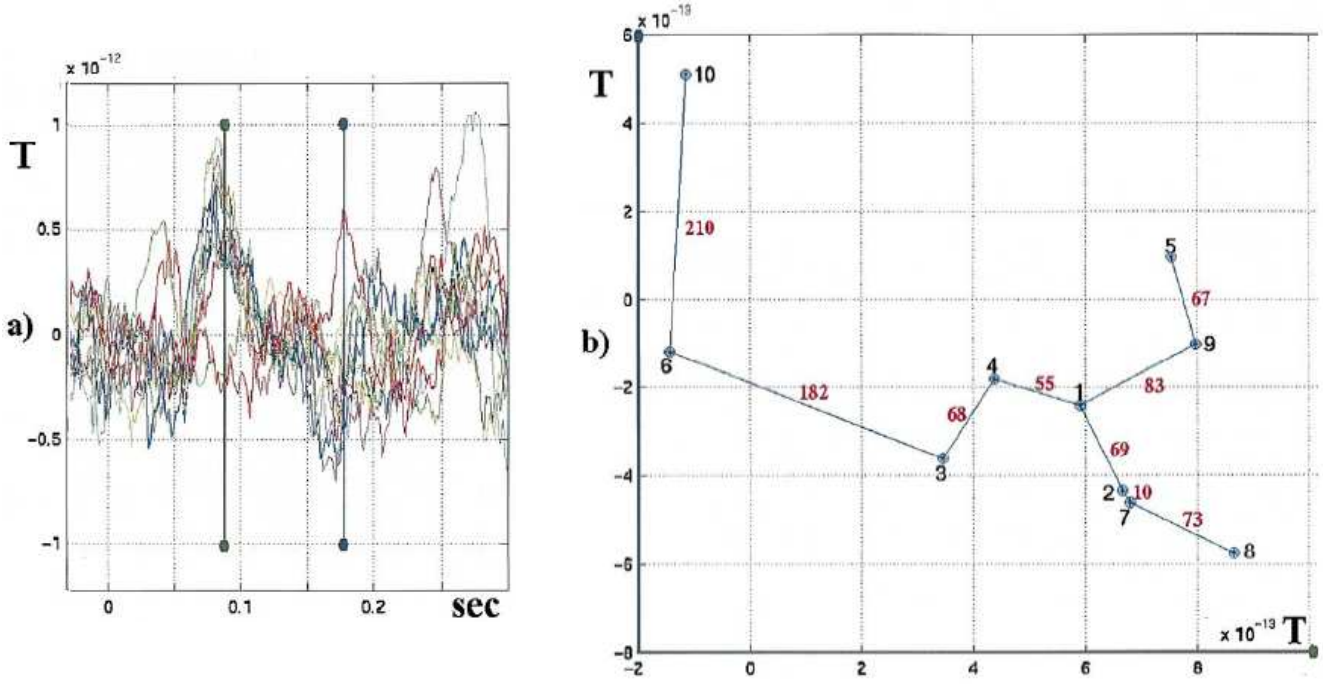
In the sequel, some representative techniques will be presented starting from the ***Minimal-Spanning-Tree-Graph*** related tools for data handling, continuing with the ***ISOMAP*** for visualization of the data and ending with the ***Neural-Qas Vector Quantization*** algorithm for data-abstraction and prototyping.

Minimal Spanning Tree

Graph theory sketches the MST structure with the following definitions. A *graph* is a set of nodes and a set of node pairs called edges. An *edge weighted* graph is a graph with a real number, called weight, assigned to each edge. A *connected graph* has a path between any two distinct nodes. A *Spanning Tree* is a connected graph that includes all the nodes without loops. The MST is the spanning tree of minimum total weight.

When the previous concepts are applied to a set of N points, a node is dedicated to each point and the corresponding pairwise distances (or generalised dissimilarities) are assigned as weights to the formed edges. The MST is the connected graph, emerged from the collection of exactly (N-1) edges, having minimum total length.

In order to demonstrate how the previous abstract graph-theoretic concepts are used to handle the available data, temporal patterns from a real experiment (time-waveforms from magnetic brain-response signals) are used in a simplified 2-dimensional example (see figure, below). With each one of the 10 patterns shown in panel a), a point in $\mathbf{R^2}$ is associated. The 2-dimensional configuration of this point sample is given in panel b). Each of the two axes spanning this reduced space expresses the strength of the Magnetic Field at a time instant in the post-stimulus range. The first (horizontal) axis was selected, by visual inspection, so as to correspond approximately to a time instant where the majority of waveforms present a positive deflection. The second (vertical) axis corresponds to a time instant chosen at random. In this graph the 10 points appear as nodes indexed from **1** to **10**. These indices reflect the physical time order of the corresponding waveforms. The MST appears as a collection of 9 line segments, the *edges*, with sample points as endpoints. The *weight* associated with each edge is also indicated. It is the pairwise Euclidean distance between the corresponding points. A scaling has been applied on these distances such that the smallest of them (2.991 10-14 T) appears on the graph as 10. With such a graph, it is easy to conceptualise the notion of *centrical/prototypical* points and *outliers*. The term *degree* of a node is used to denote the number of edges incident on it. *Centrical* points (e.g.**1**) differ from outliers (e.g. **10**) in terms of degree and weights of the associated edges.



2.2. Applying Graph theory in the ambient space - *Isomap*

The intrinsic geometry that governs the *geodesic manifold* of the point distribution can be emphasized by the incorporation of *Graph-theoretic* steps prior to the application of multidimensional scaling (MDS). The emerged dimensionality reduction technique, named ***Isomap***, comprises simple algorithmic steps, that transform the original matrix **D** to **GD** which contains the geodesic interpoint distances [Tenenbaum et al.,2000]. In brief, *Graph theory* is engaged directly in the multidimensional space (the ambient space) by building the *nearest-neighbor graph* over the given point sample. Each point is treated as a node of this graph, while each straight-line segment connecting two of these points being closer than ε is treated as an edge of it (see panel d) in the figure below). Using this graph, the *geodesic* interpoint distances are computed as the shortest paths between each pair of points. The MDS is then applied, $\mathbf{Y} = MDS(\mathbf{GD}^{\varepsilon})$, to produce the image of the original point-cloud in a Reduced Space (panel e). Isomap can be thought of as a computationally efficient graph-flattening technique that can learn a broad class of nonlinear manifolds

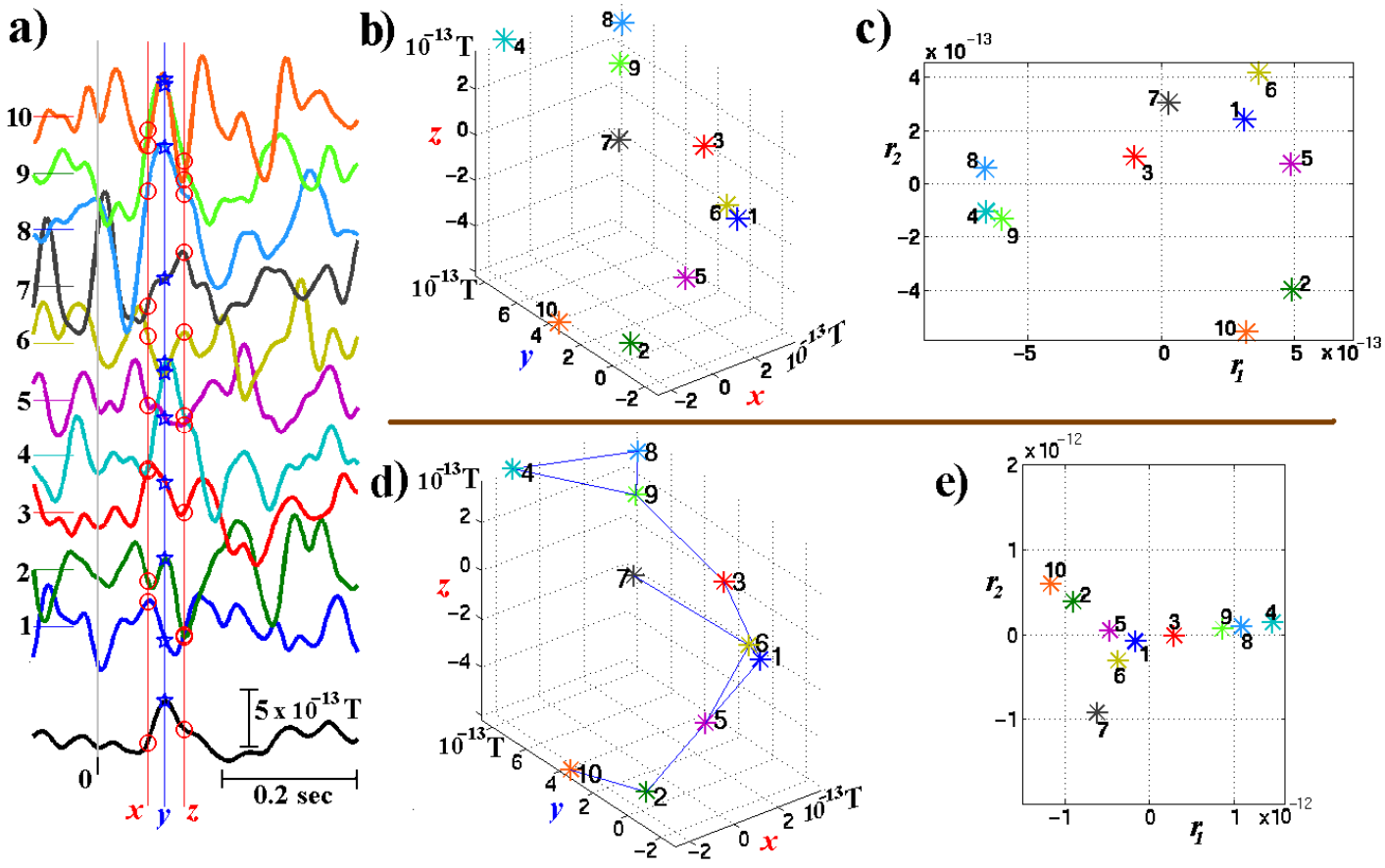


Fig.1 (a) Feature extraction step for a sample of temporal patterns (from 10 M100 auditory responses): the ensemble average waveform has been computed (black curve at the bottom) and a triplet of latencies around its peak has been selected; the signal-values at these latencies denoted by the vertical lines constitute the features for each pattern. (b) Feature Space construction: the ST-sample is represented as a point sample in a 3-dimensional space. (c) Reduced Feature Space computation: an image of the point sample in a space of 2 dimensions is derived via classical MDS. (d) Nearest neighbor graph formation: by connecting with straight-line segments these points which are closer than ε (selected as the average interpoint distance). (e) Unfolding the graph on a plane, using the ISOMAP algorithm. (adopted from [Laskaris, Clin. Neurophysiol.; 2002.])

The Isomap algorithm

Isomap is an extension of classical MDS that includes a transformation of the original distance matrix $D_{[NxN]}$ to the matrix GD=G(D) that contains the shortest path distance between all pair of points:

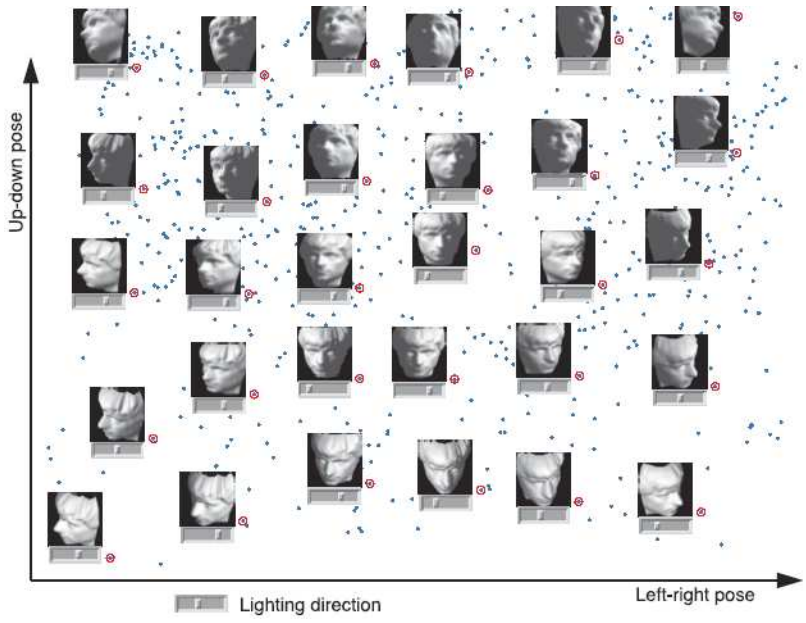
step_I. A weighted graph G is defined over all N points by connecting points Pi & Pj if (as measured by D(i,j)) they are closer than ε. The corresponding edge weights are initialized to GD(i,j)= D(i,j) if Pi , Pj are linked by an edge; GD(i,j)=∞ otherwise.

step_II. For each k=1,2,…N in turn, all entries GD(i,j) are replaced by *min*{ GD(i,j), GD(i,k)+GD(j,k)}. The fraction of points not connected to the main component of the resulting graph is detected and deleted from further analysis. As ε is reduced more points are deleted.

step_III. The images Y_i of points P_i in a space of reduced dimensions r are derived via the application of classical MDS,

$$Y_{[N'xr]} = MDS_r(GD)$$

While Isomap is a very competent procedure for learning nonlinear manifolds (see below, for a very interesting example with many potential applications in computer vision, like morphing), it is restricted by the computational demands of the geodesic-distance estimations. The handling of more than a few thousands multidimensional points (i.e. patterns) is becoming problematic. A remedy to this can be provided via the marriage of Isomap with unsupervised learning techniques. As a preprocessing-step, efficient techniques can be, first, applied in order to reform the ensemble of patterns as data-chunks, that will be then summarized via prototypes that will then be fed to the ISOMAP-routine. (The implicit assumption is that locally the Euclidean distance is a good approximation to the geodesic distance; something that holds only for relatively smooth manifolds)



The Neural-Gas network algorithm was found in practice a very convenient method for prototyping, i.e. preparing the data for the application of ISOMAP-algorithm, and its use is suggested in the case of very large pattern-databases.

Vector Quantization (VQ) based on Neural-Gas Network

VQ encodes the data manifold in the ambient (usually high-dimensional) space by utilizing only a finite set of reference vectors, the *code vectors*. It actually performs a parcellation of the ambient space known as Voronoi Tessellation, in which a *Voronoi-region* is defined around each code vector. This is a section in the original space comprised of all the points closer to a specific code vector than to any other. The vectorial observations falling within a Voronoi-region are represented by the corresponding code vector. The number of code vectors controls the resolution of the representation, i.e. the level of information abstraction. The following figure shows the Voronoi Tesselation when 15 code vectors are used in a simplified 2D-example from real data. The code vectors, denoted as red circles in panel-c), have been computed using a clustering algorithm so as to achieve the minimum coding error in the representation of the point swarm given in panel b).

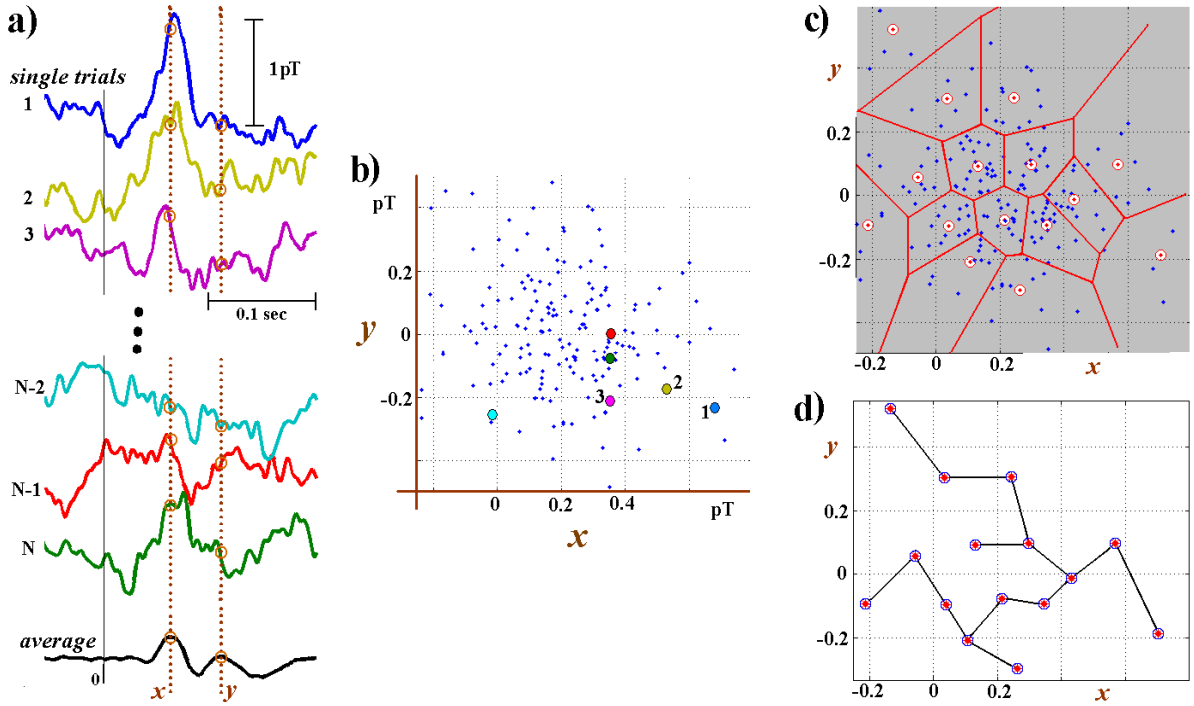


Fig. Graphical Representation of temporal patterns using Vector Quantization as abstraction step .a) Extracting features. **b)** Embedding the patterns in a 2D feature space. **c)** Applying Vector Quantization in the feature space. **d)** Constructing the Minimal Spanning Tree of the code vectors.

The codebook design is the most critical part in VQ. For this step the ***"neural-gas"*** algorithm is employed. This algorithm is an artificial neural network model, which converges efficiently to a small, user-defined number C<N of codebook vectors, using a stochastic gradient descent procedure with a ''soft-max'' adaptation rule that minimizes the average distortion error. This network is an extension of the Kohonen's self-organizing maps that shares some characteristics with the Fuzzy C-means algorithm. Its name stems from the physics of the underlying optimization scheme, since the reference vectors tend to cover all the space of the input data, while mutual repulsion forces are emerging.

dmatrix

```
function y = dmatrix(X)
% d=dmatrix(data =[N x p]),
%  data=[#vectors x dimensionality of the vector-space]

[N,p]=size(X);
A=X*X';
E=ones(N,N) ;
D=diag(diag(A))*E + E*diag(diag(A))-2*A ;
y=D;
```

****

```
X =
    1    2
    3    4
    5    6
    7    8
    9   10
```

```
>> A=X*X'
    5   11   17   23   29
   11   25   39   53   67
   17   39   61   83  105
   23   53   83  113  143
   29   67  105  143  181
```

```
>> E=ones(5,5)
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1
    1    1    1    1    1
```

```
>> diagA=diag(diag(A))
    5    0    0    0    0
    0   25    0    0    0
    0    0   61    0    0
    0    0    0  113    0
    0    0    0    0  181
```

```
>> diagA*E+E*diagA-2*A
D=
    0    8   32   72  128
    8    0    8   32   72
   32    8    0    8   32
   72   32    8    0    8
  128   72   32    8    0
```

dmatrix based on normalized vectors

```
X=
   1    2
   3    4
   5    6
   7    8
   9   10

>> Xn=normalize_vectors(X)   ;  %  this should be a small subroutine that divides each row of the Data matrix
                                  %with the norm of the row, i.e. the length of the vector. SEE BELOW

                                Xn =
                                        0.4472    0.8944
                                        0.6000    0.8000
                                        0.6402    0.7682
                                        0.6585    0.7526
                                        0.6690    0.7433

>> Dn=dmatrix(Xn)
                                Dn =
                                        0    0.0323    0.0532    0.0648    0.0720
                                   0.0323         0    0.0026    0.0057    0.0080
                                   0.0532    0.0026         0    0.0006    0.0014
                                   0.0648    0.0057    0.0006         0    0.0002
                                   0.0720    0.0080    0.0014    0.0002         0
```

```
%_____________________________________________________________________________
function   [normalized_X, RMS_values ] = normalize_vectors(X)

%   the array  normalized_X  contains  the  vectors Xi divided by the corresponding length  ||Xi||
%
%  the column-array RMS_values contains scalars that are related with the length of the vectors
%  and  correspond to RMS (root-mean-square) values in the case that the vectors are temporal-patterns/wavelets

[N,p]=size(X);

for i=1:N;
 normalized_X(i,:)= X(i,:)/norm(X(i,:));
 RMS_values(i)=norm(X(i,:)) *(1/sqrt(p)) ;
end
```

```
>>  X =
            1    2
            1    3
            2    4
            5     6

normalized_X =
                0.4472    0.8944
                0.3162    0.9487
                0.4472    0.8944
                0.6402    0.7682

RMS_values '= 1.5811    2.2361    3.1623    5.5227
```

Outlier Detection I

```
function  [Y, sel_list]=Reduced_ordering(X)
%
%   [Y,sel_list]=Reduced_ordering(X)
%
%            detecting outliers using the hypothesis that the vector-points
%           should form a spherical cluster.
%          while a small portion of them can appear as spurious points
%
%   Y is the artifact-free subset of  vectors
%   sel_list is the original indexing of the corresponding patterns
%

[N,p]=size(X);
d=dmatrix(X); sum_dist=sum(d); % correspond to each point its total distance
                                %  from the rest points (aggregate-distance)

[sd,list]=sort(sum_dist); % order the aggregate-distances

diffsd=sd(2:N)-sd(1:N-1); [mm,index]=max(diffsd);
                        %  detect where the aggregate-distance increase abruptly; the classical trick of  'plateau'-detection

list=list(1:index);  %  keep those points that correspond to small aggregate distances
sel_list=sort(list);  % bring them to the original "time-order"/indexing

Y=X(sel_list,:);
*************************************************************************
```

```
X =                >> [N,p]=size(X)      >>  d=dmatrix(X);
   1                                          0     1   100     0   11881     4   11881
   2                    N =7                  1     0    81     1   11664     1   11664
  11                    p =1                100    81     0   100    9801    64    9801
   1                                          0     1   100     0   11881     4   11881
 110                                      11881 11664  9801 11881       0 11449       0
   3                                          4     1    64     4   11449     0   11449
 110                                      11881 11664  9801 11881       0 11449       0


>> sum_dist=sum(d)
                          sum_dist =   23867     23412     19947     23867     56676     22971     56676

>> [sd,list]=sort(sum_dist)
                                  sd =  19947     22971     23412     23867     23867     56676     56676
                                  list =   3         6         2         1         4         5         7


>> diffsd=sd(2:N)-sd(1:N-1);   [ mm, index] =max(diffsd)
                                                  mm = 32809      index = 5

>> list=list(1:index)
                          list =  3     6     2     1     4

>> sel_list=sort(list)
                         sel_list = 1     2     3     4     6

>> Y=X(sel_list,:);   >>  Y' = 1  2  11   1   3
```

Outlier Detection II

```
function  [Y, sel_list] = radial_ordering( X , X_prot, k )
%
%    [Y, sel_list] = radial_ordering(X, X_prot, k)
%
%        detecting outliers using a reference-prototype
%
%    if the k is given, the k-nearest neighbors around the prototype are kept
%    otherwise (if k=[]) a simple automated-algorithm is utilized to estimate this k first
%
%    Y is the artifact-free subset of vectors
%    sel_list the original indexing of the corresponding patterns
%

[N,p]=size(X);
d=d_sample_to_vector(X,X_prot); % correspond to each point its distance to the reference point
                                %  see the m-file below
[sd,list]=sort(d); % order these distances


if isempty(k)
  diffsd=sd(2:N)-sd(1:N-1); [mm,index]=max(diffsd); % detect where the distance increase abruptly
                                                    % i.e. estimate the k (==index)
     list=list(1:index); % keep those points that correspond to small distances

else
     list=list(1:k);
end



sel_list=sort(list);  % bring them to the original "time-order"/indexing


Y=X(sel_list,:);



%_____________________________________________
function  y=d_sample_to_vector(X,Y)
%   distances =d_sample_to_vector (X, Y)
%                 X contains a set of row-vectors, Y is a row-vector;
%    distances contains the squarred Euclidean distances
%                               with respect to the reference-vector Y
%               X and Y should have the same number of columns

[N,p]=size(X);   y= diag([X-ones(N,1)*Y]*[X-ones(N,1)*Y]');
%_____________________________________________
```

```
X =
     1
     2
    11
     1
   110
     3
   110
```

```
>> X_prot = 2.5000
>> k=3
```

```
>> [N,p]=size(X)

     N =7
      p =1
```

```
>> d=d_sample_to_vector(X,X_prot)

d = 1.0e+04 *
      [ 0.0002  0.0000 0.0072  0.0002  1.1556  0.0000   1.1556
```

```
>> [sd,list]=sort(d)

sd = 1.0e+04 *
     [ 0.0000  0.0000  0.0002  0.0002  0.0072  1.1556  1.1556 ]

list =    2  6  1 4 3 5 7
```

```
>> if isempty(k), diffsd=sd(2:N)-sd(1:N-1); [mm,index]=max(diffsd);   list=list(1:index); else list=list(1:k); end

>> list=list(1:k)

list =
      2
      6
      1

>>  sel_list =

     1
     2
     6
```

```
>> Y=X(sel_list,:)


Y =
     1
     2
     3
```

Outlier Detection III

function [Y,sel_list]=NN_ordering(X,k)

```
%  [Y,sel_list]=NN_ordering(X,k)
%
%  detecting outliers using the distance to the nearest-neighbor
%   k controls the number of pattern/points to be kept
%
%  Y is the artifact-free subset of  vectors
%   sel_list is the original indexing of the corresponding  patterns


[N,p]=size(X);

d=dmatrix(X); [dd]=sort(d);   % for each point, its distances to the rest of  points are ordered


nnd=dd(2,:); % its nearest neighbor is easily identified
                    %  and the corresponding distance is attached to the point serving as a ‘non-typicality’-measure


[sd,list]=sort(nnd); % order these distances


if isempty(k)

  diffsd=sd(2:N)-sd(1:N-1);
  [mm,index]=max(diffsd);    % detect where the nn-distance increase abruptly

   list=list(1:index);  % keep those points that correspond to small nn-distances

else
    list=list(1:k);

end




sel_list=sort(list);  % bring them to the original "time-order"/indexing


Y=X(sel_list,:);
```

k=3 >> d=dmatrix(X); **>> [dd]=sort(d)**

X =

1	0	4	11881	4	1	0	0	0	0	0
3	4	0	11449	0	1	1	0	11449	0	1
110	11881	11449	0	11449	11664	4	1	11449	1	1
3	4	0	11449	0	1	4	4	11664	4	1
2	1	1	11664	1	0	11881	11449	11881	11449	11664

>> nnd=dd(2,:)

[1 0 11449 0 1]

>> [sd,list]=sort(nnd)

sd= [0 0 1 1 11449]

list = [2 4 1 5 3]

>> if isempty(k); diffsd=sd(2:N)-sd(1:N-1); [mm,index]=max(diffsd); list=list(1:index); else list=list(1:k); end

>> list=list(1:k) = 2 4 1

>> sel_list=sort(list)

sel_list = 1 2 4

>> Y=X(sel_list,:)

Y =

1

3

3

Subtractive_clustering

```
function [y1,snrave_max,y2] = subtractive_clustering(x)
%                                  x=[N x p], is the data_matrix
%                                   y1 is the extracted subset ,
%                                   y2 includes the remaining (sub)set(s)
%
%     see also the  SUBCLUST routine in fuzzy-logic toolbox of MATLAB
%

[N,p]=size(x);
d=dmatrix(x )  % building the matrix of interpoint-distances

r0=mean(mean(d)); % an estimate of the radius of influence: this is a crude approximation;
                        % it has to be estimated from noisy conditions

pot=exp(-d./(0.1*r0))  % transforming the distance-matrix

PD=sum(pot)   % a row-vector with each entry being propotional
                  % to the local–density around the corresponding point

[PDmax,imax]=max(PD); % identifying the point of highest local density

d_to_imax=d(imax,:);  % distances for x_imax from the rest of points

[ss,nearest_neighbors]=sort(d_to_imax) % radial-ordering of points with respect to the x_imax

% then we need to define how many of the ordered points should be selected
% since the algorithm was meant to be applied to pattern of time-waveforms, originally an Signal-to-Ratio
% estimator had been employed in the decision-making process.


ordered_x=x(nearest_neighbors ,:) % the input sample is reordered with respect the  x_imax
w=zeros(1,N)

for  i=2:N,
[sp,np]=SNR(ordered_x(1:i,:)); % an 'external' descriptor quantifying the 'grouping' is employed:
                  % specifically, an estimator of the SNR - the SNR_for_a_sample of time-waveformes-
                        % was applied or the nested sequence of subsets
w(i)=sp/np;
end

[snrave_max,jo]=max(w.*[1:N]) % find which subset provides the maximum SNR-measure
                                                  % i.e. identify the optimal rank jo
 if jo ~= N
 y1=ordered_x(1:jo,:); y2=ordered_x(jo+1:N,:);
 else
 y1=ordered_x; y2=[];
 end
 % actual outputs: indices of points belonging to dominant  mode
 % i.e. list of nearest_neighbors  and the threshold jo
```

```
x =
    1    2
    3    5
   22   33
   23   36
   21   45
    1    1
    2    1
```

```
>> [N,p]=size(x);

N =7 p =2
```

```
>>  d=dmatrix(x)

d=
    0   13 1402 1640 2249    1    2
   13    0 1145 1361 1924   20   17
 1402 1145    0   10  145 1465 1424
 1640 1361   10    0   85 1709 1666
 2249 1924  145   85    0 2336 2297
    1   20 1465 1709 2336    0    1
    2   17 1424 1666 2297    1    0
```

```
>> r0=mean(mean(d))

r0 = 853.5510
```

```
>> pot=exp(-d./(0.1*r0))

pot =
    1.0000    0.8587    0.0000    0.0000    0.0000    0.9884    0.9768
    0.8587    1.0000    0.0000    0.0000    0.0000    0.7911    0.8194
    0.0000    0.0000    1.0000    0.8894    0.1829    0.0000    0.0000
    0.0000    0.0000    0.8894    1.0000    0.3694    0.0000    0.0000
    0.0000    0.0000    0.1829    0.3694    1.0000    0.0000    0.0000
    0.9884    0.7911    0.0000    0.0000    0.0000    1.0000    0.9884
    0.9768    0.8194    0.0000    0.0000    0.0000    0.9884    1.0000
```

```
>> PD= sum(pot)

 PD =  3.8239    3.4693    2.0724    2.2589    1.5523    3.7678    3.7846
```

```
>> [PDmax,imax]=max(PD)
                PDmax =3.8239  imax =1
```

```
>> d_to_imax=d(imax,:)
            d_to_imax =  0    13     1402     1640     2249    1   2
```

```
>> [ss, nearest_neighbors]=sort(d_to_imax)
                    ss =  0    1    2   13  1402  1640  2249
    nearest_neighbors = 1    6    7    2    3     4     5
```

```
>> ordered_x=x(nearest_neighbors ,:)
ordered_x =
     1    2
     1    1
     2    1
     3    5
    22   33
    23   36
    21   45
```

```
W=zeros(1,N)
w = 0  0   0   0   0   0   0
```

```
>> for  i=2:N,

[sp,np]=SNR(ordered_x(1:i,:));
  % SNRsample for the nested sequence

w(i)=sp/np;
end
```

```
w'=
     0
  6.0000
  5.0000
  1.5556
  0.1796
  0.4500
  0.6957
```

```
>>[snrave_max , jo]= max(w.*[1:N])

   snrave_max = 15  ,   jo = 3
```

```
>> if jo ~= N
    y1=ordered_x(1:jo,:);
y2=ordered_x(jo+1:N,:);
   else
    y1=ordered_x; y2=[];
   end
```

```
>> y1=
     1     2
     1     1
     2     1
```

```
>> y2 =
     3     5
    22    33
    23    36
    21    45
```

```
>> snrave_max
                snrave_max = 15
```

```
>> [sp,np]=mine_snr(x);  overall_snrave= (sp/np)*7

    ans = 4.8697
```

% snrave_max > overall_snrave …….to check for cluster validity

% the procedure will be repeated for the remaining points subset y2

Multidimensional Scaling

function y = mscaling(D,r)

```
%
%  y=mscaling(dmatrix,r);
%  dmatrix is the array of pairwise distances
%  r is the dimensionality of the new space  (usually 1,2 or 3)
%  Torgeson's algorithm for Classic Multidimensional scaling



[n,n] = size(D);

A=-(1/2)*D;

Ac=mean(A); Al=mean(A'); Acl=mean(mean(A));

    for i=1:n;
        for j=1:n;
          B(i,j)=A(i,j)-Ac(i)-Al(j)+Acl;
          end
     end

 [v,d]=eig(b); % eigenvector is a column in v

evalues=diag(d);
[h,hh]=sort(evalues); % sorting from smaller to larger

 for i=1:r;
     c(:,i)= v(:,hh(n+1-i))* ((evalues(hh(n+1-i)))^(1/2));
 end

y=c;

************************************************
```

```
[n,n] = size(D)
                n = 5
```

```
>> A=-(1/2)*D
     0    -4   -16   -36   -64
    -4     0    -4   -16   -36
   -16    -4     0    -4   -16
   -36   -16    -4     0    -4
   -64   -36   -16    -4     0
```

```
>> Ac=mean(A);
>>  Al=mean(A');
>>Acl=mean(mean(A));
```

```
>> Ac =-24   -12    -8   -12   -24
>> Al=   -24   -12    -8   -12   -24
>> Acl =-16
```

```
>> for i=1:n;
       for j=1:n;
B(i,j)=A(i,j)-Ac(i)-Al(j)+Acl;
       end
  end
```

```
>> B
    32    16     0   -16   -32
    16     8     0    -8   -16
     0     0     0     0     0
   -16    -8     0     8    16
   -32   -16     0    16    32
```

```
>> [v,d]=eig(B)

v =
    0.45    0.0   -0.27    0.57   -0.63
   -0.89   -0.0   -0.13    0.29   -0.32
       0    1.0     0.0     0.0       0
   -0.00    0.0   -0.94   -0.09    0.32
       0   -0.0    0.14    0.76    0.63
```

```
d =

     0     0     0     0     0
     0     0     0     0     0
     0     0    0.     0     0
     0     0     0    0.     0
     0     0     0     0    80
```

```
>> evalues=diag(d)
     evalues = 0 0 0 0 80
```

```
>> [h,hh]=sort(evalues)
     h =  0 0 0 0 80
     hh = 3 1 2  4  5
```

```
% for i=1:r;   c(:,i)= v(:,hh(n+1-i))* ((evalues(hh(n+1-i)))^(1/2));  end

% *Defining   the dimensionality of the reduced space as   r==2.
```

```
>> for I=1:2;
c(:,I)= v(:,hh(n+1-I))* ((evalues(hh(n+1-I)))^(1/2));
end
```

y=c

```
y =
   -5.6569    0.0000
   -2.8284    0.0000
         0         0
    2.8284   -0.0000
    5.6569    0.0000
```

% If we compute the matrix of pairwise distances in the reduced space, i.e. dmatrix(y)

```
         0    8.0000   32.0000   72.0000  128.0000
    8.0000         0    8.0000   32.0000   72.0000
   32.0000    8.0000         0    8.0000   32.0000
   72.0000   32.0000    8.0000         0    8.0000
  128.0000   72.0000   32.0000    8.0000         0
```

% we can compare with the matrix in the original space dmatrix(X)
% to "measure " the distortion induced by the projection
% through the residual distance matrix : dmatrix(X)-dmatrix(Y)

An overall example

% Data matrix:

```
X =
     1     5     1
     3     5     2
     5     2    65
    21     4     2
    65     3     5
```

% Normalized_version:

```
>> Xn=normalize_vectors(X)

Xn =
    0.1925    0.9623    0.1925
    0.4867    0.8111    0.3244
    0.0767    0.0307    0.9966
    0.9781    0.1863    0.0931
    0.9960    0.0460    0.0766
```

% Computation of the two corresponding distance-matrices

```
>> D=dmatrix(X)

D =
     0     5  4121   402  4116
     5     0  3982   325  3857
  4121  3982     0  4229  7201
   402   325  4229     0  1946
  4116  3857  7201  1946     0
```

```
>> Dn=dmatrix(Xn)

Dn =
         0    0.1268    1.5279    1.2292    1.4987
    0.1268         0    1.2290    0.6854    0.9063
    1.5279    1.2290         0    1.6530    1.6918
    1.2292    0.6854    1.6530         0    0.0203
    1.4987    0.9063    1.6918    0.0203         0
```

% * producing the reduced space for both the normalized and the non-normalized version

```
>> Y=mscaling(D,2)
Y =
   -0.9386  -22.8157
   -0.4085  -20.6457
  -47.2180   21.6726
   11.2125   -6.8717
   37.3525   28.6605
```

```
>> Yn=mscaling(Dn,2)
Yn =
   -0.3999   -0.5344
   -0.1836   -0.3448
   -0.5711    0.6866
    0.5411    0.0419
    0.6134    0.1506
```

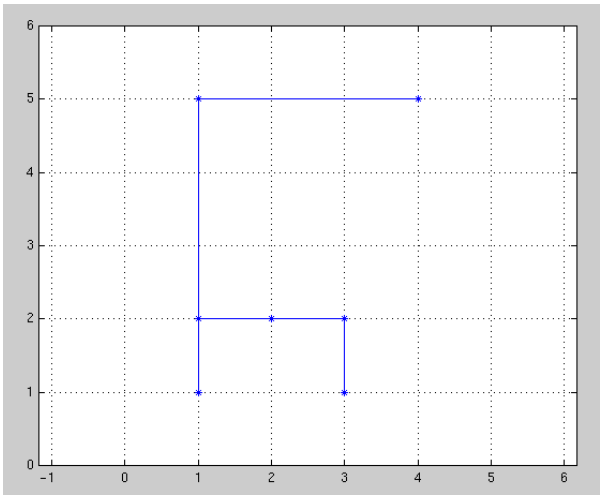
Minimal Spanning Tree

-A Graph theoretic Exploratory Data Analysis tool-

Function [links, weights] = minimal_spanning_tree(d)

```
%
%   [ links, weights ]=minimal_spanning_tree( distance_matrix/weights )
%   Prim's algorithm for constructing the Minimal-Spanning Tree (MST)
%   from a given symmetric distance/similarity matrix
%   links contains the edges, i.e. pairs of nodes, which constitute the MST
%   weights contains the corresponding weights
%
%    The basic philosophy of the algorithm is to identify the N-1 edges
%   -among the N(N-1) available- by sequentially selecting among the remainders  V-VT nodes
%    the node that is closer to the selected ones VT  and update the distances of the rest of nodes (V-VT) so as
%   the weight-array to tabulate for each un-selected node  its distance to the nearest node among the selected VT

[N,N]=size(d),
V=[1:N]  % ensemble of nodes
r=1; VT=r;  % start by first selecting the node 1; any other r could have been used

weights=zeros(1,N); links=zeros(1,N);  % initialization

V_VT=setdiff(V,VT);   % remove the selected node from the available set

weights(V_VT)=d(r,V_VT);  % the distances of the available nodes to the selected ones , i.e. the node r=1
links(V_VT)=r;         % the available nodes will compete for entering in the selected set VT ,
                        % based on the weight (i.e. distance)  of  the edge they share with node r=1

for i=1:N-2     % since N-1 edges/links need to be selected, the loop will be repeated N-2 times

 [edge_weight,u]=min(weights(V_VT)); % search in the set of unselected node for the one closer to
                                      %-anyone from-  the selected ones
 node=V_VT(u);              % refer to the original indexing of the nodes
 VT=union(VT,node);      % augment the set of selected nodes
 V_VT=setdiff(V,VT)      % remove the -just selected- node from the search space

            for j=1:N-1-i,    % for the available nodes (V-VT) the nearest distance to the selected nodes VT
                               % might have changes after moving the latest node
                              % so we need to update the corresponding entries in the weights - array
                               % actually, this update is necessary only if the latest node
                                %  is nearest neighbor with any of the remainders

           [weights(V_VT(j)),index]=min( [weights(V_VT(j)),d(node,V_VT(j))] );,

                 if index == 2,         %  if  d(node,V_VT(j))  <  weights(V_VT(j))
                 links(V_VT(j))=node;, % the nearest link for any of the available nodes is this that corresponds
                                        % to the last selected node
                 else,
                 end
           end
end

links=[2:N; links(2:N)]';  weights=weights(2:N)';  % do not care for the first selected node, r=1,

[ignore,list]=sort(weights); % present the links according to the length of the corresponding edges
links=links(list,:);   weights=weights(list);
```

```
>> X =                 >> d=sqrt(dmatrix(X)) =                                                    >>[links,weights]=minimal_spanning_tree(d)
     1     1
     1     2            0    1.0000    2.0000    1.4142    5.0000    4.0000    2.2361
     3     1       1.0000         0    2.2361    1.0000    4.2426    3.0000    2.0000     links =    2     1          weights = 1
     2     2       2.0000    2.2361         0    1.4142    4.1231    4.4721    1.0000                3     7                    1
     4     5       1.4142    1.0000    1.4142         0    3.6056    3.1623    1.0000                4     2                    1
     1     5       5.0000    4.2426    4.1231    3.6056         0    3.0000    3.1623                7     4                    1
     3     2       4.0000    3.0000    4.4721    3.1623    3.0000         0    3.6056                5     6                    3
                   2.2361    2.0000    1.0000    1.0000    3.1623    3.6056         0                6     2                    3
```



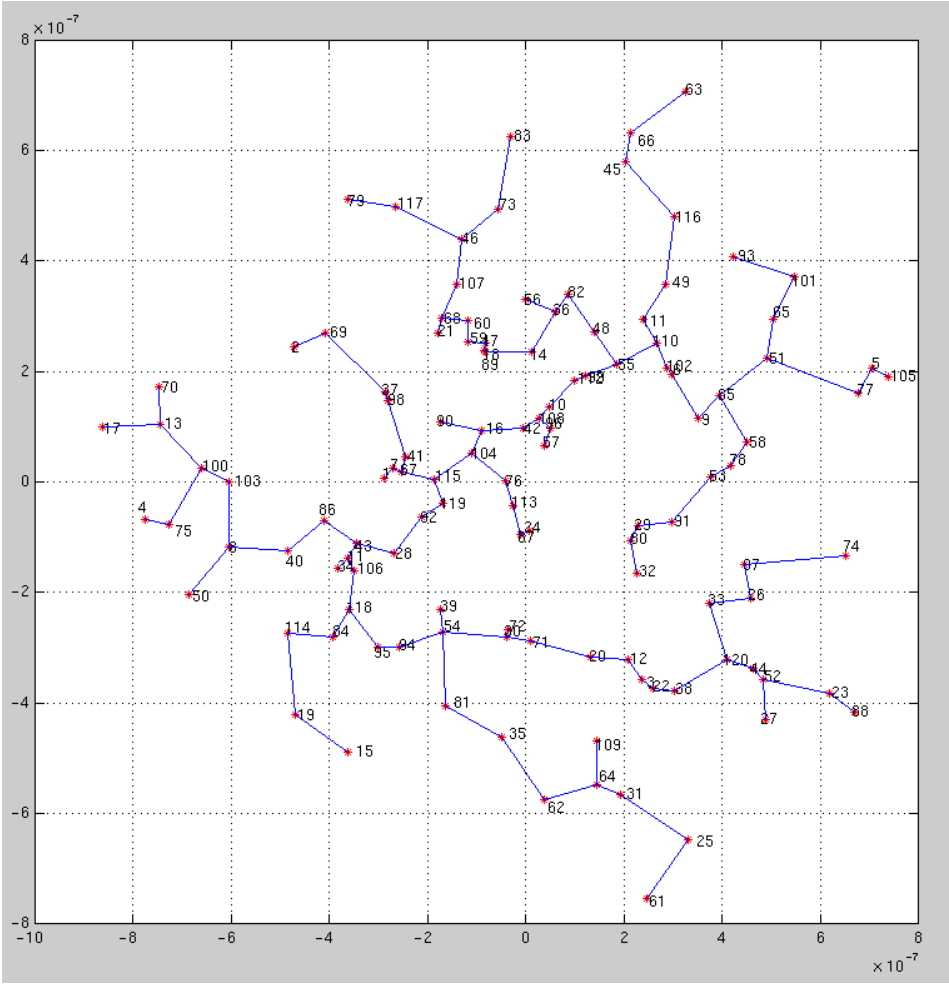
The steps of the above example follow_____

```
>> [N,N]=size(d), V=[1:N]
                              N =7     V = [1    2    3    4    5    6    7 ]

>> r=1;, VT=r
                          VT =1
>> weights=zeros(1,N); links=zeros(1,N);

>> V_VT=setdiff(V,VT)
                             V_VT = [ 2    3    4    5    6    7 ]

>> weights(V_VT)=d(r,V_VT)
                               weights = 0    1.0000    2.0000    1.4142    5.0000    4.0000    2.2361

>> links(V_VT)= r
                        links = [0    1    1    1    1    1    1]

>> MAIN LOOP , for  i=1

  >> [edge_weight,u]=min(weights(V_VT))
                                          edge_weight = 1      u = 1
>>  node=V_VT(u)
                         node = 2
>> VT=union(VT,node)
                             VT = 1    2
>> V_VT=setdiff(V,VT)
                               V_VT =  3    4    5    6    7

>> Second Loop, for j=1

          >> [weights(V_VT(j)),index]=min( [weights(V_VT(j)),d(node,V_VT(j))] )
         weights =  [0    1.0000    2.0000    1.4142    5.0000    4.0000    2.2361]    ,    index = 1

          >> if index == 2,  links(V_VT(j))=node;, else,end

             for  j=2
         >> [weights(V_VT(j)),index]=min( [weights(V_VT(j)),d(node,V_VT(j))] )

        weights =  [0    1.0000    2.0000    1.0000    5.0000    4.0000    2.2361] ,     index = 2
 >> if index == 2,  links(V_VT(j))=node;, else,end
 >>  links =  0    1    1    2    1    1    1
```

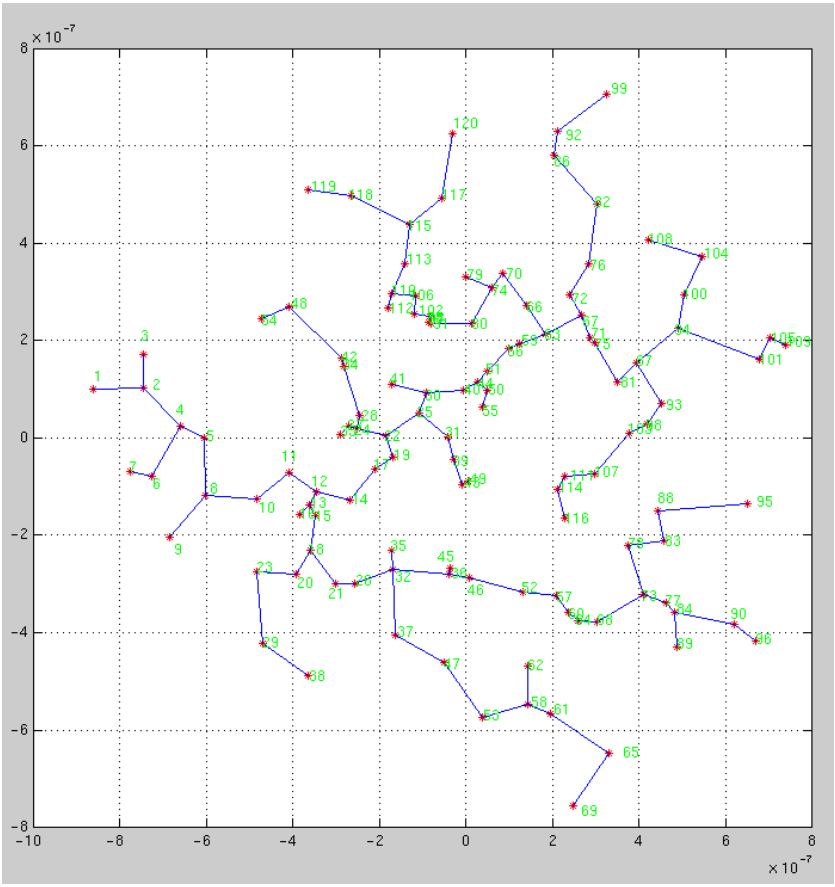
The figure aside, **shows the MST of 120 Single-Trial Signals from Auditory M100 responses after the "planing" procedure has been applied (see [Laskaris et al.,Clin.Neurophysiol. 20001]). The superposed indices indicate the time-order of the corresponding ST-signals.**

After selecting the 17-th ST as the root of the tree, MST-ordering was performed.

The new labels -ranks from this vectorial procedure have been superposed in the next figure



The Ordered Point-Sample :



Isomap

A coupling of Graph theory
with
Multidimensional Scaling

function [x,new_list,outliers,dg_new]=isomap_e(X,r,e)

```
%   function  [x, new_list, outliers, dg_new] = isomap_e_mscaling(vectors,r,e)
%  x:[N x r] projections of  vectors:[N x p]
%   r=reduced dimensionality
%   e=distance defining a typical neighborhood-size

[N,p]=size(X);
d=sqrt(dmatrix(X));

%%%%%%%% if no typical radius e is given %%%%%%%%%%%%%
if nargin==2
sd=sort(d); e=1.1*max(sd(2,:)); % alternatively e=mean(mean(d));
else
e=e;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%% %%computation of graphical distances  %%%%%%%%%%%%
%%%%%%%%%%%% FLOYD'S algorithm %%%%%%%%%%%%%%%%%

dg=Inf*ones(size(d));
c=d<e;
dg(c)=d(c);

for k=1:N;
    for i=1:N;
       for j=1:N;

       dg(i,j)=min(dg(i,j), dg(i,k)+dg(k,j));
       end
     end
 end
  %-----------------------------------------------------

%%%%%   detecting outliers based on the graph distances  %%%%%%%%
sdg=sort(dg);
for i=1:N,s(i)=length(find(sdg(:,i)==inf));,end
list= find(s>min(s));
%-------------------------------------------------------------------------

%%    OUTPUT DEFINITION   %%%%%%%%%%%%%%%%%%5

 new_list=setdiff([1:N],list);  outliers=list;
 dg_new=dg(new_list,new_list);
 x=mscaling(dg_new.^2,r);
```

% Inputs:

r=1, e=2

X =

1
2
3
4
11
5
6

>> [N,p]=size(X)

N =7, p = 1

% Construct the MATRIX of
% PAIRWISE EUCLIDEAN DISTANCES

>> d=sqrt(dmatrix(X))

0	1	2	3	10	4	5
1	0	1	2	9	3	4
2	1	0	1	8	2	3
3	2	1	0	7	1	2
10	9	8	7	0	6	5
4	3	2	1	6	0	1
5	4	3	2	5	1	0

% CREATE THE MATRIX of
% PAIRWISE **GRAPHICAL** DISTANCES

>> dg=Inf*ones(size(d))

Inf	Inf	Inf	Inf	Inf	Inf	Inf
Inf	Inf	Inf	Inf	Inf	Inf	Inf
Inf	Inf	Inf	Inf	Inf	Inf	Inf
Inf	Inf	Inf	Inf	Inf	Inf	Inf
Inf	Inf	Inf	Inf	Inf	Inf	Inf
Inf	Inf	Inf	Inf	Inf	Inf	Inf
Inf	Inf	Inf	Inf	Inf	Inf	Inf

% FIND IN THE MATRIX of EUCLIDEAN DISTANCES those smaller than e
% and create the corresponding mask

>> c=d<e

1	1	0	0	0	0	0
1	1	1	0	0	0	0
0	1	1	1	0	0	0
0	0	1	1	0	1	0
0	0	0	0	1	0	0
0	0	0	1	0	1	1
0	0	0	0	0	1	1

% Use the mask to initialize the MATRIX of GRAPHICAL DISTANCES

>> dg(c)=d(c)

0	1	Inf	Inf	Inf	Inf	Inf
1	0	1	Inf	Inf	Inf	Inf
Inf	1	0	1	Inf	Inf	Inf
Inf	Inf	1	0	Inf	1	Inf
Inf	Inf	Inf	Inf	0	Inf	Inf
Inf	Inf	Inf	1	Inf	0	1
Inf	Inf	Inf	Inf	Inf	1	0

% RUN FLOYD'S algorithm to define the Graphical Distances

>> for k=1:N;for i=1:N;for j=1:N; dg(i,j)=min(dg(i,j), dg(i,k)+dg(k,j));end, end, end

% The computed Graphical Distances (after the previous step) are

>> dg

0	1	2	3	Inf	4	5
1	0	1	2	Inf	3	4
2	1	0	1	Inf	2	3
3	2	1	0	Inf	1	2
Inf	Inf	Inf	Inf	0	Inf	Inf
4	3	2	1	Inf	0	1
5	4	3	2	Inf	1	0

% In order to identify possible Outliers, we order the Graphical Distances across columns

>> sdg=sort(dg)

0	0	0	0	0	0	0
1	1	1	1	Inf	1	1
2	1	1	1	Inf	1	2
3	2	2	2	Inf	2	3
4	3	2	2	Inf	3	4
5	4	3	3	Inf	4	5
Inf	Inf	Inf	Inf	Inf	Inf	Inf

% We then "count" the Inf entries

>> for i=1:N, s(i)=length(find(sdg(:,i)==inf)); ,end

% For each vector we assign the number of vectors "sensed" as outliers by the certain vector

>> s

1 1 1 1 6 1 1

% We finally locate the Graphical Outliers

>> list= find(s>min(s))

list = 5

% We keep the rest of the vectors

>> new_list=setdiff([1:N],list)

1 2 3 4 6 7

% We then extract the corresponding Matrix of Graphical Distances

>> dg_new=dg(new_list, new_list)

0	1	2	3	4	5
1	0	1	2	3	4
2	1	0	1	2	3
3	2	1	0	1	2
4	3	2	1	0	1
5	4	3	2	1	0

% Finally, we apply multidimensional scaling……

>> x=mscaling(dg_new.^2,r)

2.5000
1.5000
0.5000
-0.5000
-1.5000
-2.5000

%The output consist of the coordinates x in the reduced space the list of vectors included
% in the graph new_list and the detected outliers

>> outliers=list

5

Vector Quantization Based on the *Neural Gas Network*

Function [prototypes, class_indicators, Average_Error, Convergence_Index]

= Vector_Quantization(X, k, iteration_factor)

```
%  Neural-Gas Vector Quantization Algorithm
%  X is the input set of patterns, k the size of the code-book
%                                   (i.e. the number of prototypes / centroids/ codevectors)
%  the iteration_factor controls the number of iterations : = = (iteration_factor) x (size of the input sample)
%
%  prototypes: tabulates the k code-vectors
%  class_indicators: tabulates the labels that assign each vector Xi to the nearest prototype
%  Average_Error is an index of performance: it is the average Distortion induced by the adopted coding scheme
%  Convergence_Index indicates the improvement, with respect to the initial/random selection of prototypes ,
%  achieved with the iterative-execution of the basic adaptation-step

[N,p]=size(X);
_____________________________________________________________________
%_____________________ initialization _____________________________
%  select randomly k prototypes.
% case-(i) If  k>N/2 create these prototypes by averaging  randomly selected subsets of X
% case-(ii) If  k<N/2 select randomly k vectors from  the sample X

rindex=permut(N);    %  this simple function is listed below

fl=floor(N/k);    % floor-function returns the smaller integer closer to the argument

    if fl>=2    % case (i)
      for i=1:k
      rr=rindex((i-1)*fl+1:(i)*fl);      % split the random permutation e.g. [1 4 11 ……] into k groups of indexes
      prot(i,:)=mean(X(rr,:));            % use the corresponding indexes to split the curves into k groups
                                           % and average within each group to produce the prototypes
      end

    else   % case (ii)
      prot=X(rindex(1:k),:);          % select randomly k curves from X to be used as prototypes
    end


%_________ initial coding error  _________

for i=1:N
d=d_sample_to_vector(prot,X(i,:));  % since V.Q. assigns each vector  Xi to the closest of the prototypes
[error(i)]=min(d);  % a measure of the initial coding-error  (i.e. when codevectors are randomly selected)
end                  % can be computed.
initial_Average_Error=mean(error);
%_____________________________________
```

```
%_____________________Basic adaptation Rule _____________________

tmax=iteration_factor*N;    %  number of iterations, i.e. how many times the adaptation-rule is going to be applied

rr=[];                      % create an 1-d array containing a random-sequence of numbers pointing to the vectors Xi
                            %  of the  input data-matrix X ; the length of this array is equal to the total number of iterations
for i=1:iteration_factor,
rr=[rr;permut(N)'];
end

li=0.3*k; lf=0.01; ei=0.5; ef=0.005; % i stand for initial and f for final value of the two parameters l(t) and e(t)
                                     % that are adapting at each iteration: e(t) modulates the strength of the
                                     % correction , while l(t) controls how many prototypes are modified
lt=li; et=ei;            % initialize these two parameters


for i=1:tmax;                % Back-bone of the V.Q. procedure, i.e. the training of  the prototypes

     u=X(rr(i),:);                          % pick at random a vector Xi from the input set
    du=d_sample_to_vector( prot,u);         % compute its distance from the prototypes

    [sdu,ordering_list]=sort(du);       % order the prototypes according to their closeness to the certain vector Xi
    [ignore,order]=sort(ordering_list);    % assign to each prototype its rank
    order=order-1;                      % so as the nearest prototype has rank '0', the second nearest rank '1', etc….

    hl=exp(-order/lt); % based on their order a parameter of influence is defined for each prototype

     % a (vectorial) correction is estimated for each prototype, according to the parameter of influence and the
     %  "error" of each prototype, i.e. its vectorial-difference from the certain Xi
     %  this correction is finally modulated by the current value of e(t)

     % the corrections to the set of prototypes are computed with the following loop
     % or, more efficiently,  with the subsequent matrix-operations
     % for i=1:k; dprot(i,:)=et * hl(i) * (u-prot(i,:)); end

     dprot= et*repmat(hl,1,p).* (repmat(u,k,1)-prot);
     prot=prot+dprot;            % the vectorial-corrections are applied to the current-prototypes

    lt=li*(lf/li)^(i/tmax); % the parameter l(t) is adapted so as to converge at its final-value lf  at the end
   et=ei*(ef/ei)^(i/tmax);  % >>    >>      e(t)    >>          >>          >>

end

prototypes=prot;          % the final estimation of the code-book

% using a simple nearest-classification rule each input Xi is assigned to one of the estimated code-vectors
%  in this way a grouping of the vectors Xi is also carried out.

for i=1:N
d=d_sample_to_vector(prototypes,X(i,:));  % compute the (squared) distance of Xi to each prototype
                                          % -this is also the so called quantization-error-
[error(i),indicator(i)]=min(d);  % assign the Xi to the prototype that results to the smallest quantization-error
end
class_indicators=indicator;  %  for each Xi, keep as a label the index of its nearest-prototype
```

```
Average_Error=mean(error); % estimate the average quantization-error after the nearest-prototype assignment
Convergence_Index = abs(Average_Error-initial_Average_Error)/initial_Average_Error;
                        % an index of  convergence: can serve as a diagnostic that the number of iteration should
                         % be increased (if it is not smaller than 1 ).


%_____________________________________
function   y=permut(N)
% in Matlab there is the corresponding  "built-in" function randperm
%  random permutation of the numbers [1,2,….,N ]

r=randn(1,N); % generate N random numbers and
[ignore, ordering_list ]=sort(r); % order them

y=ordering_list;  % their order is a random permutation of the numbers [1:N]
```

```
>> X=[1 2; 1 3; 3 4; 4 5; 5 1; 2 3; 5 6]

X =
    1    2
    1    3
    3    4
    4    5
    5    1
    2    3
    5    6
```

```
>> k=3, iteration_factor=2

>> [N,p]=size(X) ; N = 7  p = 2

>> rindex=permut(N)

rindex =     4     5     3     1     6     2     7
```

```
>> fl=floor(N/k) ;   fl = 2

>> if fl>=2
       for i=1:k
       rr=rindex((i-1)*fl+1:(i)*fl);
       prot(i,:)=mean(X(rr,:));
       end

   else
     prot=X(rindex(1:k),:);
   end
```

```
>> prot =
    4.5000    3.0000
    2.0000    3.0000
    1.5000    3.0000
```

```
 >> for i=1:N
d=d_sample_to_vector(prot,X(i,:));
[error(i)]=min(d);
end
>>  error = 1.2500    0.2500    2.0000    4.2500    4.2500         0    9.2500
```

```
>> initial_Average_Error= mean(error)
     initial_Average_Error = 3.0357

>> tmax= iteration_factor*N;    tmax
=14

>> rr=[]; for i=1:iteration_factor,
rr=[rr;permut(N)']; end
```

```
>> rr = 6
     1
     3
     5
     7
     2
     4
     2
     7
     6
     1
     3
     5
     4
```

```
>> li=0.3*k;  li=0.9000  >> lf=0.01; ei=0.5; ef=0.005;  >> lt=li; et=ei;
```

```
%___Basic Loop_________
>> for  i= =1

>> u=X(rr(i),:);     u = 2     3

>> du=d_sample_to_vector( prot,u)

du = 6.2500    0    0.2500

>> [sdu,ordering_list]=sort(du)

sdu = 0  0.2500   6.2500
ordering_list =  2 3  1

>> [ignore,order]=sort(ordering_list)
          order =  3 1  2
>> order=order-1
                order = 2 0  1
```

```
>> hl=exp(-order/lt)
hl = 0.1084  1.0000   0.3292

>> dprot=  et*repmat(hl,1,p).*
(repmat(u,k,1)-prot)
dprot =
   -0.1355         0
         0         0
    0.0823         0

>> prot=prot+dprot
prot =
    4.3645    3.0000
    2.0000    3.0000
    1.5823    3.0000
```

```
>> lt=li*(lf/li)^(i/tmax)
;
       lt = 0.6526

>>
et=ei*(ef/ei)^(i/tmax) ;
       et = 0.3598
```

```
%___continuation of the Basic Loop _________
>> for i=2:tmax;
   u=X(rr(i),:);
   du=d_sample_to_vector( prot,u);
   [sdu,ordering_list]=sort(du);
   [ignore,order]=sort(ordering_list); order=order-1;
    hl=exp(-order/lt);


   % for i=1:k; dprot(i,:)=et * hl(i) * (u-prot(i,:)); end

   dprot= et*repmat(hl,1,p).* (repmat(u,k,1)-prot);
   prot=prot+dprot;

   lt=li*(lf/li)^(i/tmax); et=ei*(ef/ei)^(i/tmax);
end
%_____________End_________________
```

```
>> prototypes=prot;
prototypes =
   4.4717   3.3122
   2.2402   3.1921
   1.3210   2.6820

>> for i=1:N
d=d_sample_to_vector(prototypes,X(i,:));
[error(i),indicator(i)]=min(d);
end

>> error = 0.5681   0.2042   1.2301   3.0711   5.6255   0.0946   7.5032

>> indicator =  3    3    2    1    1    2    1

>> class_indicators=indicator

class_indicators = 3    3    2    1    1    2    1

>> Average_Error=mean(error)
Average_Error = 2.6138

>> Convergence_Index = abs(Average_Error-initial_Average_Error)/initial_Average_Error

Convergence_Index =  0.1390


% A simple 2-D example
%Points in the 4 corners: >>  p1 =[1  0] ;   p2= [0  0] ;   p3= [0 1];   p4 = [1  1]
%Construct a sample X, by taking 11 from the 1st, 5 from the 2nd , 7 from the 3rd  and  5 from the last
% points=[repmat(p1,11,1);repmat(p2,5,1);repmat(p3,7,1);repmat(p4,5,1)];
% Add noise >> X=points+0.2*randn(size(points));
>>[prototypes,]=Vector_Quantization(X,4,30);
   0.9987   0.7567
  -0.0552  -0.0187
  -0.0798   0.9969
   0.9784  -0.0329
```

REFERENCES

D. F. Morrison, *Multivariate Statistical Methods.* McGraw-Hill Inc., 1990.

A. Jain and R. Dubes, *Algorithms for Clustering Data*, New Jersey: Prentice Hall, 1988.

J. Astola, P. Haavisto P, Y. Neuvo, *Vector median filters.* Proc. of the IEEE, 1990; 78:678.

I. Gath, A. B. Geva, *Unsupervised Optimal Fuzzy Clustering*. IEEE PAMI. 1989; 11-7:773-781.

R.C. Hardie, G R Arce. *Ranking in Rp and its use in multivariate image estimation.* IEEE Trans on Circuits and Systems for Video Technology, 1991; 1:197-209.

E. Parzen**,** *On Estimation of a Probability Density Function and Mode.* Annals of the Institute of Statistical Mathematics, 1962; 33:1065-1076

P. J. Rousseeuw, B. C. Zomeren, *Unmasking Multivariate Outliers and Leverage Points.* Journal of the American Statistical Association, 1990; 85-411:633-639

J.T. Tou, R.C. Gonzalez, *Pattern Recognition Principles.* Addison-Wesley, 1974.

R. Yager, D. Filev, *Generation of Fuzzy Rules by Mountain Clustering.* Journal of Intelligent and Fuzzy Systems, 1994; 2:209-219.

J. Friedman and L. Rafsky, ‘’Graphics for the multivariate two-sample problem’’. *J. Am. Statist. Assoc.,* vol. 76, pp. 277-287, 1981.

T. Martinez, S. Berkovich and K. Schulten, “Neural-Gas Network for Vector Quantization and its application to Time-Series prediction”, *IEEE Trans Neural Networks,* vol.4, pp. 558-569, 1993.

N. Laskaris, S. Fotopoulos, P. Papathanasopoulos and A. Bezerianos, “Robust moving averages, with Hopfield Neural Network implementation, for the monitoring of evoked potential signals”, *Electroencephalogr. Clin. Neurophysiol.*, vol.104, pp. 151-156, 1997.

N. Laskaris and A.A. Ioannides, “Exploratory data analysis of evoked response single trials based on minimal spanning tree”, *Clin. Neurophysiol.,* vol. 112, pp. 698-712, 2001.

N. Laskaris and A.A. Ioannides, “Semantic geodesic maps: a unifying geometrical approach for studying the structure and dynamics of single trial evoked responses”, *Clin. Neurophysiol.* vol. 113, pp. 1209-1226, 2002.

N. Laskaris N., S. Fotopoulos, A.A. Ioannides **‘‘**Mining Information from event-related recording’’ IEEE Signal Processing Magazine (special issue on *Signal Processing for Mining Information*) 2004; 21(3): 66-77.