file: rep03c.tex, modified: March 06, 2004, 28.08.2004

5 Applied Methods

5.1 Tests of normality based on multivariate kurtosis

Kurtosis evaluated using a population p.d.f.

Multivariate kurtosis, $\beta_{2,d}$, was defined by Mardia [6, 7], as (let us mention that the index 1, e.g., $\beta_{1,d}$ is reserved for multivariate skewness):

$$\beta_{2,d} = \mathsf{E}\{(\mathbf{x}-\boldsymbol{\mu})^T\boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})\}^2, \tag{1}$$

where $\boldsymbol{\mu}, \boldsymbol{\Sigma}$ denote the expected value (mean) and covariance matrix of the r.v. $\mathbf{x}$ appropriately. The symbol d denotes the dimensionality of the vector (= to the number of its components).

One may see from eq. [1] that this is the expected value of the Mahalanobis distance evaluated for a random vector $\mathbf{x}$ with a general p.d.f. $f(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$.
Mardia has shown that for normal distributions (i.e. for p.d.f. $f(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \mathcal{N}_d(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$)

$$\beta_{2,d} = d(d+2).$$

In this context the population *excess kurtosis* is defined as

$$\gamma_{2,d} = \beta_{2,d} - d(d+2). \tag{2}$$

Sample kurtosis

Suppose now, we have a sample $\mathbf{x}_i = (x_{i1}, x_{i2}, \cdots, x_{id})^T$ $(i = 1, \ldots, N)$, each vector $\mathbf{x}_i$ coming from p.d.f. $f(\mathbf{x}, \boldsymbol{\mu}, \boldsymbol{\Sigma})$, with unknown parameters $\boldsymbol{\mu}, \boldsymbol{\Sigma}$.

The sample counterpart of $\beta_{2,d}$ determined by eq. [1] is the sample kurtosis $b_{2,d}$ defined as

$$b_{2,d} = \frac{1}{N}\sum_{i=1}^{N}\{(\mathbf{x}_i-\bar{\mathbf{x}})^T\mathbf{S}^{-1}(\mathbf{x}_i-\bar{\mathbf{x}})\}^2, \tag{3}$$

where $\bar{\mathbf{x}} = (\bar{x}_{.1}, \ldots, \bar{x}_{.d})^T$ denotes the sample mean, and $\mathbf{S}$ the sample covariance matrix.

The sample counterpart of $\gamma_{2,d}$ defined by eq. [2], is the *sample excess kurtosis* $G_{2,d}$, is defined as

$$G_{2,d} = b_{2,d} - d(d+2). \tag{4}$$

Tests of multivariate kurtosis

Mardia [6, 7] has derived two asymptotic statistics for testing, whether the sample kurtosis differs significantly from that characterizing multivariate normal distributions.

The test statistics are given as statistics $u^\star$ and u' defined by formulae [5] and [6] below:

$$u^* = \frac{b_{2,d} - d(d+2)}{[8d(d+2)/N]^{1/2}}. \tag{5}$$

$$u' = \frac{[b_{2,d}(N+1) - d(d+1)(N-1)][(N+3)(N+5)]^{1/2}}{[8d(d+2)(N-3)(N-d-1)(N-d+1)]^{1/2}}. \tag{6}$$

Under multivariate normality, both statistics are distributed $\mathcal{N}(0,1)$.

Results for the analyzed data sets

The results are given in the table below :

Data set	d	$b_{2,d}$	$E\{b_{2,d}\|\mathcal{N}\}$	$u^\star$	u'	Significance
Erosion:	3	21.49	15	34.67	50.87	***
Phonetic:	20	471.16	440	23.26	39.04	***
Glass:	7	205.67	63	92.96	104.28	***
Ionosphere:	32	2772.65	1088	338.30	387.73	***

Three stars denote that $P < 0.001$.

Both test statistics indicate clearly that none of the analyzed data sets can be considered as having kurtosis plausible to be observed for normally distributed data. Indeed, all four data sets exhibit a highly elevated kurtosis.

5.2 Looking for outlying points using Mahalanobis distances

The outlying points were sought on the basis of robust Mahalanobis distances evaluated using the `fastmcd` algorithm developed by Rousseeuw and van Driessen [9]. For calculations we have applied the Matlab function `fastmcd` offered by the cited authors; the function yields (a.o.) an index-plot of robust Mahalanobis distances. The plots are reproduced in our report.

Let $\mathbf{x}_i \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, $i = 1, \ldots, N$. It may be easily shown that squared Mahalanobis distances, evaluated for subsequent data vectors $\mathbf{x}_i$ using known location and scale parameters $(\boldsymbol{\mu}, \boldsymbol{\Sigma})$, are distributed as chi^2_d.

In case of unknown values of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ we take their estimates, i.e. $\hat{\boldsymbol{\mu}} = \bar{\mathbf{x}}$ and $\hat{\boldsymbol{\Sigma}} = \mathbf{S}$, with $\mathbf{S}$ denoting the sample covariance matrix.

Let $chi_{.975}$ denote the (upper) 0.975 quantile of the chi^2_d distribution. Values of Mahalanobis distances greater then the boundary $bd = \sqrt{chi_{.975}}$ may be suspected as outliers.

The results for the four analyzed data sets are given in the table below('Observed Rob' means that the underlying frequencies were estimated using the robust covariance matrix proposed by Rousseeuw and van Driessen [9]):

Data set	d	N	$chi_{.975}$	$\sqrt{chi_{.975}}$	Expected $\|\mathcal{N}$	Observed	Observed Rob
Erosion:	3	3422	9.3484	3.0575	85.550	46	871
Phonetic:	20	1962	34.1696	5.8455	49.050	143	346
Glass:	7	214	6.0128	4.0016	5.350	21	65
Ionosphere:	32	351	49.4804	7.0342	8.775	79	165

In previous chapters we have shown for each analyzed data set the index plot exhibiting the robust Mahalanobis distances. In the same plot also the boundary $bd = \sqrt{chi_{.975}}$ is shown. One may notice that – for each of the data sets – there are much more points surpassing the boundary, as if the data were normally distributed.

In fact, except the erosion data, the three other data sets contain no outliers. We decided to inspect more closely 20 data points exhibiting the largest Mahalanobis distances. We imagine that these are the most peripheral points when considering their position in relation to the entire data cloud. These points got the name: *distal points*.

5.3 Visualizing data using Self-Organizing Maps

The concept of a self-organizing map

Self-organizing maps provide a kind of clustering of the data. The method performs favorably with other methods of clustering. For a comparison with other clustering methods – in the context of DNA microarrays – see Dougherty [2].

We have used maps organized in sheets. This means that every map was located in a plane. The established map is based on a regular grid of neurons, see the left exhibit in the figure below. Otherwise, we may look at the map as composed of regular, square cells, which surround the neurons, see the right exhibit in the figure below.

Let M denote the total number of neurons, and m_1, m_2 — the side length of the map. We have: $M = m_1 \times m_2$.

Below we show a map of size $m_1 \times m_2 = 2 \times 3$; it is based on 6 neurons and has 6 map units.

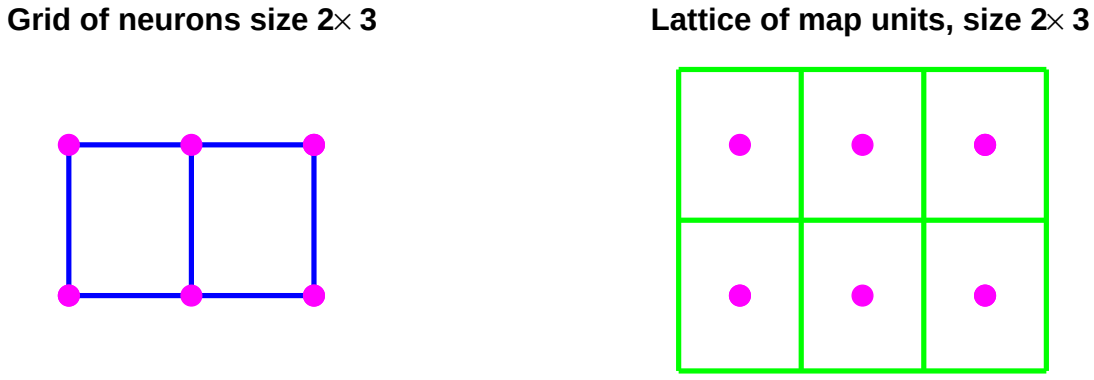

Figure 19: *Kohonen's self-organizing map (SOM) of size* $m_1 \times m_2 = 2 \times 3$. Left*: Map represented by a grid of neurons.* Right*: Map represented by a lattice of cells containing the neurons in their center.* {*grid1.eps*}

The coordinates of the neurons (in the map) are given by *reference vectors* $\mathbf{r}_i$, $i = 1, \ldots, M$, $\mathbf{r}_i \in R^2$.

A key concept in self-organizing maps is the *neighborhood function*. It permits to evaluate the neighborhood of two neurons.

We have used the *Gaussian* neighborhood function defined by the formula below:

$$g_{i,c} = \exp\{-\frac{\|\mathbf{r}_i - \mathbf{r}_c\|^2}{2\sigma^2}\}. \tag{7}$$

The neighborhood function determines a kind of membership function expressing in the scale [0, 1], for the vector $\mathbf{r}_i$, the likeliness of the event 'being a neighbor of $\mathbf{r}_c$'. The function has the form of a Gaussian kernel centered at $\mathbf{r}_c$. The width of the kernel is determined by the parameter σ^2. The value of the neighborhood function [7] decreases with increasing distance of $\mathbf{r}_i$ from $\mathbf{r}_c$.

In practice, the neighborhood function $h_{i,c}$ is time-dependent (time measures the progress of training; below we write more about 'training' the SOM). In such case, the width parameter σ^2 of the function [7] is time-dependent: $\sigma^2 = \sigma^2(t)$; it is a decreasing function of

time. Thus we may write

$$g_{i,c} = g_{i,c}(t) = \exp\{-\frac{\|\mathbf{r}_i - \mathbf{r}_c\|^2}{2\sigma^2(t)}\}. \tag{8}$$

Now let us consider the **feature space** R^d. We introduce there M prototype vectors, denoted $\mathbf{m}_1, \ldots, \mathbf{m}_M$. In the `SomTB` toolbox [14] this is achieved by constructing the plane based on the first two principal components of the data and putting that plane in the data space. Then, the grid of neurons from the map is reproduced (proportionally) in the $< PC1, PC2 >$ plane constructed that way.

Kohonen [4] calls the prototypes *codebook vectors*.

Training the map – to get the prototypes adapted to the density of the data

We wish the prototypes play the role of representatives of the data. To achieve this, the previously established prototypes should have some liberty to move in the feature space and adapt themselves to the data points. This is done during the process called *training*.

The training is performed as a discrete process for $t = 0, 1, \ldots$. During the process of training the prototypes approach the data points and adapt themselves to the (d-dimensional) density distribution of the data. This is done iteratively and may be performed in a *sequential* or a *batch* mode.

Sequential training of the SOM

The prototypes are adjusted after every presentation of a subsequent data vector $\mathbf{x}$(t).

Let $\mathbf{m}_1(t), \ldots, \mathbf{m}_M(t)$ denote the positions of the prototypes at time t. Then, for $t = t + 1$, the following actions are carried out:

(a) Select randomly a data point $\mathbf{x} = \mathbf{x}(t)$ and find its closest prototype (in the meaning of the assumed Euclidean distance). Say, this is the prototype $\mathbf{m}_c$ (subscript 'c' from 'conqueror').

(b) Update all prototypes $\mathbf{m}_1, \ldots, \mathbf{m}_M$ according to the rule

$$\mathbf{m}_i(t) = \mathbf{m}_i(t-1) + \eta(t) g_{i,c}(t)(\mathbf{x}(t) - \mathbf{m}_i(t-1)) \tag{9}$$

where $\eta(t) \in [0, 1]$ is a variable learning rate, usually decreasing with time t .

The process of updating is repeated until a STOP criterion is satisfied. The stop criterion may be based on the maximum number of iterations (steps), or on the accuracy of the updated vectors $\mathbf{m}_i(t)$.

Batch training of the SOM

The adjustment of the prototypes is done in a 'batch', i.e. after presenting a series of N data vectors.

A quotation from documentation of that package ([14], p. 9): '*Also batch training is iterative, but instead of using a single data vector at a time, the whole data set is presented to the map before any adjustments are made.*'

The batch training algorithm implemented in [14] works as follows:

Let $\mathcal{I}(\mathcal{M})$ denote the set of the integers $[1, 2, \ldots, M]$.
Find for each data vector $\mathbf{x_j}$ its nearest prototype $\mathbf{m_c}$, i.e., find such index c ($c \in \mathcal{I}(\mathcal{M})$) that $c = arg\ min_{k \in \mathcal{I}(\mathcal{M})}\{\| \mathbf{x_j} - \mathbf{m_k} \|\}$.

This done, the entire data set is subdivided into M subsets containing $n_{V1}, \ldots, n_{VM}$ elements respectively.

Next, calculate the sum of vectors in each subset:

$$\mathbf{s_i}(t) = \sum_{j=1}^{n_{Vi}} \mathbf{x_j}. \tag{10}$$

Then, the new values of the prototypes $\mathbf{m_i}$ are calculated as:

$$\mathbf{m_i}(t+1) = \frac{\sum_{j=1}^{M} g_{ij}(t)\mathbf{s_j}(t)}{\sum_{j=1}^{M} n_{Vj} g_{ij}(t)}. \tag{11}$$

Notice, that in the formula above the value of the index c is calculated individually for each $\mathbf{x_j}$, thus truly one should write $c = c(j)$.

For calculations presented in this report, we have used the 'batch' mode, as implemented in the SomTB2 package [14].

After the process of training the feature space is subdivided into M disjoint regions, called Voronoi regions. Each Voronoi region contains one prototype. Also, the entire data set is subdivided into M subsets (some of them may be empty). Consider the jth ($j = 1, \ldots, M$) Voronoi region. All data vectors belonging to the jth region have the respective jth prototype $\mathbf{m}_j$ as their closest prototype. In turn, the jth prototype is in one-to-one correspondence with the jth map unit.

Let $\mathbf{x}_k$ be a data vector, and $\mathbf{m}_j$ its closest prototype. Then $\mathbf{x}_k \in V_j$. We say that the jth map unit is the Best Matching Unit (BMU) to the vector $\mathbf{x}_k$.

The quality of the representation of the data in the map may be characterized by two indices: the quantization error q_e, and the topological error t_e.

The *quantization error* q_e is evaluated as the average distance from each data vector to its nearest prototype ([4, 14]).

The *topological error* t_e is evaluated as the fraction of all data vectors for which the first and second nearest prototype are not represented in adjacent units of the map ([4, 13, 14]).

The measure t_e was proposed by Kiviluoto [3]. For others concepts of a topological error, see Venna and Kaski [12] and the references therein.

Remind that the M prototypes are in one-to-one correspondence with the reference vectors from the map.

The map may exhibit various information about the occupancies of Voronoi regions (V_i), or equivalently, occupancies of the map cells (C_i), ($i = 1, \ldots, M$), also information on the mutual distances of prototypes in the multivariate space R^d.

- We may put over each cell (node)
  - counts of the occupancies. These may be shown directly as integers $h_1, \ldots, h_M$, or, schematically, by drawing shaded squares with area proportional to the counts,
  - labels, containing the ID#'s of data vectors belonging to the given cell (node).

  This information may be displayed for the entire data set, or - for its subparts.

- The map may be also painted using different color shades of a chosen color-map. The technique *umat* proposed by Ultsch [11] permits to paint the colors in such way that the color shades in the map reflect the distances of prototype vectors in the feature space R^d. A useful function in Matlab serving this purpose is the function umati; it performs a smoothing of colors over the entire map.

Summarizing:

The map (SOM) is a way of presenting data from a multivariate space in a sheet. The map may be imagined as a structure based on a *grid of neurons*, or, alternatively, as a *lattice of M regular (square) cells*, surrounding the neurons.

Graphically, the map may be represented as a *grid of neurons*, or, alternatively, as a lattice of M regular (square) cells, surrounding the neurons.

The numeration in the map is column-wise, i.e. for a map of size $m_1 \times m_2$ it proceeds in the sequence:

$$1, \ldots, m_1, m_1 + 1, \ldots, (m_2 - 1)m_1, (m_2 - 1)m_1 + 1, \ldots, m_2 m_1,$$

with m_1 denoting the number of rows, and m_2 – the number of columns in the map lattice.

The position of the neurons in the map is characterized by *reference vectors* $\mathbf{r_i}$ ($i = 1, \ldots, M$).

The neighborhood function $g_{i,c}$ defines a score expressing the nearness of $\mathbf{r_i}$ in relation to $\mathbf{r_c}$ ($i, c \in [1, \ldots, M]$).

The *prototypes* (called also *codebook vectors*) are imagined as points located in the feature space R^d. They play the role of representatives of the entire data set.

The entire feature space is subdivided into M disjoint, adjacent regions, called *Voronoi regions*. The number of data points belonging to the $i - th$ region is denoted by n_{V_i}. Obviously

$$n_{V_1} + \ldots + n_{V_M} = N. \tag{12}$$

All n_{Vi} data points ($i = 1, \ldots, M$) belonging to one Voronoi region V_i have a common representative, namely the prototype $\mathbf{m_i}$.

There is a one-to-one correspondence between the prototypes $\mathbf{m_i}$ in the feature space and the reference vectors $\mathbf{r_i}$ (neurons) in the map.

Also, there is a one-to-one correspondence between the Voronoi regions V_i (defined in the feature space) and the map cells C_i (composing the map).

The sequence $V_1, \ldots, V_M$ in R^d has the same numeration as their prototypes $\mathbf{m_1}, \ldots, \mathbf{m_M}$; the same numeration have the reference vectors $\mathbf{r_1}, \ldots, \mathbf{r_M}$, and the map cells $C_1, \ldots, C_M$.

Table of denotations

Meaning	denotation
overall numeration of units	1, 2, ..., M
prototypes	$\mathbf{m_1}, \ldots, \mathbf{m_M}$
reference vectors	$\mathbf{r_1}, \ldots, \mathbf{r_M}$
map cells	$\mathbf{C_1}, \ldots, \mathbf{C_M}$
Voronoi regions	$\mathbf{V_1}, \ldots, \mathbf{V_M}$
occupancies of Voronoi regions,	$n_{V1}, \ldots, n_{VM}$
counts (of map units)	$n_1, \ldots, n_M$
$\parallel$ equivalent $\Updownarrow$	$\parallel$ $\parallel$
hits into map units	$h_1, \ldots, h_M$

5.4 Visualizing data using Generative Topographic Mapping

Generative Topographic Mapping (GTM) is a method which considers probability densities in the Bayesian layout. The method was proposed by Bishop et. al. [1], and elaborated further by Svénsen [10].

The concept of a latent space

We consider data vectors $\mathbf{x} \in R^d$. In practice, the components of these data vectors are correlated. We seek for explanation of the correlations among the components. This is done by introducing the concept of a *latent space* R^L, with $L < d$.

Let $\mathbf{z} \in R^L$ denote generally a data vector (with L components) belonging to the space R^L. We define a statistical model, which links together the two spaces: R^L and R^d. The model assumes that the *observed vector* $\mathbf{x}$ *is originating from a vector* $\mathbf{z}$ *living in the latent space*, and that this fact may explain the correlations stated among the components of $\mathbf{x}$.

First of all, we define a general function $\mathbf{y}(\mathbf{z})$ which maps the vector $\mathbf{z}$ to a vector $\mathbf{y}$ located on a L-dimensional manifold embedded in R^d:

$$\mathbf{y}: \quad \mathbf{z} \rightarrow \mathbf{y}(\mathbf{z}; \mathbf{W}), \tag{13}$$

with matrix $\mathbf{W}$ denoting generally parameters of the not yet fully specified mapping $\mathbf{y}$ (the function $\mathbf{y}$ and its parameters will be described below). In such way the space R^L is mapped to a L-dimensional manifold embedded in the space R^d.

Next we add an axis–aligned Gaussian noise model in the target space R^d. This results in the model

$$\mathbf{x} = \mathbf{y}(\mathbf{z}; \mathbf{W}) + \boldsymbol{\epsilon}, \quad \text{with } \boldsymbol{\epsilon} \sim \mathcal{N}_d(\mathbf{0}, \sigma^2 \mathbf{I_d}). \tag{14}$$

In such way, the two spaces will be linked by a function $\mathbf{y}$ ($\mathbf{z}$; $\mathbf{W}$) which maps $\mathbf{z}$ to $\mathbf{y}(\mathbf{z};\mathbf{W})$ and is parameterized with the matrix $\mathbf{W}$.
The equation [14] tells us that the observed $\mathbf{x}$ is generated by a latent $\mathbf{z}$, to which a random noise was added.

Probability distributions of z and of x and their conditional likelihood

Suppose, we have defined a probability density $p(\mathbf{z})$ in the latent space R^L. This probability density induces a density $p(\mathbf{x}|\mathbf{z}, \mathbf{W})$ in the data space.

From model [14] we may write the data density conditional on the latent variables as:

$$\boxed{p(\mathbf{x}|\mathbf{z}; \mathbf{W}, \sigma) = \frac{1}{(2\pi\sigma^2)^{d/2}} \exp\{- ||\mathbf{y}(\mathbf{z}; \mathbf{W}) - \mathbf{x}||^2/2\sigma^2\}.} \tag{15}$$

Our goal is to find the posterior probability of $\mathbf{z}$, i.e. $p(\mathbf{z}|\mathbf{x})$. To achieve this, we need $p(\mathbf{z})$.

Bishop and Svensén [1, 10], see also [8], have assumed that $p(\mathbf{z})$ may be considered as composed by a *sum of delta* functions centered on nodes $\mathbf{z}_1, \ldots, \mathbf{z}_M$ in latent space:

$$p(\mathbf{z}) \;=\; \frac{1}{M} \sum_{j=1}^{M} \delta(\mathbf{z} - \mathbf{z}_j). \tag{16}$$

If the centers $\mathbf{z}_j$ are uniformly spread in latent space, then the pdf above is an approximation to a uniform distribution.

It may be shown that under [16] the conditional probability $p(\mathbf{x}|\mathbf{z})$ becomes a simple sum of M Gaussians defined above:

$$p(\mathbf{x}|\mathbf{z}) = \frac{1}{M} \sum_{j=1}^{M} p(\mathbf{x}|\mathbf{z}_j, \mathbf{W}, \sigma). \tag{17}$$

Obviously eq. [17] represents a mixture model in which all mixture components are Gaussians with the same variance σ^2 and the same mixing coefficients $1/M$. However this is a *constrained mixture*, because the centers are related by the mapping $\mathbf{y}$ – they all are located on a L-dimensional manifold embedded in R^d.

Thus we have our probability model.

We may define now the **log likelihood** and by maximizing it we may find estimates of the parameters of the model.

For $\mathbf{x_1}, \ldots, \mathbf{x_N}$, an independent sample of size N the log likelihood writes

$$\mathcal{L}(\mathbf{W}, \sigma) = \sum_{n=1}^{N} \ln \{ \frac{1}{M} \sum_{j=1}^{M} p(\mathbf{x}_n|\mathbf{z}_j, \mathbf{W}, \sigma)\} \tag{18}$$

If $\mathbf{y}(\mathbf{z};\mathbf{W})$ were a differentiable function of $\mathbf{W}$, we might find the parameters $\mathbf{W}$ and σ of the pdf $p(\mathbf{x}|\mathbf{z}, \mathbf{W}, \sigma)$ by applying standard non–linear optimization methods (of course, firstly we should specify the function $\mathbf{y} = \mathbf{y}(\mathbf{z}; \mathbf{W})$).

However, because the model consists of a mixture of Gaussians, it is possible to use the **EM** algorithm for training of the model, which is generally easier in our case.

Specifying the mapping $\mathbf{y} = \mathbf{y}(\mathbf{z};\mathbf{W})$

Bishop and Svensén [1, 10], see also Nabney [8], have made the following specification of the mapping function $\mathbf{y}$:

$$\mathbf{y}(\mathbf{z};\mathbf{W}) = \mathbf{W}\,\boldsymbol{\phi}(\mathbf{z})\,, \tag{19}$$
$$d \times 1 = (d \times K) \cdot (K \times 1)$$

where $\boldsymbol{\phi}(\mathbf{z}) = [\phi_1(\mathbf{z}), \ldots, \phi_K(\mathbf{z})]^T$ are K fixed basis functions with centers $\mathbf{c_1}, \ldots, \mathbf{c_K}$, all evaluated at point $\mathbf{z}$. $\mathbf{W}$ is a $d \times K$ matrix.

The equation [19] may be written explicitly coordinate-wise

$$\begin{aligned} y_1(\mathbf{z};\mathbf{W}) &= w_{11}\phi_1(\mathbf{z}) + \cdots + w_{1K}\phi_K(\mathbf{z}) \\ \vdots &= \vdots \\ y_d(\mathbf{z};\mathbf{W}) &= w_{d1}\phi_1(\mathbf{z}) + \cdots + w_{dK}\phi_K(\mathbf{z}) \end{aligned}$$

or, alternatively, as one general equation for $(i = 1, \ldots, d)$

$$y_i(\mathbf{z};\mathbf{W}) = w_{i1}\phi_1(\mathbf{z}) + \cdots + w_{iK}\phi_K(\mathbf{z}) = \sum_{k=1}^{K} \phi_k(\mathbf{z})\, w_{ik}.$$

One may see that the ith coordinate is determined as linear combination of the ith row of $\mathbf{W}$ and of the vector $\boldsymbol{\phi}(\mathbf{z})$, the last being a nonlinear function of $\mathbf{z}$.

Estimating the parameters W of the GTM model

It is relatively easy to estimate the parameters of the assumed model using the EM algorithm [10, 8]. This is an iterative algorithm, which guaranties convergence to a (local) minimum. Each iteration performs two steps: the E step and the M step.

In the EM algorithm it is important to have some good initial starting parameters. These could be taken from the condition below:

$$E = \frac{1}{2}\sum_{j=1}^{M} \parallel \mathbf{W}\boldsymbol{\phi}(\mathbf{z_j}) \,-\, \mathbf{U}\mathbf{z_j} \parallel^2,$$

where $\boldsymbol{\phi}(\mathbf{z_j})^T$ denotes the jth row of the matrix $\boldsymbol{\Phi}$ defined in eq. [22], and the columns of $\mathbf{U}$ contain the relevant (in our case: $L = 2$) eigenvectors of the data covariance matrix.

Suppose, at iteration m we got some values $\mathbf{W}^{(m)}$ and $\sigma^{(m)}$ of the parameters. Then

E − step is the same as for Gaussian mixtures.

We calculate the *posterior probability* (*responsibility*) $R_{jn}^{(m)}$ of each component j for each data point $\mathbf{x_n}$ (i.e. that the jth component in R^L has generated the $\mathbf{x_n}$th observation)

$$\boxed{R_{jn}^{(m)}(\mathbf{W}^{(m)}, \sigma^{(m)}) \;=\; P(j|\mathbf{x_n}, \mathbf{W}^{(m)}, \sigma^{(m)}) = \frac{p(\mathbf{x_n}|\mathbf{z_j}, \mathbf{W}^{(m)}, \sigma^{(m)})}{\sum_{j'=1}^{M} p(\mathbf{x_n}|\mathbf{z_{j'}}, \mathbf{W}^{(m)}, \sigma^{(m)})}} \tag{20}$$

M – step we maximize the expectation of the complete likelihood with respect to the parameters $\mathbf{W}$, σ. The responsibilities R_{jn} act here as weights in the update equations for $\mathbf{W}$ and σ.

The step E is obvious. Step B needs some elaboration.
The *complete data log–likelihood* in the (m)th iteration is given by the equation

$$< \mathcal{L}_{com}(\mathbf{W}, \sigma) >= \sum_{n=1}^{N} \sum_{j=1}^{M} R_{jn}^{(m)}(\mathbf{W}^{(m)}, \sigma^{(m)}) \ln \{p(\mathbf{x_n}|\mathbf{z_j}, \mathbf{W}, \sigma)\} . \tag{21}$$

In the equation above, the $\mathbf{W}^{(m)}, \sigma^{(m)}$ appearing in the responsibilities are assumed as fixed and known; however the parameters $\mathbf{W}, \sigma$ appearing in the conditional probabilities $p(\mathbf{x_n}|\mathbf{z_j})$ are unknown and they constitute the parameters to be optimized so as to yield the maximum of the complete likelihood. This leads to the following GLS (General Least Squares) equation for $\mathbf{W}$:

$$\mathbf{\Phi}^T \mathbf{G}^{(m)} \mathbf{\Phi}\, \mathbf{W}^T \;=\; \mathbf{\Phi}^T \mathbf{R}^{(m)} \mathbf{X}, \tag{22}$$

with dimensions of succeeding components

$$(K \times\ M) \cdot (M \times M) \cdot (M \times K) \cdot (K \times d) \;=\; (k \times M) \cdot (M \times N) \cdot (N \times d).$$

The solution of the [22] yields the updated solution $\mathbf{W}^{(m+1)}$.

The matrices appearing in [22] have the following meaning:

$\mathbf{\Phi}$: known values of basic functions evaluated at centers $\mathbf{z_1}, \ldots, \mathbf{z_1}$; need to be evaluated only once, at the begin of the iterations,
$\mathbf{\Phi} = \{\phi_{ji}\} = \{\Phi_i(\mathbf{z_j}\})$, $j = 1, \ldots, M$, nb of nodes, $i = 1, \ldots, K$, nb of RBF centers.
The matrix $\mathbf{\Phi}$ has the following structure:

j↘i	1	2	...	K	RBF centers
1					
nodes ⋮			$\phi_{ji} = \Phi_i(\mathbf{z_j})$		
M					

$\Phi_i(\mathbf{z_j})$ denotes the value of the ith RBF evaluated at node $\mathbf{z_j}$.

G: known diagonal matrix $diag\{G_{jj}\}$, $(j = 1, \ldots, M)$, with $G_{jj} = \sum_{n=1}^{N} R_{jn}$,

W: unknown weights, to be found, $\{w_{kh}\}$, $(k = 1, \ldots K,\ h = 1, \ldots, d)$

R: known responsibilities $\{R_{jn}\}$, $(j = 1, \ldots M,\ n = 1, \ldots, N)$,

X: known data matrix $\{x_{nh}\}$, $(n = 1, \ldots N,\ h = 1, \ldots, d)$.

In the same step also the parameter σ gets updated:

$$(\sigma^{(m+1)})^2 \;=\; \frac{1}{Nd} \sum_{n=1}^{N} \sum_{j=1}^{M} R_{jn}(\mathbf{W}^{(m)}, \sigma^{(m)}) \;\| \mathbf{W}^{(m+1)} \boldsymbol{\phi}(\mathbf{z_j}) - \mathbf{x_n} \|^2 . \tag{23}$$

Notice that in the above updating formula, the responsibilities are the old ones, computed in the mth E step, but the squared error for the nth observation is computed using the adjusted weights $\mathbf{W}^{(m+1)}$.

It can be shown that the data likelihood increases at each iteration – until a local maximum is reached. Thus the algorithm is convergent.

Quoted after Nabney [8]: If the RBFs have a large number of degrees of freedom, the mapping $\mathbf{y}(\mathbf{z};\mathbf{W})$ is overly complex and the established manifold may exhibit higher curvature. One way to prevent this is to add a weight decay regularization term $\lambda \sum_{i=1}^{K} \sum_{h=1}^{d} w_{ih}^2$ to the error function. This leads to a small modification of the M step

$$\left(\mathbf{\Phi}^T \mathbf{G}^{(m)} \mathbf{\Phi} + \lambda \mathbf{I}\right) (\mathbf{W})^T \;=\; \mathbf{\Phi}^T \mathbf{R}^{(m)} \mathbf{X}, \tag{24}$$

Visualizing the posterior probabilities

Assuming $q = 2$, the method may be applied for visualization od the data in a 2-dimensional plane.

We may visualize either the posterior densities $p(\mathbf{z}|\mathbf{x_n})$ (these are given by a sum of delta functions centered at the lattice points $\mathbf{z_j}$), or the expected means of these distributions given as

$$< \mathbf{z}|\mathbf{x_n} > = \sum_{j=1}^{M} R_{jn} \mathbf{z}_j. \tag{25}$$

The responsibilities R_{nj} are probabilities *à posteriori* evaluated during the E-step of the EM algorithm (see eq. [20]): $R_{jn} \;=\; R_{jn}(\mathbf{W}, \sigma) \;=\; P(j|\mathbf{x_n}, \mathbf{W}, \sigma)$.

Magnification factors

Quoted after Nabney [8]: The EM algorithm will attempt to place the mixture components in regions of high density and will move the components away from regions of low density. It can do it, because the non-linear mapping from latent space to data space enables the manifold to stretch across regions of low data density. The *stretching* (or *magnification*) can be measured using techniques of differential geometry.

To determine the magnification factor we need to work out the changes in small (infinitesimal) volume dV and dV' when passing from R^L to the manifold in R^d. It is shown [8] that this change is given by the equation

$$\frac{dV'}{dV} \;=\; [det(\mathbf{\Psi} \mathbf{W}^T \mathbf{W} \mathbf{\Psi}^T)]^{1/2}, \text{ where } \mathbf{\Psi} \;=\; \{\psi_{ji}\} \;=\; \left\{ \frac{\partial \Phi_j}{\partial \mathbf{z_i}} \right\}. \tag{26}$$

The right term ($\sqrt{(det)}$) in the above equation is called *magnification factor*.

The magnification factors are computed for each node in R^L. Thus each unit in the map has assigned its corresponding stretching factor, when mapping to the data space.

Next, the stretching factors (their magnitudes) are combined with a palette of colors (color-map). For practical reasons (printing the map in gray-scale) it is advisable to choose the color-map in such a way that increasing stretching is indicated by increasing darkness of the colors.

The Matlab function imagesc permits for smoothing the colors over the map.

Implementation in Netlab

GTM

function net = gtm(dimLatent, nlatent, dimData, ncentres, rbfunc, PRIOR)
Creates a Generative Topographic Map.

Input parameters

The first 5 input parameters are mandatory, the 6th is optional:

dimLatent	:	L, dimension of the latent space,
nlatent	:	M, number of nodes (delta functions) in latent space,
dimData	:	d, dimension of the data space,
ncentres	:	K, number of RBF centers in latent space
rbfunc	:	type of RBF function
PRIOR	:	optional, for determining width of RBF functions

Created structure

The function GTM returns a *map* structure net with the following fields:

type	:	'gtm'
nin	:	dimension of data space
dimlatent	:	dimension of latent space
rbfnet	:	RBF network data structure
gmmnet	:	GMM data structure
X	:	sample of latent points

The parameters in the RBF and GMM sub-models are set by calls to the corresponding creation routines RBF and GMM.

The function gtm may be called with, or without the last parameter prior. PRIOR sets a Gaussian zero mean prior on the parameters of the RBF model.

PRIOR must be a scalar and represent the inverse variance of the prior distribution. This gives rise to a weight decay term in the error function.

GTMINIT

function net = gtminit(net, options, data, sampType, varargin)
Takes an existing GTM net and initializes the weights. Also generates a sample of latent data points (nodes in R^L) and sets the centres (and widths if appropriate) of NET.RBFNET.

The function works only for $L = 1$ or $L = 2$.

Input parameters

The first 4 input parameters are mandatory; depending on the value of the input parameter sampType there may be other succeeding parameters:

net	:	name of the existing structure 'net' created tb the function gtm,
options	:	a vector specifying options of calculations, see explanations below,
data	:	training data,
sampType	:	can be 'regular, 'uniform' or 'gaussian',
varargin	:	optional, used only when 'sampType' is regular

The vector of '**options**' needs here only option(7), which determines the width of the RBF basic functions. Default is: options(7)=1. The widths of the RBF basis functions are set by a call to rbfsetw passing OPTIONS(7) as the scaling parameter.

Meaning of some other components of 'options' are explained when presenting the function 'gtmem' below.

It was said above that gtminit creates the latent data points ('latent data sample, 'nodes') and RBF centres are created. The numbers of the nodes and of the RBF centres were already declared as input to the 'gtm' function.

If the 'sampType' is 'UNIFORM' or 'GAUSSIAN' then the latent data is found by sampling from a uniform or Gaussian distribution correspondingly. The RBF basis function parameters are set by a call to RBFSETBF with the DATA parameter as dataset and the OPTIONS vector.

If the SAMPTYPE is 'REGULAR', then *regular grids* of latent data nodes and RBF centres is created; the user should specify lengths of the grid sides. For a two-dimensional latent space, these parameters must be vectors of length 2 with the number of points in each of the x and y directions to create a rectangular grid.

Finally, the output layer weights of the RBF are initialized by mapping the mean of the latent variable to the mean of the target variable, and the L-dimensional latent variable variance to the variance of the targets along the first L principal components.

GTMEM

function [net, options, errlog] = gtmem(net, t, options)

Uses the EM algorithm to estimate the parameters of a GTM model defined by a data structure NET.

The matrix T represents the data whose expectation is maximized, with each row corresponding to a vector. It is assumed that the latent data NET.X has been set following a call to GTMINIT, for example. The optional parameters have the following interpretations.

OPTIONS(1) is set to 1 to display error values; also logs error values in the return argument ERRLOG. If OPTIONS(1) is set to 0, then only warning messages are displayed. If OPTIONS(1) is -1, then nothing is displayed.

OPTIONS(3) is a measure of the absolute precision required of the error function at the solution. If the change in log likelihood between two steps of the EM algorithm is less than this value, then the function terminates.

OPTIONS(14) is the maximum number of iterations; default 100.

The optional return value OPTIONS contains the final error value (i.e. data log likelihood) in OPTIONS(8).

Other useful functions: gtmprob, gtmpost, gtmlmeans, gtmlmode

prob	=	gtmprob,	computes $p(\mathbf{x}\|\mathbf{z}, \mathbf{W}, \sigma)$, formula (15),
[post a]	=	gtmpost,	computes responsibilities, formula (20),
means	=	gtmlmeans,	posterior density $p(\mathbf{z}\|\mathbf{x_n})$
modes	=	gtmlmode,	posterior mode.

References

[1] Bishop C.M., Svensèn M., and Williams C. K. I. (1996). The generative topographic mapping. *Neural Computation* **10** (1), 215–235.

[2] Dougherty E.R., Barrera J., et al. (2002) Inference from clustering with application to gene-expression microarrays. *Journal of Computational Biology* **9**, Nb. 1, 105–126.

[3] Kiviluoto K. (1996). Topology preservation in self-organizing maps. Proceedings ICNN'96, V. 1, IEEE Neural Networks Council, June 1996, Piscataway, New Jersey, USA, pp. 294–299.

[4] Kohonen T. (1995). Self-Organizing Maps. Springer Series in Information Sciences, V. 30, Berlin.

[5] Kohonen T., Hynninen J., Kangas J., Laaksonen J. (1995). SOM_PAK, The self-Organizing Map Program Package. Version 3.1 (April 7, 1995). Laboratory of Computer and Information Science, Helsinki Univ. of Technology, Espoo, Finland.

[6] Mardia, K. V. (1970). Measures of multivariate skewness and kurtosis with applications. *Biometrica* **57**(3), pp. 519 – 530.

[7] Mardia, K. V. Applications of some measures of multivariate skewness and kurtosis in testing normality and robustness studies. *Sankhya: The Indian Journal of Statistics* 36, 1974,B Pt. 2,pp. 115-128.

[8] Nabney I.T. (2001). *Netlab: Algorithms for Pattern Recognition*. Springer London, Berlin, Heidelberg. Springer Series: Advances in Pattern Recognition.

[9] Rousseeuw P.J., van Driessen K. (1999). A fast algorithm for the minimum covariance determinant. *Technometrics* **41**, pp. 212 – 223.

[10] Svensén J. F. M. (1998). GTM: The generative topological mapping. Ph.D. dissertation. Aston University. Available as NCRG/98/024 at http://www.ncrg.aston.ac.uk/

[11] Ultsch A. (1993). Self-organizing neural networks for visualization and classification. Opitz O.,Lausen B., and Klar R. (edts): Information and Classification. Berlin, Springer-Verlag, 307–313.

[12] Venna J., Kaski S. (2001). Neighborhood preservation in nonlinear projection methods: An experimental study. In: Proceedings ICANN 2001, Vienna. Edited by G. Dörffner, H. Bischof, and K. Hornik. Springer, Berlin, pp. 485 – 491.

[13] Vesanto J. (1999). SOM–based data visualizing methods. Intelligent Data Analysis, **3**(2), pp. 111 – 126.

[14] Vesanto J., Himberg J., et al. (2000). SOM Toolbox for Matlab 5. SOM Toolbox Team, HUT, Finland, Libella Oy, Espoo, pp. 1 – 54. `http://www.cis.hut.fi/projects/somtoolbox`, Version 0beta 2.0, Nov. 2001.