Laboratory of Data Analysis
University of Jyväskylä

EUREDIT - WP4.5, WP5.5 Internal reports

READ THIS BEFORE D4.5.1

D5.5.1 - Description of The Imputation Methodology based on The Tree-Structured Self-Organizing Map

(Draft Version 29. March 2002)

Pasi P. Koikkalainen - **University of Jyväskylä (theory)**
Pasi Piela - **Statistics Finland (Appendices A,B)**
Seppo Laaksonen - **Statistics Finland (Appendix A)**

Contents

Appendices

Summary

This report describes how the tree-structured self-organizing map (TS-SOM) can be used for imputation of incomplete data sets. The text begins with a rough introduction of the SOM and TS-SOM type of neural networks, and is followed by the description of the TS-SOM based imputation methodology.

In the neural networks litterature there are many, often conflicting intepretations about the role of the self-organizing map. The reason for confusion is that there is no generally accepted objective for the method, only implementations, from which it is hard to make a distinction between the fundamental idea and the choises made by the computer programmer. In this paper we assume that the self-organizing map is a discrete approximation of a more gereric class of models, principal latent surfaces. This is useful for our purposes, since it allows us to think the SOM as a nonlinear, unsupervised regression model.

The TS-SOM is a fast and easy to use variant of the SOM. When all computational tricks are used, our TS-SOM is faster to organize than any other clustering algorithm that we know, although this might not be be true if the same speedup techniques would be used with other algorithms. The computational complexity of the algorithm allows us to solve may real world problems that could otherwise be solved with unrealisticly simple models only. By ease of use we mean that TS-SOM, unlike most other neural networks algorithms, is not sensitive to any implementation specific parameters. We do not need to play with training parameters to get TS-SOM well organized.

When using SOM type of learning algorithms for imputation, several alterntive approaches can be taken. For example, we may use fully observed part of data for model building, and then use the model as a map from observed variables to incomplete ones. The actual imputation of values may simulate randomness around the model, or it may copy the missing values from the observed samples. This type of methodology does not nessessary require the development of new algoritms, instead it is a hyprid of basic tools of data analysis and statistical methods. This approach, however, is limited by the availability of good fully observed data. The more ambiqutious way is to modify the existing TS-SOM algorithms to be able to hande incomplete and erroneous data as well.

In this report we present a novel training algorithm for the tree-structured self-organizing map. The new algorithm is capable of utilizing all information of the data set, including the situation that no fully observed data vectors are available. The algorithm is derived as an incomplete likelihood estimate for mixture models, which is our intepretation of the discrete SOM model. Unlike in most training algorithms for incomplete data, we do not assume anything about missing values, i.e. we do not explicitely build expected likelihood or impute during the training. Intrinsicly, however, the our method is a variant of the EM-algorithm. When the method is fully impemented, it is not much slower than the basic TS-SOM, and more importantly, its time complexity is the same with respect to the number of clusters (neurons) and data records.

Finally, using a toy example the key characteristics of the meethodology are demonstraed. A more troughout evaluation of the methodology is given in the documents, related to EurEdit data sets Appendix 1 (DLFS), Appendix 2 (Sars), Appendix 3 (UK ABI Data), and Appendix 4 (Swiss). Appendices 3 and 4 are in preparation (not included here). Another closely related document is D4.5.1 explains how the TS-SOM algorithm can be used for error localication.
The methods are fully implemented under our NDA (Neural Data Analysis) software as described in the document (D4.5.2 & D5.5.2)

1 Introduction

In this section a brief review of Kohonen's self-organizing map (Kohonen 1982) is given. Special attention is made to introduce those characteristics and variations of the basic method that are later needed in the derivation of the SOM training algorithm for incomplete data.

Note that this kind of description is not fully accecpted in the literature of the self-organizing map. Some developers of the SOM methodology do not want to make any connections between the SOM and principal curves, MDL, or mixture models. Here those connections are essential, and based on the personal viewpoint of the author, see (Koikkalainen, 1999). Similar formulations of the SOM can also be found from other authors, for example in (Utsugi, 1997), (Cherkassky 1995) and (Hastie, Tibsirani and Friedman 2001).

1.1 Why neural networks for editing and imputation ?

The following motivation is adopted from Vladimir N. Vapnik (Vapnik 1995):

> *Ronald A. Fisher simplified the problem of statistical inference -estimating probability measures- by reducing it to the problem of estimating parameters of density function.*
>
> *As the information technology revolution provided opportunities for estimating high-dimensional functions (in 1960s) the problem of "curse of dimensionality was discovered":* `Dense samples as needed to learn` **pdf** `well, but dense samples are hard to get in high dimensions.`
>
> *A neural network solution to this problem was discovered first time in the early 1990s.*

What this means is that many traditional methods do not work well with high dimensional data. In 1960's people started to believe that this is a law of nature and nothing can be done. However we know due Kolmogorov's Theorem (Hilberts 13th problem) that that any continuous function $f(x_1, x_2, \ldots, x_M)$ in $\mathbb{R}^M$ can be presented using a one dimensional function $g_f(u)$

$$f(x_1, x_2, \ldots, x_M) = \sum_{j=1}^{2M+1} g_f[\sum_{i=1}^{M} \lambda_i \phi_j(x_i)],$$

where

$\{\lambda_i\}_{i=1}^{M}$ = universal constants (don't depend on f).
$\{\phi_j(\xi)\}_{i=1}^{M}$ = universal transformation (don't depend on f).
$g_f(u)$ = continuous 1-D function. It totally characterizes $f(x_1, x_2, \ldots, x_M)$.

Informally we may intepret this such that for any high dimensional estimation problem there is an equivalent one dimensional problem. Therefore the problem of "*curse of dimensionality*" does not exist in nature.

It is true that some tasks are more difficult than others and often the difficult ones are multi-dimensional. But **dimensionality of not the problem**, rather the real underlying problem is the **COMPLEXITY** of data, as noted in (Friedman 1995).

The benefit of neural networks is that they can be used, under some implementation limitations, for any dimensional problems. The self-organizing map, for example, combines dimension reduction and data modelling under a single learning algorithm.

In practice there is no free lunch as noted in **??**. While some old problems are solved elegantly with neural networks, some new ones are introduced. Most notably, it is hard to give an exact meaning for

neural network parameters, and therefore traditional ways, based on parameter significance, cannot be used for model validation.

1.2 Principal curves and surfaces and data complexity

The objective of any SOM algorithm is best explained via principal curves and surfaces (Hastie and Stuezle 1989, LeBlanc and Tibshirani (1990)) that generalize the idea of SOM as regression surfaces of type

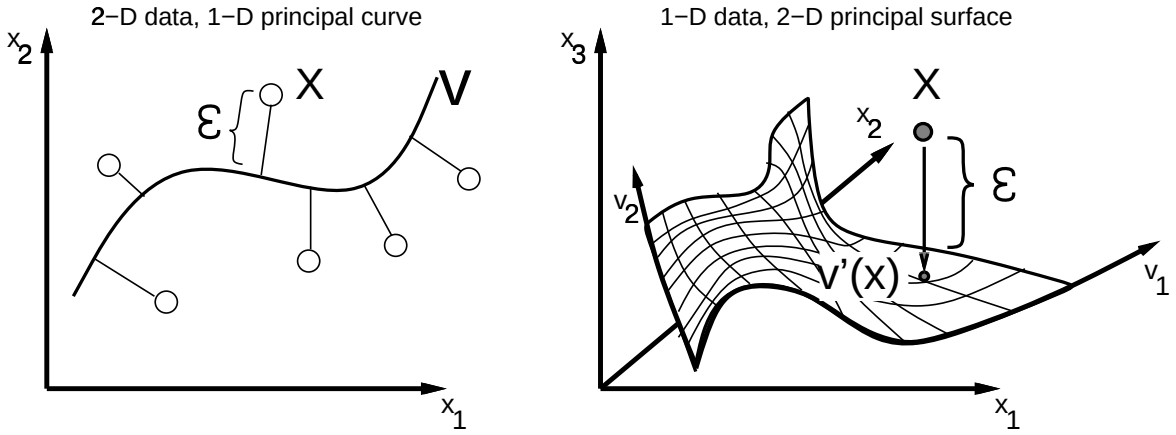
$$\mathbf{x}(\mathbf{v}) = \mathbb{E}[\mathbf{X}|\mathbf{v}'(\mathbf{X}) = \mathbf{v}], \tag{1}$$

where M dimensional data vectors $\mathbf{x} \in R^M$ are projected onto lower $S < M$ dimensional latent surface $\mathbf{v} \in R^S$ via mapping

$$\mathbf{v}'(\mathbf{x}) = \arg\min_{\mathbf{v}''} \|\mathbf{x} - \mathbf{x}(\mathbf{v}'')\|$$

as illustrated in Fig. 1.

Chart 1: The mapping of a vector $\mathbf{X}$ onto principal surface $\mathbf{v}$.



To represent data accurately the surface tends to fold inside data, and in the extreme case without restrictions, the surface $\mathbf{v}$ may go trough all N samples $\{\mathbf{X}(j)\}_{j=1}^N$ of a discrete data set. To control the folding and to quarantee that a stable solution exists, the flexibility of principal curves and surfaces must be penalized (regularized). If one expects that the solution must be smooth, then one but not the only possibility, is to use a kernel smoother, which allows us to rewrite equation (1) in form

$$\mathbf{x}(\mathbf{v}) = \int H(\mathbf{v} - \mathbf{z})\mathbb{E}[\mathbf{X}|\mathbf{v}'(\mathbf{X}) = \mathbf{z}]d\mathbf{z}, \tag{2}$$

where kernel function sums to unity $\int H(\mathbf{v})d\mathbf{v} = 1$. The most obvious choice is to use a gaussian density

$$H(\mathbf{v}|\sigma_v) = \frac{1}{(2\pi\sigma_v^2)^{S/2}} \exp\left\{-\frac{\mathbf{v}^2}{2\sigma_v^2}\right\}$$

as a kernel, where parameter σ_v controls the degree of smoothing.

In practice, and due the smoothing, we expect that there will be an error

$$\epsilon = \mathbf{x}(\mathbf{v}'(\mathbf{X})) - \mathbf{X},$$

which is zero mean because equation (2) defines a regression model. The error term ϵ is important since it represents the randomness of data around our model $\mathbf{x}(\mathbf{v})$. A natural objective is then to find such a model that separates the behavior of underlying, but unknown, true phonema from the residual noise of measurements (samples). In neural networks, one tries to do this univerally, without assumptions about the true model. Although it is practically impossible to find the model this way, it is still possible to find good models by minimizing our uncertainty about them. Informally, we try to do as good models as data allows, which is often quoted using Occam's razor: "simplest explanation is the best".

The "best" model, from the information theoretic point of view, is the one that minimizes the joint complexity of the model and residual randomness ϵ

$$\mathrm{Comp}[\epsilon, \mathbf{x}(\mathbf{v})] = \mathrm{Comp}\underbrace{[\epsilon|\mathbf{x}(\mathbf{v})]}_{\text{residual}}\,\mathrm{Comp}\underbrace{[\mathbf{x}(\mathbf{v})]}_{\text{model}}.$$

In probabilistic terms, the concept of complexity, Comp, is quite similar to likelihood, which nicely relates information theoretic methods like the Minimal Description Lenght (Rissanen 1989) (MDL) to other model selection methods like the Akaike Information Criteria (Akaike 1977) (AIC). Like likelihood, the complexity is often measured in log form, $\log \mathrm{Comp} = \mathrm{L}$, which can be explained as a lenght L of a description, i.e. computer program, that is required to represent the term.

When using MDL for principal curves and surfaces one tries to measure the description lenght of data $\mathcal{D} = \{\mathbf{X}\}$ via "best" model such that

$$\mathrm{L}(\mathcal{D}) = \min_{\mathbf{x}(\mathbf{v})} \left(\mathrm{L}\underbrace{[\epsilon|\mathbf{x}(\mathbf{v})]}_{\text{residual}} + \mathrm{L}\underbrace{[\mathbf{x}(\mathbf{v})]}_{\text{model}} \right).$$

The problem is to measure $\mathrm{L}[\epsilon|\mathbf{x}(\mathbf{v})]$ and $\mathrm{L}[\mathbf{x}(\mathbf{v})]$ in practice. To do this we must understand howto implement principal curves and surfaces, and howto quantify their complexity.

2 The self-organizing map

The self-organizing map can be understod as a discretezed implementation of a principal surface, as depicted in Fig. 2. In self-organizing map the continuous surface $\mathbf{v}$ is represented with a discrete lattice of points $\mathbf{v}_i$, $i \in I$, where I is a set of node (neuron) indexes. Therefore the cardinality of the indexing set I is also the number of nodes in the SOM lattice.

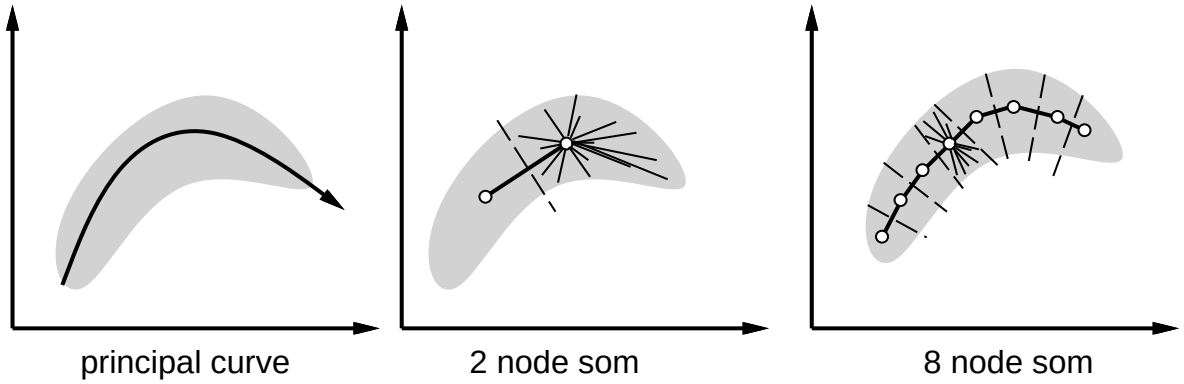
In the SOM algorithm the locations of the nodes $\mathbf{v}_i$ in data coordinates are parametrized via weight vectors $\mathbf{w}_i = \mathbf{x}(\mathbf{v}_i)$ and the projection $\mathbf{v}'(\mathbf{X})$ from data onto the principal surface is replaced with a discrete search

$$b(\mathbf{X}(j)) = \arg\min_i \|\mathbf{w}_i - \mathbf{X}(j)\|,$$

where b is the *best matching node* for a given sample $\mathbf{X}(j) \in \mathcal{D}$.

The heart of the SOM algorithm is the updating rule for the weight parameter vectors. The complexity of the SOM is again controlled via a smoother, as well as with the number of nodes of the lattice, resolution, which implicitely defines a limit of how rich the solution can be. We shall later see that the resolution of the SOM is en essential tool for us, when selecting the best possible model for the imputation of incomplete data.

Chart 2: The relation between a principal curve and its discrete impementation via the self-organizing map.



In discrete form the equation (2) of the smoothed principal curve can be written through a Naradaya-Watson type of kernel estimator (see (**?**)). This is a weighted expectation of type

$$\mathbf{w}_i = \sum_k h_{i,k} \mathbb{E}[\mathbf{X} \mid b(\mathbf{X}) = k], \tag{3}$$

where the value of the kernel is computed in discrete lattice points such that $h_{i,k} = \frac{H(\mathbf{v}_i - \mathbf{v}_k|\sigma_v)}{\sum_k H(\mathbf{v}_i - \mathbf{v}_k|\sigma_v)}$.

In the literature there are several SOM training rules to estimate equation (3) from data. Many of them have been derived in an "ad hoc" manner. Yet, many of them do not actually solve (3), but some, often unknown variation of it. In the early days, most of the proposed algorithms were based on Robbins-Monro type of stochastic optimization, while the best algorithms today are batch algorithms, variations of the famous *Expectation Maximation* (EM) algorithm. This is an important observation because it introduces **a possibility to derive an elegant incomplete data training algorithm for self-organizing maps**.

The SOM is often explained as a vector quantization algorithm, but in our context it is more useful to think it as a mixture model, as shown in Fig. 3.

The SOM nodes (neurons) are now gaussian densities

$$f_{\mathbf{X}}(\mathbf{x}|\mathbf{v_i}, \mathbf{A}_i) \sim \exp\left\{-\frac{1}{2}(\mathbf{x} - \overline{\mathbf{w}}_i)^T \mathbf{A}_i^{-1} (\mathbf{x} - \overline{\mathbf{w}}_i)\right\},$$

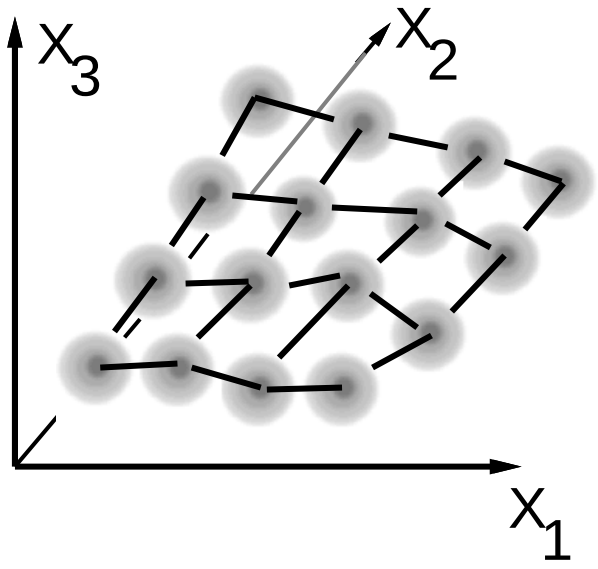
where matrix $\mathbf{A}_i$ defines the widths (variances) of the generator. In practice, however, we usually assume that $\mathbf{A}_i$ is diagonal (only diagonal elements are nonzero), which simplifies the use of the SOM in practice, allowing us to compute all dimensions $x_1, x_2, \ldots, x_M$ independently. Futher simplification is possible if we assume that node density variances are zero. Then the computation of variances can be ignored, and all samples will be classifed into "hard", nonoverlapping Voronoi regions around the *best matching nodes*, as it is done in almost all SOM algorithms.

The likelihood function for the mixture version of the SOM can now be written as

$$\mathcal{L}(\mathbf{W}; \mathcal{D}) = \prod_j \left\{ \sum_i \sum_k h_{i,k} f_{\mathbf{X}}(\mathbf{X}(j)|\mathbf{v}_k, \mathbf{A}_k) \mathtt{Pr}(k) \right\}.$$

It is then a simple task to derive the maximum likelihood solution for the SOM mixture model. The

Chart 3: Illustration of the SOM as a mixture model, where each node is a Gaussian generator.



solution is similar to standard mixture update, execpt that the centroids are also smoothed on the SOM lattice as follows

$$\frac{\partial \mathcal{L}(\mathbf{W})}{\partial \mathbf{w}_i} = 0 \quad \Rightarrow \quad \mathbf{w}_i = \frac{1}{\sum_k h_{i,k} N_k} \sum_k h_{i,k} N_k \overline{\mathbf{x}}_k, \tag{4}$$

where

$$\overline{\mathbf{x}}_k = \frac{1}{N_k} \sum_{j \in \Omega_k} \mathbf{X}(j). \tag{5}$$

Equation (5) is a mean of classified samples (using maximum probability $\mathtt{Pr}(k|\mathbf{X}(j))$) to component k, which are given as a set

$$\Omega_k = \{j \mid b(\mathbf{X}(j)) = k\}, \tag{6}$$

where N_k is its cardinality (therefore the prior probability of component k is $\mathtt{Pr}(k) = \frac{N_k}{N}$). Note the difference to standard mixture model, where we compute probabilities $\mathtt{Pr}(k|\mathbf{X}(j))$ for all nodes, and then use update $\overline{\mathbf{x}}_k = \frac{1}{\sum_j \mathtt{Pr}(k|\mathbf{X}(j))} \sum_j \mathtt{Pr}(k|\mathbf{X}(j))\mathbf{X}(j)$.

2.1 The SOM algorithm in practice

The problem with the maximun likelihood updating rule (4) is that it requires the knowledge of which of the samples $\mathbf{X}(j)$ will be associated to which of the nodes v_k. Actually we do not have this information before the training is complete, and therefore the training algorithm must deal with incomplete information. The usual way to solve the problem is to use the EM-algorithm.

In the EM-algorithm the unknown likelihood $\mathcal{L}(\mathbf{W})$ is replaced with an expected likelihood $Q(\mathbf{W}^{t+1}|\mathbf{W}^t)$, where $\mathbf{W}^t$ is the set of parameters from the previous epoch t. When the maximum

likelihood (ML) solution for parameters $\mathbf{W}^{t+1}$ is computed, the likelihood changes to $Q(\mathbf{W}^{t+2}|\mathbf{W}^{t+1})$. This is then repeated until algorithm reaches convergence, which it is guaranteed to do.

Typical SOM the estimation procedures are loosely based on this idea. The follwing two steps are repeated until the change between $\mathbf{W}^{t+1}$ and $\mathbf{W}^{t}$ is small enough.

1. Distribute the samples between the nodes as defined by the equation (6). This differs from the standard EM-algorithm by selecting only one node, the maximum probability one, for each sample. In a special case where the variances of the components are close to zero, there is no difference between the two.
2. Update nodes using equations (5) and (4). This is a normal maximum likelihood optimization step which differs from normal mixture model update only by the introduction of the kernel smoother $h_{i,k}$ as an additional weighting.

The actual training algorithm is summarized in Algorithm 1. It should be noted, however, that in practice some details, for example the initialization of the node positions, are done differently.

Algorithm 1: Batch algorithm for the self-organizing map.

1. `Initialize (e.g. with random positions):`
   `epoch` $t = 0$
   $\forall_i : \mathbf{w}_i(t) = \mathbf{X}(j) \quad j = \mathrm{rand}(1, 2, \ldots N)$

2. `Divide data into Voronoi regions` $\Omega_i(t)$
   `such that`

   `FOR` $j = 1, 2, \ldots, N$:
   $j \in \Omega_i(t)$ `if` $i = \arg\min_k \|\mathbf{X}(j) - \mathbf{w}_k\|$

3. `Compute centroid means:`
   $\overline{\mathbf{x}}_i = \frac{1}{N_i} \sum_j \mathbf{X}(j)$, `where` $j \in \Omega_i(t)$
   `and` $N_i = \#\Omega_i$

4. `Update centroids (use smoother along SOM lattice):`

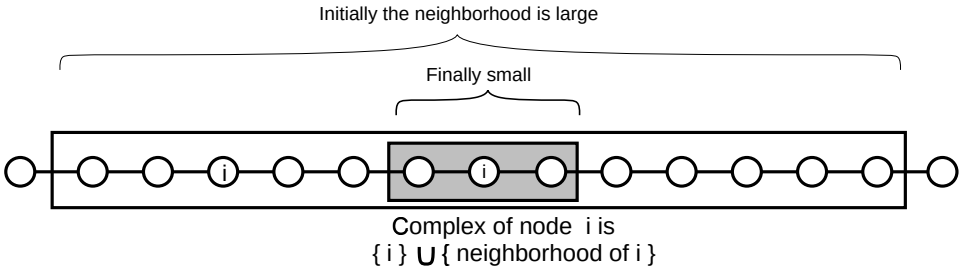
   `FOR` $i = 1, 2, \ldots, P$:
   $$\mathbf{w}_i(t+1) = \frac{1}{\sum_k h_{i,k}(t) N_k} [\textstyle\sum_k h_{i,k}(t) N_k \overline{\mathbf{x}}_k],$$

5. `Test if converged:` $\forall_i : |\mathbf{w}_i(t+1) - \mathbf{w}_i(t)| \leq \delta$
   `else do next epoch` $t := t + 1$; GOTO 2.

In the SOM literature the kernel smoother $h_{i,k}(t)$ is known as the neighborhood function. On the SOM lattice it defines a set of nodes, a complex, that cosists of the node and its closest neighbors, as depicted in Fig. 4. Small number of nodes corresponds to small width of the smoother, while large width averages over many nodes. In most SOM implementations the smoother width is changing as the training proceeds. It is recommended that large smoothing is used in the beginning, to initialize the network, and small width in the final phases to fine tune the net. Unfortunately there is no objective criteria this.

In this presentation many practical issues about the actual SOM implementation are omitted. In practice the most difficult task is to select the proper number of nodes, and to adjust the smoother width of smoothing during the training.

Chart 4: Complex of node i: neighborhood plus the node itself.



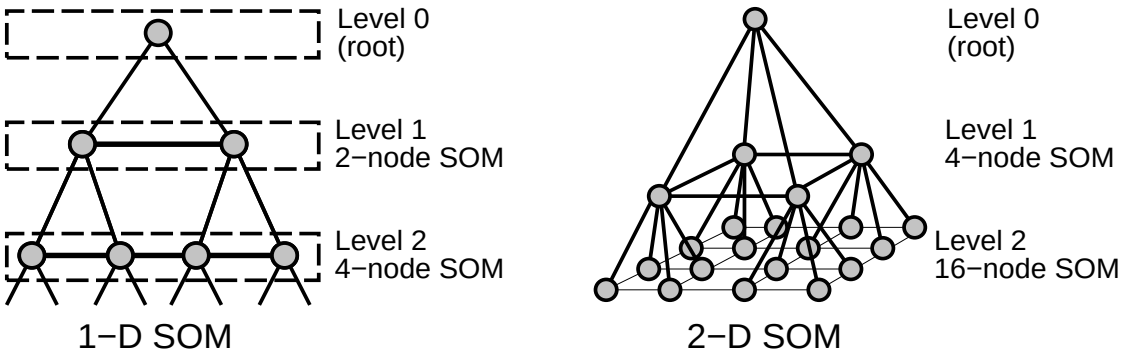
3 The tree-structured self-organizing map

The tree-structured self-organizing map (Koikkalainen 1990, Koikkalainen 1994) (TS-SOM) is, among other things, a computationally fast variation of the SOM. Methodologically it is a combination of the self-organizing map, tree-structured clustering and computational speedup techniques.

Another, sometimes even more important benefit of the TS-SOM is that unlike the basic SOM, it does not require any "ad hoc" parameter settings to get well organized. The difficult choices about the number of nodes and width of the smooter (size of neighborhood) are automized. Therefore the method suits well to data cleaning tasks, where it might be difficut or impossible to play with the parameters of the algorithm, while the data itself is a major problem.

The computational benefit of the TS-SOM over basic SOM is most notable when the number of data samples is large and the complexity of data requires many updates before convergence. If P is the number of SOM lattice nodes, then full search SOM makes P computations of the distance $d(\mathbf{x}, \mathbf{w}_k) = \|\mathbf{x} - \mathbf{w}_k\|$ to find the best node b($\mathbf{X}(j)$). This is per each presentation of a sample $\mathbf{X}(j)$. The speedup becomes significant after one thousand samples, after which the computation of distances $d(\mathbf{x}, \mathbf{w}_k)$ clearly dominates the computing time of the algorithm.

Chart 5: Data structures of the TS-SOM. Several SOMs (layers) are connected to form a tree like structure.



When organizing TS-SOM, several SOMs (layers) with different resolutions are trained, strating from simple ones and increasing the number of nodes by 2^S times when a new layer is introduced. Layers are connected such that each node is parent of exactly 2^S child nodes on the next layer, as depicted in Fig. 5. The number of nodes P_l on layer l is therefore 2^{lS}.

The tree-like structure of the TS-SOM is useful in two ways: i) it can be used as a search tree to speed

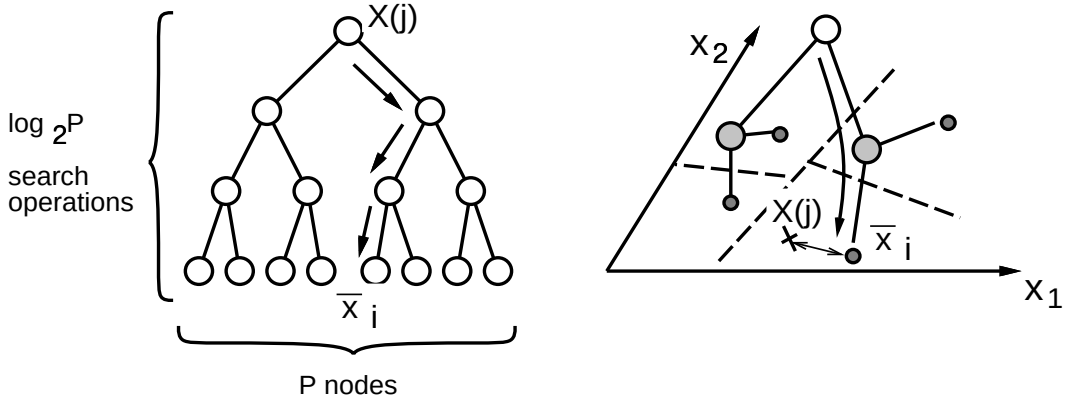
up the search of the best matching node $b(\mathbf{X}(t))$, and ii) it can be used as a constructive estimator of the data, from highly smoothed solutions (less nodes) to more complex ones.

3.1 The TS-SOM training algorithm

The actual training algorithm for the TS-SOM includes several computational tricks and algorithmic details. Normal user of the TS-SOM based software can omit these details, but anyone who needs to implement the method can not.

Most of the specific issues related to TS-SOM training are due the unique combination of kernel smoothed SOM updates and tree search. To understand this, we compare the difference of the TS-SOM to the tree-structures vector quantization algorithm TS-VQ (Buzo et al. 1982).

Chart 6: Search in the tree stuctured vector quantizer (TS-VQ) and the decision regions caused by the search.

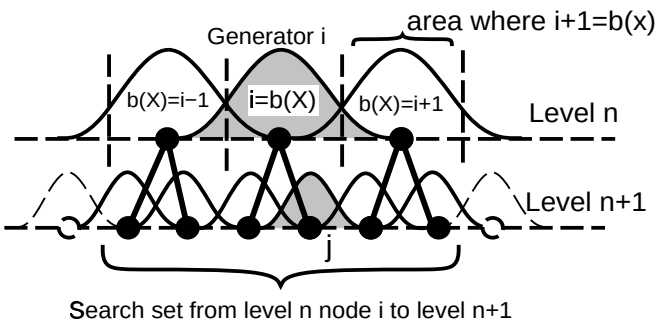


The TS-VQ is a splitting algorithm that divides data between the nodes of the same parent using nearest neighbor rule, and updates the nodes to the centroids of their data sets. In this type of algorithms, as depicted in Fig 6, each node i of the tree holds one portion of data, Ω_i, which is not optimal Voronoid shape because of the tree search. The data sets on the same tree level l are disjoint $\Omega_i^l \cap \Omega_k^l = 0$ and convex. On the next level $l+1$, the sons of the parent node i split the set Ω_i^l such that $\cup_k \Omega^{l+1}_{\mathrm{son_k(i)}} = \Omega_i^l$, and $\cap_k \Omega^{l+1}_{\mathrm{son_k(i)}} = 0$. The update of son positions to centroids $\overline{\mathbf{x}}_{\mathrm{son_k(i)}}$ and the splitting of data sets can can be done independently on the same tree level. The benefit of the tree search is quite clear, To find the est matching node, only $log_2 P$ search operations are needed. The method is also more realiable than, for example, the K-means algorithm, because of the constructive manner of tree building.

The main difference between TS-VQ and TS-SOM is that the nodes of the TS-SOM layer are not independent. This is because of kernel smoothing along the SOM lattice. As a consequence, the computation of centroids and the search of the best matching node cannot be done independently for different parent nodes. In theory this could prevent us using tree-search technigues. Fortunately, in practice all computations are localized to small neighborhood around the parent node. To understand why, we need to see figure 7 and recall that SOM nodes can be intepred as density components of a mixture model. Now, if each SOM node represents local density of data, then a significant part of data points of parent is distributed between the sons of the parent and its neighbors. The width of this disrtibution can be selected by adjusting the width of the kernel smoother.

It is essential for the understanding of the TS-SOM algorithm that **in TS-SOM the smoothing kernel width is fixed** such that the neighborhood function $h_{i,k}$ is is nonzero only for the nearest neighbors of the node (but not the node itself), as depicted in 8. In fact we may write

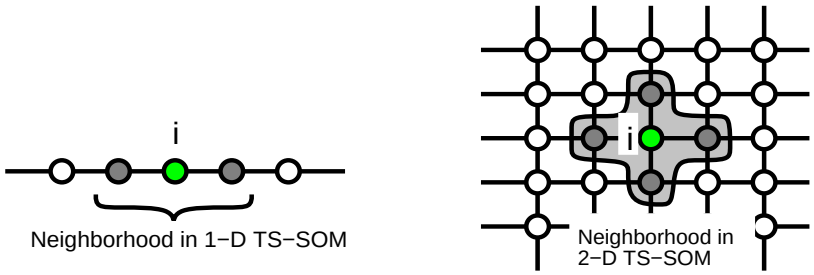
Chart 7: The relation between parent nodes and son nodes in a 1-D TS-SOM structure. The influence of parent node is distributed to the sons of parent and its neighbors.



$$h_{i,k} = \begin{cases} 1 & \text{if } k = i \\ \gamma & \text{if } ldist(k,i) = 1, \\ 0 & \text{otherwise} \end{cases} \qquad ldist(k,i) \text{ is the SOM lattice distance (number of nodes)} \ , \quad (7)$$

where $0 < \gamma < 1$ is the neighbor weighting parameter that is related to kernel smoother width. It is typically constant 0.25.

Chart 8: Neighborhood $\mathrm{Ne}(i)$ of a node in TS-SOM. It is always fixed to distance of one node around node i. Note that i is not a neighbor of itself.



The fixed neighborhood on each TS-SOM level (SOM layer) separates the TS-SOM training algorithm from both basic SOM, where the neighborhood shrinks as a function of time, and from TS-VQ, where no neighborhood exists and nodes are independent. Because of the fixed neighborhood

i) The search of the best matching node from node i on layer l to next layer nodes can be localized to a constant size **search set** of $(2^S + 1) \times 2^S$ nodes. (sons of parent and its neighbors).

ii) Update of node position $\mathbf{w}_i$ is almost surely bounded inside the search set.

iii) Layers with smaller number of nodes correspond to strongly smoothed SOMs because the width of the smoother covers large area in the data space. This is discussed in section 3.8.

3.2 Search of the best matching unit

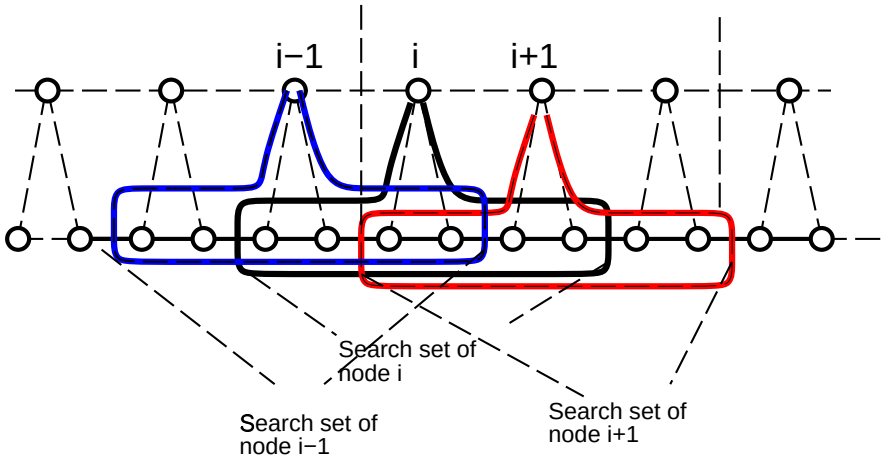
In the TS-SOM algorithm the search of the best matching node $b(\mathbf{X}(j))$ for a given sample $\mathbf{X}(\mathbf{t})$ is quite special. To understand howto implement the incomplete data version of the TS-SOM, one cannot avoid this subject, which is the main reason, why this presentation is so much involved with all kind of TS-SOM technicalities.

In TS-SOM terminology the search of b can be conditionalized to any node i on layer l. The search will then continue to next layer $l+1$ nodes. Since the search is always localized, we can define a **the search set** $\texttt{Sset}(i)$. Let $\mathrm{Ne}(i)$ be the set of nearest neighbors of node i, and $\mathrm{Sons}(k)$ the set of sons of node k. The search set is now

$$\texttt{Sset}(i) = \{ \text{ sons of i}\} \cup \{\text{sons of neighbors ofi}) = \mathrm{Sons(i)} \cup \mathrm{NS}, \mathrm{NS} = \cup_{\mathrm{k}\in\mathrm{Ne(i)}}\mathrm{Sons(k)}$$

By definition, the search sets of the neighboring neurons are overlapping as shown in Fig. 9. Therefore **search is not bounded to direct decendents of a node** like it is in the TS-VQ algorithm. If the search set were not overlapping, then lateral smoothing on the SOM lattice could move nodes out of the search regions, defined by their parents. The error between tree-search and full search would then become intolerable, and the SOM would not organize well into data. Note that because the size of the search set is constant, also the time complexity per node is constant $O(1)$.

Chart 9: Overlapping search sets to find the next layer best matching node from neighboring nodes $i-1$, i and $i+1$.



The tree search, based on search sets, starts from the root node and goes layer by layer down to the currently adaptive layer. Because the layers are trained in order, $l = 0, 1, 2, \ldots$, from root to more complex levels, and fozen after training, the role of the upper layers is only to be a search structure for the currently adaptive layer.

3.3 The lookup search technique

The TS-SOM training, like any iterative algorithm, presents the same sample several times for the network. Since the upper layers of the TS-SOM are frozen, the search will go exactly via same nodes. This is clearly a computational bias that can be eliminated.

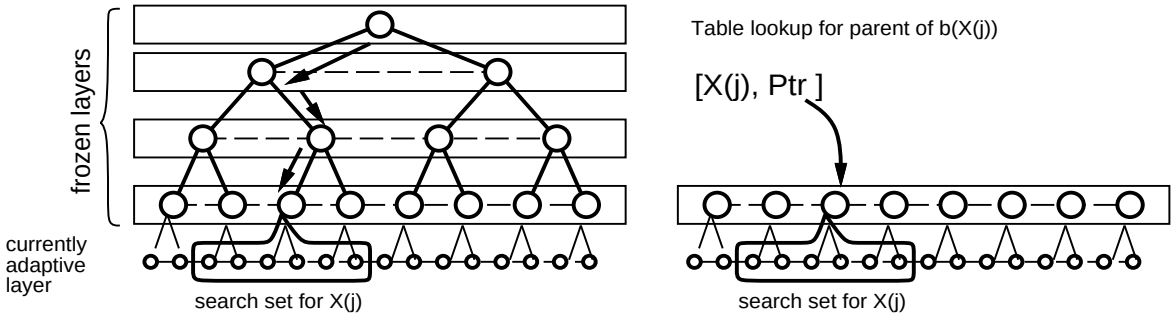
This simple technique is based on a table lookup. The idea is to add one new variable for each data vector, a pointer to the parent node of the currently adaptive neurons. This node defines without any loss the same search set as what the tree search does (see Fig. 10).

After a layer is trained (it is frozen) the indexes of the best matching nodes are stored in the lookup variables

$$L^l(j) = b^l(\mathbf{X}(j)).$$

When training a new layer, $l+1$, the search of the best matching node is for sample $\mathbf{X}(j)$ is then

Chart 10: In the lookup search, the search through first TS-SOM layers is replaced with a lookup table.



$$b^{l+1}(\mathbf{X}(j)) = \arg \min_{k \in \texttt{Sset}(\mathrm{L}^l(j))} \|\mathbf{X}(j) - \mathbf{w}_k\|.$$

Since the size of `Sset` is always constant $(2^S + 1) \times 2^S$, significant time saving is achieved. If the samples are repeated many times, the complexity of the search with respect to the number of neurons is now close to constant $O(1)$ instead of $O(log_{2^S})$ of the tree-search. At first this might seem magical, but if one thinks that the complexity of the problem is defined by the data, not bye the number of neurons, then additional neurons, after the data set is sufficiently well represented, should not increase the complexity of the training.

3.4 Node updating

After the best matching nodes for all samples are found, then all nodes are updated. Let, as usual

$$\overline{\mathbf{x}}_k = \frac{1}{N_i} \sum_{j \in \Omega_i} \mathbf{X}(j)$$

be the conditional sample mean of node i. Because of the assumption that nodes are gaussian generators, as well as due the smoothing, the sample is distributed over all nodes in the complex (neighborhood plus the node itself). This can be intepred as a mixing probability, which is defined by the smoothing parmeter γ (see equation 7.)

The updating is therefore based on equation **??**, as usual.

$$\mathbf{w}_i = \frac{1}{\sum_{k \in \mathrm{CNe}(i)} h_{i,k} N_k} \sum_k h_{i,k} N_k \overline{\mathbf{x}}_{k \in \mathrm{CNe}(i)}, \quad \text{where} \quad \mathrm{CNe}(i) = \{i\} \cup \mathrm{Ne}(i).$$

The neighborhood size and weighting constant are fixed. Therefore on each TS-SOM layer the amount of regularization (smoothing width) is fixed during the training. This is notably difference to traditional "ad hoc" training of the SOM, where the objective of the optimization of weights is chancing during the learning becase of decreasing neighborhood.

3.5 Initialization of nodes for new layer

In most iterative algorithms the computing time can be decreased significantly by good initialization of parameters. While random selection of weight walues might work with basic SOM, more efficient methods can be derived for the TS-SOM. This is because of the previously adaptde layers have already captured come characteristics of data.

In most implementations we use a simple approach that initializes the weight vectors $\mathbf{w}_k$ of son nodes from their parents after the parent layer l is frozen.

$$\texttt{Initially}: \forall_{k \in \mathrm{Sons}(i)} \mathbf{w}_k^{l+1}(0) = \mathbf{w}_i^l$$

Another, but computationally more costly initialization is to interpolate the new layer nodes from the parents. We have tested FFT interpolation for this, which gives good results, but looses in practical applications because of increased computing time.

3.6 Summary of the algorithm

The TS-SOM training can now be summarized as given in Algorithm 2. Training starts from the layer 0, which has only one node. Then layers are trained in order.

After training all layer the nodes are frozen, and the lookup pointers from samples to their best mathing nodes are updated. The weights of the new layer nodes are initialized from the parent nodes of the previouly fozen layer layer.

The training rule itself is similar to previously introduced SOM batch training, execpt that the neighborhood size is fixed only to the nearest neighbors of the node.

Algorithm 2: TS-SOM training.

0. Initially : $l = 0$ (layer is root);
$\mathbf{w}_{root} = \overline{\mathbf{x}} = \frac{1}{N_{\mathrm{root}}} \sum_j \mathbf{X}(j)$

1. Initialize new layer :

*Step*1.1 *Initialize weights from parent nodes* i :
$\forall_{k \in \mathrm{Sons}(i)} \mathbf{w}_k^{l+1}(0) = \mathbf{w}_i^l$;

*Step*1.2 Initialize lookups for all samples j,
$L(j) = b^l(\mathbf{X}(j))$

*Step*1.3 (update layer index)
$l := l + 1$;

2. Train layer : l

Step 2.1 (*Use lookup search*)

$\Omega_i = \{ j \mid \mathrm{b}^l(\ \mathbf{X}(j)\ ) = i \}$;

Step 2.2 (*Compute node means*)

$\overline{\mathbf{x}}_i = \frac{1}{N_i} \sum_j \mathbf{X}(j)$, `where` $j \in \Omega_i$; ;

Step 2.3 (*Compute node positions*)

$$\mathbf{w}_i = \frac{1}{\sum_{k \in \mathrm{CNe}(i)} h_{i,k} N_k} \sum_k h_{i,k} N_k \overline{\mathbf{x}}_{k \in \mathrm{CNe}(i)},$$
where $\mathrm{CNe}(i) = \{i\} \cup \mathrm{Ne}(i)$.

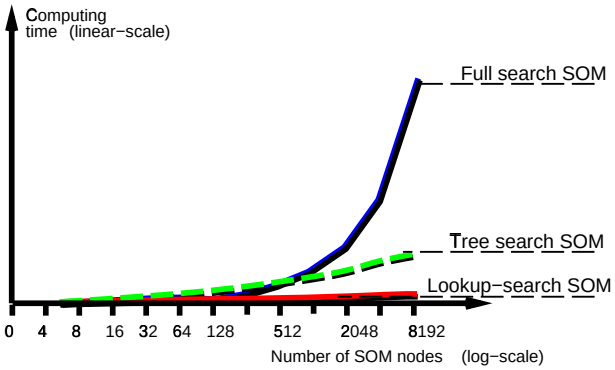
4. Repeat until converged :
If $\|\mathbf{W}^{new} - \mathbf{W}^{old}\| > \delta$ GOTO 2;

5. Next layer. If more layers GOTO 1;

3.7 Time complexity of the TS-SOM

When using TS-SOM, the time complexity of the search of $b(\mathbf{X}(t))$ is reduced from $O(P)$ to range between $O(1)$ and $O(\log_{2S} P)$. Especially for complex problems the computing time is close constant, $O(1)$, with respect to the number of neurons P. This is due a special **lookup search technique** that requires only a fixed subset of nodes to be searched per sample, when the same sample is presented several times, as it is done in all iterative algorithms.

Chart 11: Comparison of computing times when using full search, tree search and lookup search to train self-organizing maps.

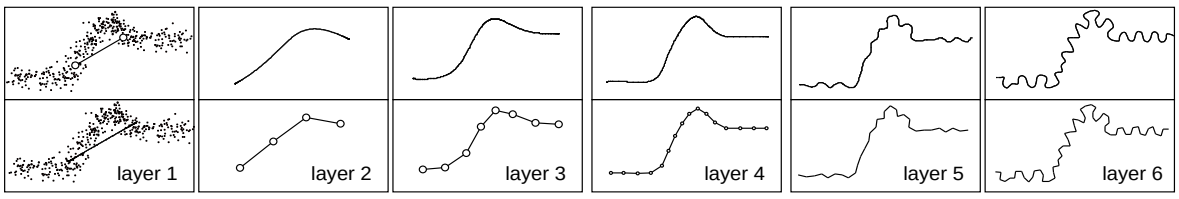


This is proven by example runs, as shown in Fig. 11, where several 1-D SOM's with different number of nodes are trained using (i) normal full search, (ii) tree search and (iii) lookup search techniques. The data dimension or the number of samples does not significantly change the relative behavior of the curves.

3.8 Algorithmic complexity of data and the TS-SOM layer

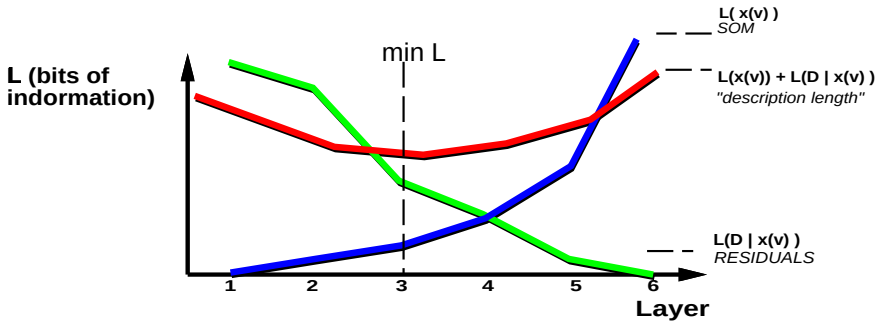
Empirical results of the TS-SOM algorithm are visualized in Fig. 12. The bottom and top images are same 1-D SOMs for given 2-D data, with an increasing number of nodes. On the top the smoothness of the curve is visualized by adding more nodes by Fourier interpolation (zeros added to spectrum). This operation does not add any information because the Fourier spectrums of top and bottom images are exactly same. Therefore one can conclude that TS-SOM layers with small number of nodes correspond to large width of the kernel smoother when compared to basic SOM algorithm. This means also, that by selecting the appropriate TS-SOM layer, we can select the complexity of SOM.

Chart 12: Bottom: several TS-SOM layers (SOMs with an incresing number of nodes). Top: Same as bottom, but missing nodes are added using FFT interpolation.



To find the best representation of data, the discription lenghts of SOM layers $L(\mathbf{x}^l(v))$ and residuals $L(\epsilon|\mathbf{x}^l(v))$, where $\epsilon = \mathbf{x}(\mathbf{v}'(\mathbf{X})) - \mathbf{X}$, are computed. The result are shown in Fig. 13 that suggests that the best model of the given data is SOM layer 3, with eight gaussian generators.

Chart 13: Description lenghts of SOM layers and residuals.



4 Training TS-SOM with incomplete data

The TS-SOM algorithm, as introduced in the previous section, is designed for complete data vectors without any missingness on variables or data records. In this section we extend the algorithm such that it is able to use all information of incomplete data sets.

When there are missing samples we need external information, **sampling weights**, $sw(j)$, that associate to each sample j information about its sampling bias. If we know that the `iid` assumption is violated, then $sw(j)$ is a measure of how many `iid` samples corresponds to each of our observed sample j. This can take be any positive real value. During the TS-SOM training sampling weights are added to the computation of node priors.

When there are **missing variables**, then incomplete likelihood model for SOM can be used. This itself requires very little computational effort since the SOM algorithm is already based on a variation of the EM-algorithm. Only difference to previous training is that missing values are ignored when computing sample means. Mathematically this is same as mean value imputation of missingness during the training, but no imputations are needed in practice.

Unfortunately, missingness prevents us from computing the best matching node b, which should be the most probable node for a given completely observed input $\mathbf{X}(j)$. Yet the SOM algorithm is using the best matching node only because of its computational efficiency. It is simpler to classify data between the nodes than compute actual posterior probabilities Pr(i | $\mathbf{X}$(j)) of nodes for given samples. If some variable values are missing, this approach cannot be used.

The computation of posteriors for all nodes would be very costly, especially when compared to lookup search of the TS-SOM algorithm. Therefore we decided to compute `Nb` best matching (most probable) nodes instead. From the computer implementation point of view this requires some effort, but the computational cost is only about `Nb` times more than that of normal TS-SOM algorithm. Another good news is that when `Nb` is "sufficiently" large, the difference between true mixture model SOM and the usual (most probable node) SOM vanishes.

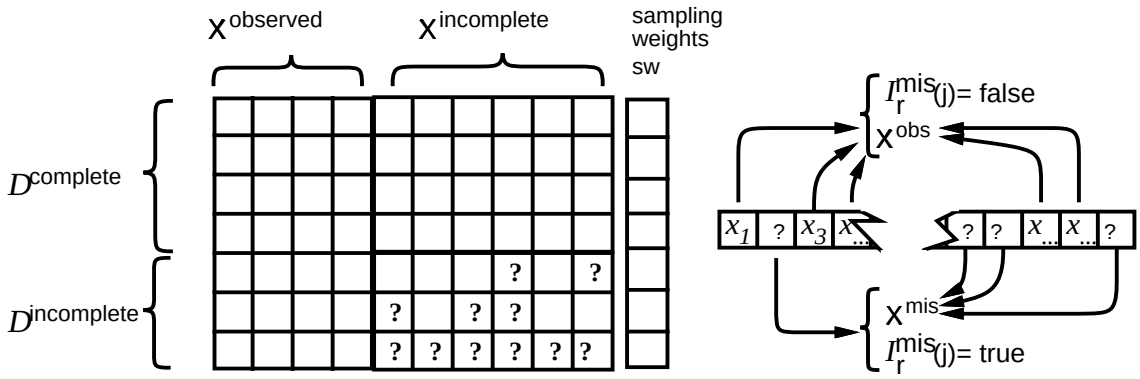
The computer implementation of our new TS-SOM is a variant of dynamic multi path programming, which are also know as fast marching methods. The lookup search can be used as before, but now `Nb` new pointer variables need to be introduced per each sample. For very large data sets this might cause problems with memory consumption. However, the algorithm seems to perform well with only a couple of "best" nodes.

4.1 Notations about missingess.

To explain the incomplete data version of the TS-SOM we indicate the missingness as shown in Fig. 14. This is quite trivial for anyone who has been working with incomplete data, but for a computer

programmer, the data model must be specifed carefully to avoid unnesessary memory consumption.

Chart 14: Notations, used to indicate missing values.



Elements of data matrix are either observed $d_r^{\text{obs}}(j)$ or missing $d_r^{\text{mis}}(j)$. The matrix D can therefore be factorized between two sets of elements such that

$$\mathcal{D} = \mathcal{D}^{\text{obs}} \cup \mathcal{D}^{\text{mis}},$$

where values of $\mathcal{D}^{\text{obs}}$ are normal variables and $\mathcal{D}^{\text{mis}}$ consists of special markers `mval`, values that are only used to indicate missingness.

Data records (samples j) are divided between complete and incomplete data vectors such that

$$\mathbf{X}(j) \in \begin{cases} D^{\text{incomplete}} & \text{if } \exists_r I_r^{\text{mis}}(j) = true \\ D^{\text{complete}} & \text{if } \forall_r I_r^{\text{mis}}(j) = false \end{cases}$$

where $I_r^{\text{mis}}(j)$ is an indicator function for missingess of variable r in sample j. Formally

$$I_r^{\text{mis}}(j) = \begin{cases} \texttt{true} & \text{if } \mathbf{X}_r(j) = \texttt{mval} \\ \texttt{false} & \text{if } \mathbf{X}_r(j) \neq \texttt{mval}. \end{cases}$$

In computer only data matrix D with indicators `mval` in place of missing variables need to be allocated. Variable indicators $I_r^{\text{mis}}(j)$ can be computed when needed. In the qualitative and quantative evaluation of the results we use sample indicators $\mathbf{X}(j)^{\text{imp}} \in D^{\text{incomplete}}$ and $\mathbf{X}(j)^{\text{obs}} \in D^{\text{complete}}$ to note differences between observed and imputed samples.

4.2 Using sampling weights in TS-SOM training

The role of sampling weights is to easily noted by changing the computation of sample means. We use weighted cardinalities N_k^{sw} instead of normal cardinalities N_k.

This technique is also in essential role when the multipath version of incomplete data TS-SOM training is used, regardless wether actual sampling weights are used or not. Therefore, instead of using normal sampling weights $sw(j)$, corrected sampling weights $\hat{sw}$ are used in the following equations. When there is no variable level missingness $\hat{sw} = sw$.

The definition of weighted cardinalities is

$$N_k^{\text{sw}} = \sum_{j \in \Omega_k} \hat{sw}(j). \tag{8}$$

Clearly $N_k^{\mathrm{sw}} = N_k$ if $sw(j) = 1$ for all samples j. Now the node update can be written as

$$\mathbf{w}_i = \frac{1}{\sum_k h_{i,k} N_k^{\mathrm{sw}}} \sum_k h_{i,k} N_k^{\mathrm{sw}} \overline{\mathbf{x}}_k, \tag{9}$$

and the weighted sample mean for node k is

$$\overline{\mathbf{x}}_k = \frac{1}{N_k^{\mathrm{sw}}} \sum_{j \in \Omega_k} \hat{sw}(j) \mathbf{X}(j). \tag{10}$$

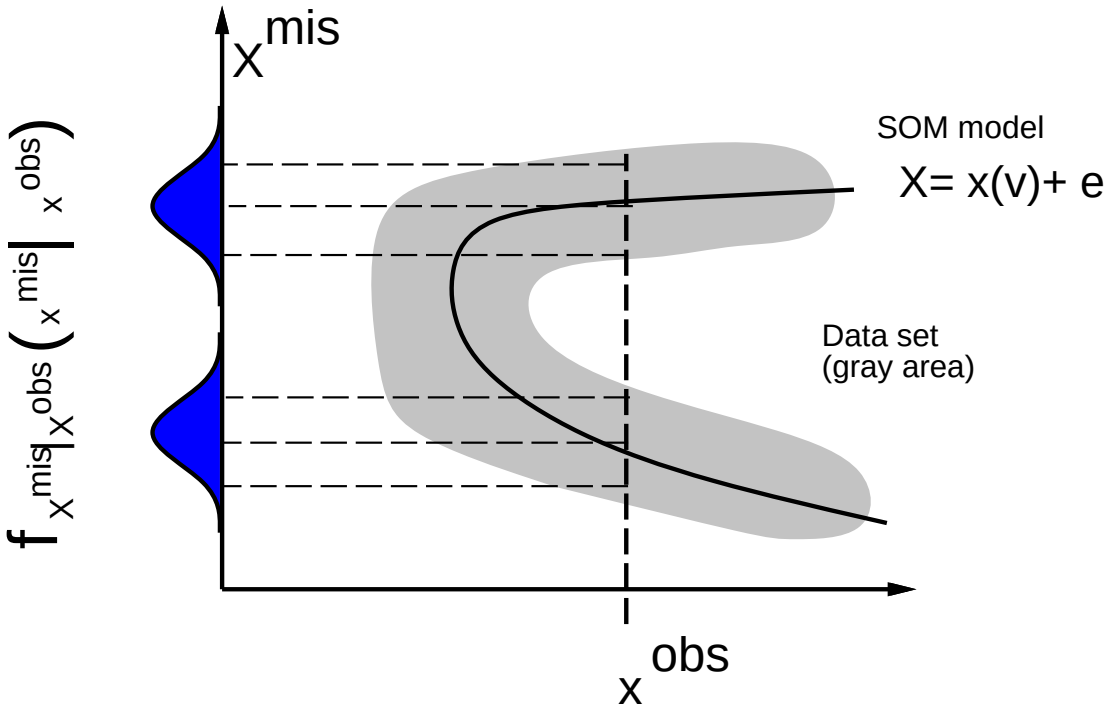
4.3 Incomplete data likelihood for TS-SOM

The formulation of incomplete data version of the TS-SOM is a simple application of the EM-algorithm, which is already used in the complete data version of the TS-SOM. We can always factorize the expected log-likelihood, the Q function, to observed and missing parts

$$\ln Q(\mathbf{W}^{t+1}|\mathbf{W}^t) = \mathbb{E}[\ln \mathcal{L}(\mathbf{W}^{t+1}; D^{\mathrm{mis}})|D^{\mathrm{obs}}, \mathbf{W}^t] + \mathbb{E}[\ln \mathcal{L}(\mathbf{W}; D^{\mathrm{obs}})|\mathbf{W}^t].$$

where maximixation of $\mathbb{E}[\ln \mathcal{L}(\mathbf{W}; D^{\mathrm{obs}})|\mathbf{W}^t]$ is computed as before and $\mathbb{E}[\ln \mathcal{L}(\mathbf{W}^{t+1}; D^{\mathrm{mis}})|D^{\mathrm{obs}}, \mathbf{W}^t]$ is maximized using the SOM (principal surface). This means that missing values are taken from the mariginalized distribution $f_{\mathbf{X}^{\mathrm{mis}}|\mathbf{X}^{\mathrm{obs}},\mathbf{W}^t}(\mathbf{x}^{\mathrm{mis}}|\mathbf{X}^{\mathrm{obs}}, \mathbf{W}^t)$ as depicted in Fig. 15.

Chart 15: Marginal distribution of missing values for a given observation and principal curve.



The marginal distribution is our best knowlegde about the distribution of missing values, given what is observed. We shall see that is leads to simple and computationally implementation, when TS-SOM is used as a model.

4.4 Selection of the best matching units

The procedure requires, potentially, several best SOM nodes for each partial observation $\mathbf{X}^{\text{obs}}(j)$. For computational reasons the number of best nodes `Nb` is decided in advance, allowing the selection of `Nb` best nodes to be defined as a search

$$b_N(\mathbf{X}^{obs}) = \{k_1, k_2, \ldots, k_{\texttt{Nb}} \mid d_{k_1}(\mathbf{X}^{obs}) < d_{k_2}(\mathbf{X}^{obs}) < \ldots < d_{k_{\texttt{Nb}}}(\mathbf{X}^{obs})\},$$

where $d_{k'} = \|\mathbf{X}^{\text{obs}} - \mathbf{x}^{\text{obs}}(\mathbf{v}_{k'})\|$, and $\mathbf{X}^{\text{obs}}$ is the observed part of the sample vector.

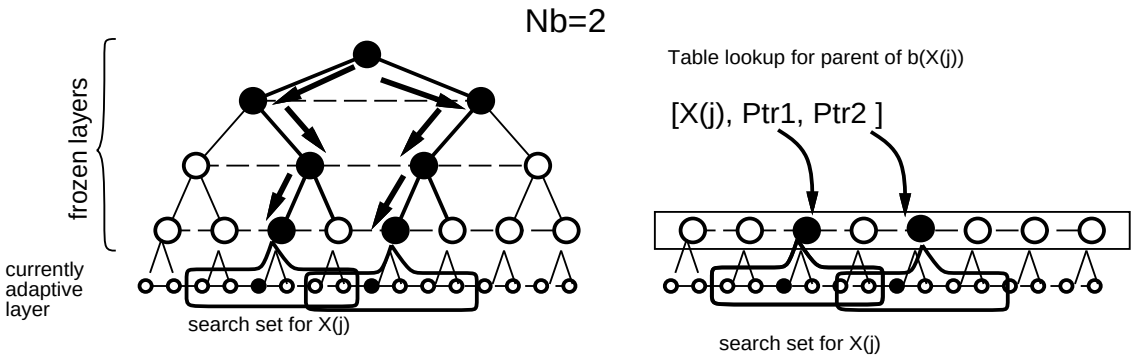
At first it seems that this cannot be implemented easily using TS-SOM speedup techniques. In practice, however, the only difference to basic TS-SOM is that instead on one lookup variable, a vector of `Nb` lookup pointers is used (see 16). Now, after a layer is trained, (and frozen), the lookups for all samples j are updated to

$$\mathbf{L}^l_N(j) = \{L^l_1(j), L^l_2(j), \ldots, L^l_{\texttt{Nb}}(j)\} = b^l_N(\mathbf{X}(j)^{obs})$$

Then, when training a new layer $l+1$, the best matching units are computed such that search is limited to a combined search set of nodes

$$k' \in \cup_{k''} \texttt{Sset}(L^l_{k''}(j))$$

Chart 16: Tree search and lookup search visualized when multiple search pathts are used.



The combined search set size is bounded by constant $(2^S+1)\times 2^S \times \texttt{Nb}$. Because of this, the computing time of the search is maximally `Nb` longer than that of the basic TS-SOM algorithm. In reality the time is shorter because the search sets of different paths are often overlapping.

It should be noted that multipath search is needed only for samples $j \in D^{\text{incomplete}}$. For complete data vectors normal, one path, search is used. Thus, the computational cost of multipath search is in average $\frac{(2^S+1)\times 2^S}{N} \times (N^{\text{complete}} + N^{\text{incomplete}} \times \texttt{Nb})$.

4.5 Updating with incomplete observations.

When multipath serach is used with imcomplete sample vectors, several "best" nodes are obtained. This multiplies the weight of incomplete samples, which must be corrected with sampling weights $\hat{sw}(j)$ such that

$$\hat{sw}(j) = \frac{1}{\texttt{Nb}} sw(j), \quad \text{if } j \in D^{\text{incomplete}}$$

where $sw(j) = 1$, if there are no sampling weights.

The idea could be futher generalized by computing posterior probabilities $\Pr(k|\mathbf{X}^{\text{obs}})$ for each node k in the set $b_N(\mathbf{X}^{\text{obs}})$. The the sampling weights would be specific to both nodes and samples

$$\hat{sw}_k(j) = \Pr(k|\mathbf{X}^{\text{obs}})sw(j) \quad \text{if } j \in D^{\text{incomplete}}.$$

The threatment of missingness is easiest to understant on variable level. For a moment, we assume that the value of variable r is either know or imputed from the marginal distribution

$$X_r(j) = \begin{cases} X_r^{\text{obs}}(j) & \text{if } I_i^{\text{mis}} = \texttt{false} \\ X_r^{\text{imp}}(j) & \text{if } I_i^{\text{mis}} = \texttt{true}, \end{cases} \quad \text{where } X_r^{\text{imp}}(j) \sim f_{\mathbf{X}^{\text{mis}}|\mathbf{X}^{\text{obs}},\mathbf{W}^t}(\mathbf{x}^{\text{mis}}|\mathbf{X}^{\text{obs}},\mathbf{W}^t)$$

The imputation is made by first selecting the SOM node i from set $b_N(\mathbf{X}^{obs})$, as discussed earlier, and then using the SOM model and randomness around it to get the actual value

$$X_r^{\text{imp}} = w_{i,r}^t + \epsilon, \quad \text{where} \quad i \in b_N(\mathbf{X}^{obs})$$

We first note that the number of samples per node is divided between observed and missing parts $N_{i,r}^{\hat{sw}} = N_{i,r}^{\hat{sw},\text{mis}} + N_{i,r}^{\hat{sw},\text{obs}}$, where $\hat{sw}$ denotes that the corrected sampling weights are used

The SOM updating rule for variable r and node i can be written as

$$w_{i,r}^{t+1} = \sum_k h_{i,k}\mathbb{E}[X_r|k \in b_N(\mathbf{X}^{obs})] = \frac{1}{\sum_k h_{i,k}(N_{k,r}^{\hat{sw},\text{obs}} + N_{k,r}^{\hat{sw},\text{mis}})} \sum_k h_{i,k}(N_{k,r}^{\hat{sw},\text{obs}}\overline{x}_{k,r}^{\text{obs}} + N_{k,r}^{\hat{sw},\text{mis}}\overline{x}_{k,r}^{\text{mis}}),$$

where

$$\overline{x}_{k,r}^{\text{obs}} = \frac{1}{N_{k,r}^{\hat{sw},\text{obs}}} \sum_{j\in\Omega_k, I_r^{\text{mis}}(j)=\texttt{false}} \hat{sw}(j)X_r^{\text{obs}}(j)$$

$$\overline{x}_{k,r}^{\text{mis}} = \frac{1}{N_{k,r}^{\hat{sw},\text{mis}}} \sum_{j\in\Omega_k, I_r^{\text{mis}}(j)=\texttt{true}} \hat{sw}(j)X_r^{\text{imp}}(j) = \frac{1}{N_{k,r}^{\hat{sw},\text{mis}}} \sum_{j\in\Omega_k, I_r^{\text{mis}}(j)=\texttt{true}} \hat{sw}(j)(w_{i,r}^t + \epsilon)$$

$$= \frac{\sum_{j\in\Omega_k, I_r^{\text{mis}}(j)=\texttt{true}} \hat{sw}(j)}{N_{k,r}^{\hat{sw},\text{mis}}} w_{i,r}^t = w_{i,r}^t,$$

where $\overline{x}_k^{\text{mis}} = w_{i,r}^t$ because is the average over zero mean noise ϵ is zero. Therefore **no actual imputations are needed**, we simply use old weight value $w_{i,r}^t$ as a mean of all missing values.

4.6 Summary of the incomplete TS-SOM training algorithm

The TS-SOM training with incomplete data vectors is summarized in Algorithm 3. The main differences to basic TS-SOM are the use of multiple paths and the incomplete data update rule for node centroids.

Algorithm 3: TS-SOM training for incomplete data and sampling weights.

0. Initially : $l = 0$ (layer is root);

Step 0.1 (*corrct sampling weights*)

$$\forall_j : \hat{sw}(j) = \begin{cases} sw(j) & \text{if } j \in D^{\text{complete}} \\ \frac{1}{\texttt{Nb}} sw(j) & \text{if } j \in D^{\text{incomplete}} \end{cases}$$

Step 0.2 (*compute root node position*)

$$\mathbf{w}_{root} = \overline{\mathbf{x}} = \frac{1}{N^{sw}_{\text{root}}} \sum_j sw(j)\mathbf{X}(j)$$

1. Initialize new layer :

Step 1.1 *Initialize weights from parent nodes* i :
$\forall_{k \in \text{Sons}(i)} \mathbf{w}_k^{l+1}(0) = \mathbf{w}_i^l$;

Step 1.2 Initialize lookup vectors for all samples j,
$\mathbf{L}_N(j) = b_N^l(\mathbf{X}(j))$

Step 1.3 (update layer index)
$l := l + 1$;

2. Train layer : l

Step 2.1 *Use lookup search for every node* i

$$\Omega_i = \{j \mid i \in \text{b}_N^l(\ \mathbf{X}^{\text{obs}}(j)\ )\ \};$$

Step 2.2 *For every node* i

Step 2.2.1 (*Compute node priors for every variable* r)

$$N_{i,r}^{sw,\text{obs}} = \sum_{j \in \texttt{Robs}} \hat{sw}(j), \quad \text{where } \texttt{Robs} = \{j \mid I_r^{\text{mis}}(j) = \texttt{false}, j \in \Omega_i\}$$

$$N_{i,r}^{sw,\text{mis}} = \sum_{j \in \texttt{Rmis}} \hat{sw}(j), \quad \text{where } \texttt{Rmis} = \{j \mid I_r^{\text{mis}}(j) = \texttt{true}, j \in \Omega_i\}$$

Step 2.2.2 (*Compute node means for every variable* r)

$$\overline{x}_{k,r}^{\text{obs}} = \frac{1}{N_{k,r}^{\hat{sw},\text{obs}}} \sum_{j \in \Omega_k, I_r^{\text{mis}}(j) = \texttt{false}} \hat{sw}(j) X_r^{\text{obs}}(j)$$

$$\overline{x}_{k,r}^{\text{mis}} = w_{i,r}^t,$$

Step 2.3 *Compute node positions for every variable* r

$$w_{i,r}^{t+1} = \frac{1}{\sum_k h_{i,k}(N_{k,r}^{\hat{sw},\text{obs}} + N_{k,r}^{\hat{sw},\text{mis}})} \sum_k h_{i,k}(N_{k,r}^{\hat{sw},\text{obs}} \overline{x}_{k,r}^{\text{obs}} + N_{k,r}^{\hat{sw},\text{mis}} \overline{x}_{k,r}^{\text{mis}}),$$

4. Repeat layer training until converged :
If $\|\mathbf{W}^{new} - \mathbf{W}^{old}\| > \delta$ GOTO 2;

5. Next layer. If more layers GOTO 1;

Algorithm 3 is an direct implementation of the EM-algorithm for tree structured self-organizing maps. The construction of expected likelihood, E-step, consists steps 2.1, 2.2 and partially step 2.2.2. The M-step is then implemented by the observed part of 2.2.2 and step 2.3.

From the updating rule, step 2.3, it is possible to conclude that if there are no missing variables, then the algorithm is same as the basic TS-SOM training. Also, when the number of missing values is small, observed variables dominate the computing of node positions, and there is no notable difference to basic TS-SOM. Therefor one could just ignore all incomplete samples and use normal TS-SOM for model building.

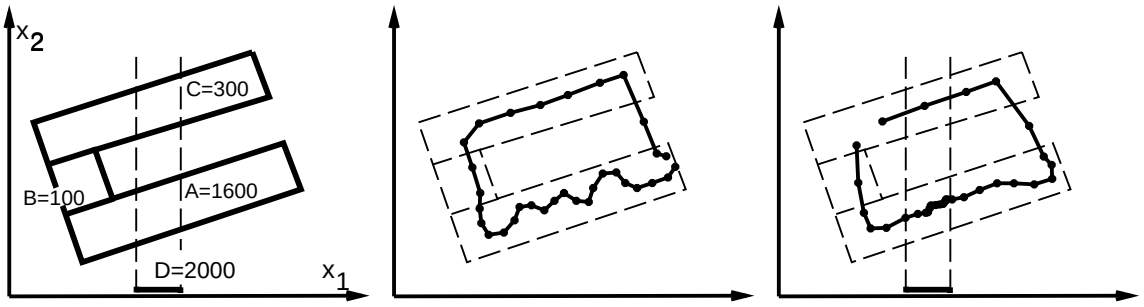
In some practical cases, however, the number of completetely observed samples is small. Especially, when working with large dimensional incomplete data sets where missingness can occur anywhere, it is less probable to find a lot of fully complete observations. In the extreme case, no completely observed sample vectors exist, and the only possibility for model building is to use incomplete data trainig alforithms.

For Euredit project, incomplete trainig has also another role, it is used as a supporting technology for error detection. The idea is to add robustness to training by ignoring all variable values that are potential errors. This is done by marking as missing those variable values that are out of our (small) confidence value. Then, after training, we can investigate the samples with respect to our model and mark as errors the values that are explained by the model by only a very low probability. In this procedure we are usually quite conservative during the training, and a lot of samples will be moved to incomplete category. Therefore, incomplete training must be used.

4.7 Toy example with incomplete data

We demonstrate the training algorithm with a simple toy example, as shown in Fig. 17. There are a total of 2000 completely observed samples, and the same number of incomplete ones. Therefore we may expect that incomplete training differs clearly from the normal training rule.

Chart 17: An example of incomplete training algorithm.



The setup of the experiment is visualized in the left part of figure. Complete observations are sampled unifromly from three, 2-D areas, A, B, and C, where the sample set sizes in are $N_A = 1600, N_B = 100$ and $N_C = 300$. For incomplete data, $N_D = 2000$ uniform 1-D samples from range D on x_1 axis are observed.

In the middle figure a 1-D SOM (layer 6) is shown after normal TS-SOM training, where the incomplete samples are ignored, while the figure on right is trained using incomplete data TS-SOM algorithm.

The key observation is that more nodes are associated to area where the missing nodes are observed. Since the distribution of fully observed samples suggests that area A is more dense than area B, also SOM nodes are mainly in this area. Second note is that the SOM curve is folding less in the imputed area. This is due the smaller variance as well as the small number of nodes in the networks.

5 Using TS-SOM for the imputation of incomplete data

In principle, the TS-SOM model for imputation can be build without incomplete training, assuming that enough of completely observed samples are observed. Yet, the incomplete data TS-SOM methodology, or an understanding about it, is essential for the imputation of incomplete data.

5.1 Bayesian intepretation

Recall our definition of the SOM as a regression surface. We can use the following Baysian intepretation for it

$$\mathbf{X} = \mathbf{x}(\mathbf{v}) + \epsilon \sim f_{\mathbf{X}|\mathbf{V}}(\mathbf{x}\ |\mathbf{v}).$$

What is important, is that in the Bayesian formulation the noise term ϵ is always a part of the model. For example, we could assume a symmetric Gaussian noise with variance β

$$f_{\mathbf{X}|\mathbf{V}}(\mathbf{x}\ |\mathbf{v}) = \frac{1}{(2\pi\beta)^{M/2}} \exp\left\{-\frac{1}{\beta}\|\mathbf{x}(\mathbf{v}) - \mathbf{x}\|^2\right\}.$$

Continuing our Bayesian argumentation, a partially observed sample $\mathbf{x}^{\mathrm{obs}} = \mathbf{X}^{\mathrm{obs}}(j)$ conditinalizes a probability distribution of its projection onto the principal surface as

$$f_{\mathbf{V}}(\mathbf{v}|\mathbf{x}^{\mathrm{obs}}).$$

Now, when using SOM, the distribution of complete observation given partially observed variables is

$$f_{\mathbf{X}}(\mathbf{x}|\mathbf{x}^{\mathrm{obs}}) = \int f_{\mathbf{X}|\mathbf{V}}(\mathbf{x}\ |\mathbf{v}) f_{\mathbf{V}|\ \mathbf{x}^{\mathrm{obs}}}(\mathbf{v}|\mathbf{x}^{\mathrm{obs}}) d\mathbf{v}. \tag{11}$$

This model contains all information about the true values, under the assumption that the model and noise formulation are realistic. In the absense of true knowledge about the two, we must use other ciriteria, like the simplest ("best") model that data allows. This leads us to model complexity selection as discussed in section 1.

In practice equation (11) can be used in several ways to get actual values for missing variables. Some of the most obvious possibilities are

i) Monte Carlo simultation, where we draw random samples from (11).

ii) Compute the expected value from (11)

iii) Compute the most probable value (mode) from (11)

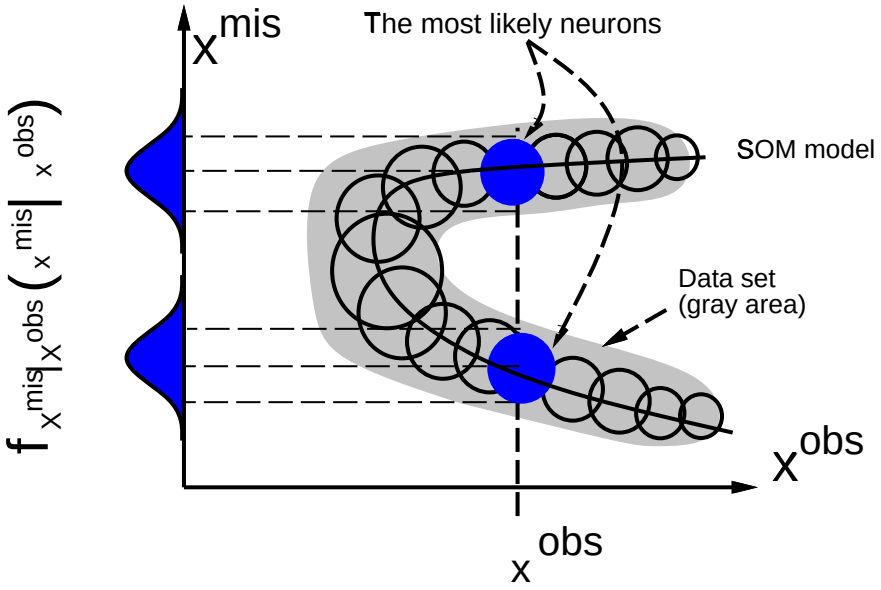
iv) Use (11) to associate probabilities for existing samples, and then use these probabilities to select a donator sample.

The implementation of the Bayesian principal surface and the related imputation techniques are done using a generative self-organizing map, as explained in the following section.

5.2 Generative SOM model for imputation

In section 2 the SOM was explained as a mixture model, although that information was not really needed in the derivation of algorithms. Where SOM is an implementation of principal curves, the generative SOM is an implementation of our previous Bayesian formulation. It allows us to do imputation as shown in Fig 18. The methodology is partially same as in the incomplete TS-SOM training, execpt that instead of model updating we need to impute the actual missing values.

Chart 18: Using generative SOM as an imputation model



In our current model the nodes are assumed to be multivariate Gaussian densities, where for simplicity the covariance matrix is diagonal (variables are linearly independent). During the model building the variances $\beta_{i,r}$ for all nodes and variables are estimated from the training data. Then, for each node and variable we have approximation of (11) as

$$f_{X_r}(x_r|i) = \frac{1}{\sqrt{2\pi\beta_i}} \exp\{-\frac{1}{\beta_r}(w_{i,r} - x)^2\}$$

and a prior probability $\Pr(i) = \frac{N_i}{N}$.

First part of the imputation procedure is same as before, as set $b_N(\mathbf{X^{obs}}(j))$ of `Nb` best matching (most probable) SOM nodes are searched, as explained in section 4.4. This corresponds to computation of $f_{\mathbf{V}}(\mathbf{v}|\mathbf{x}^{\text{obs}})$ in our Bayesian formulation.

Then a node k is selected according the prior probabilities $\Pr(i)$ from set $k \in b_N(\mathbf{X^{obs}}(j))$. Finally the values for all missing variables X_r^{mis} are selected from the component density

$$X_r^{\text{imp}} \sim f_{X_r}(x_r|k).$$

Since the Gaussian approximation extends to infinity, which is unrealistic, the values larger than $a \times \sigma$ are rejected, where a is a user given parameter and $\sigma_{k,r} = \sqrt{\beta}_{k,r}$ is the standard deviation.

5.3 Toy example revisited

We continue the previous toy example, that was first discussed in section 4.7. As before incomplete data training with TS-SOM is used. As shown in Fig. 19 three example sets were select using uniformly sampling (bottom figues) with different priors from regions A, B and C.

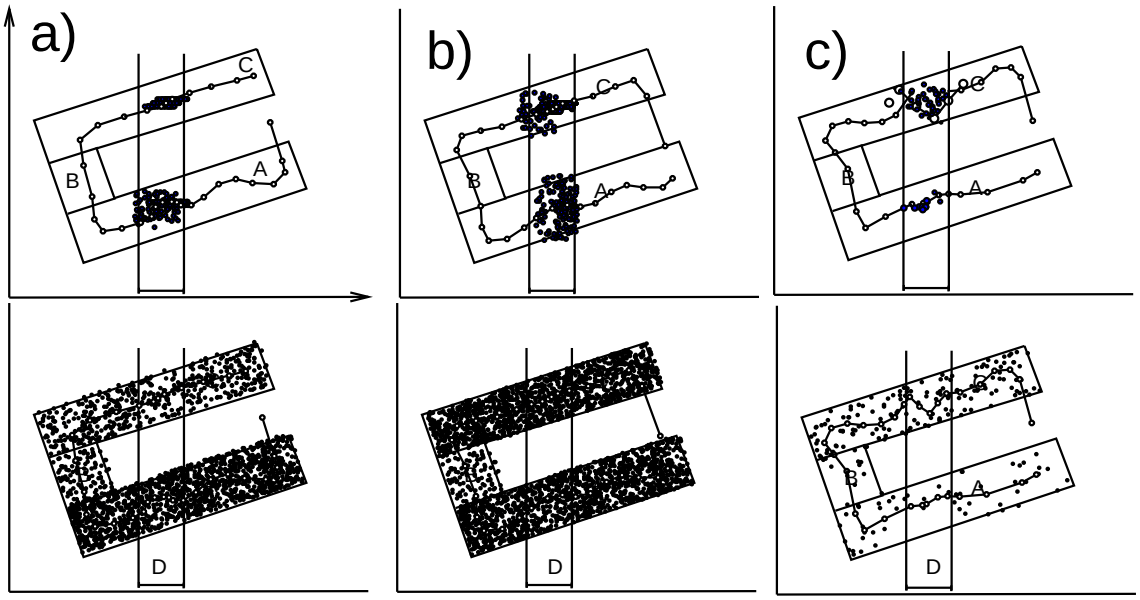
The experiment was repeated three times with different distributions. The original sample set sizes and the distribution of imputed samples on regions A,B and C are shown in the following table. As before, data set D is observed on axis x_1 only.

The distributions of imputed data sets D for cases a), b) and c) (from left to right) are shown in the upper figures. Visual inspection of results shows that data are roughly distributed as in the training data. In the leftmost image, data a), the imputed samples seem to have smaller variance than in fully observed data. The reason is that variances were computed during training, where the missing values were imputed to node mean. Also the idea of using different and independent variances in all nodes need rethinking. Some kind of smoothed version, or true mixture model version would be more reliable.

Table 1: Data set sizes for the toy example and imputation results (for classes A,B and C).

Original data	A	B	C	D (incomplete)	Imputed data	A	B	C
example a)	1500	100	400	300	a)	223	0	77
example b)	1000	100	1000	300	b)	159	0	131
example c)	50	10	150	50	c)	39	0	11

Chart 19: Imputation examples with incomplete data version of the TS-SOM



5.4 Variations of TS-SOM based imputation models

The role of the incomplete TS-SOM training algorithm is to build a model of the joint distribution of missing and observed observations.

$$f_{\mathbf{X}^{\text{imp}},\mathbf{X}^{\text{obs}}}(\mathbf{x}^{\text{imp}},\mathbf{x}^{\text{obs}})$$

The actual imputation of incomplete data can be done in several ways

1. pick a random sample (as before)
$\mathbf{X}^{\text{imp}} \sim f_{\mathbf{X}^{\text{imp}}|\mathbf{X}^{\text{obs}}}(\mathbf{x}^{\text{imp}}|\mathbf{x}^{\text{obs}})$

2. use mean values
$\mathbf{X}^{\text{imp}} = \mathrm{E}[\mathbf{X}^{\text{imp}}|\mathbf{X}^{\text{obs}}]$

3. use random doner $[\mathbf{X}^{\text{imp}}, \mathbf{X}^{\text{obs}}] \in \Omega^{\text{comp}}$

4. etc.. ...

In the following appendices variations of methods 2. and 3. are tested for DLFS and SARS data sets.

NOTE Several things are still missing from this report:

i) Better description of imputation strategies (SOM+donator samples)
ii) Practicalities with real world data (imputation strategy etc.).
iii) More real world data sets.
iv) Conclusions

References

Koikkalainen, P., Tree Structured Self-Organizing Maps, In *Kohonen Maps* (1999) (E.Oja and S.Kaski Eds.), Elsevier-Science.

A. Utsugi, Hyperparameter selection for self-organizing maps, Neural Computation 3 (1997), 623–635.

F. Mulier and V. Cherkassky, Self-Organization as an Itreative Kernel Smoothing Process, Neural Computation 7 (1995), 1165–1177.

T. Hastie and W. Stuezle, Principal Curves, JASA 406 (1989), 502–516.

T. Hastie, R. Tibsirani and J. Friedman, *The Elements of Statistical Learning - data mining, Inference and prediction.* Springer (2001).

V. Vapnik, *Statistical Learning Theory*, Springer (1995).

J. Friedman, Tutorial lecture, Advances in Neural Information Processing NIPS 1995.

M. LeBlanc and R. Tibshirani, Adaptive Principal Surfaces, JASA 425 (1994), 53–64.

J.Rissanen, *Stochastic Complexity in Stochastic Inquery*, World Scientific (1989).

H. Akaike, Information theory and extension of the maximum likelihood principle, *Second International Symposium on Information Theory*, pp. 267–281.

T. Kohonen, Self-organized formation of topologically correct feature maps, Biological Cybernetics 43 (1982), 59–69.

P. Koikkalainen and E. Oja, Self-Organizing Hierarchical Feature Maps, In proc. IJCNN-90 (1990), IEEE press, 279–284

P. Koikkalainen, Progress with the Tree-Structured Self-Organizing Map, In proc. ECAI'94 (1994), A. Cohn (ed.), John Wiley & Sons, 211–215.

Buzo, A., Gray, A.H., Gray, J.D., Markel J.D, Speech Coding Based upon Vector Quantization, *IEEE Transactions on ASSP*, 28, 562–574 (1982).

A Preliminary Test Results For Euredit Using The Danish Labour Force Survey Data

Pasi PIELA and Seppo LAAKSONEN
Statistics Finland
FIN-00022 STATISTICS FINLAND, FINLAND
e-mail: firstname.surname@stat.fi

1. Introduction

This appendix includes the first Statistics Finland test results based on the Danish Labour Survey Data. We present these results using the so-called development data set, that is, we know the real values for each missing one afterwards. Basically, the best specifications of these results will be applied when using the so-called evaluation data set. This kind of work has already been preliminarily done but any results of these are not included in this report.

We decided not to present our test results for each method, separately, but instead for each data set, separately. This gives easier opportunity to compare the results based on various new and traditional methods and techniques. Statistics Finland is involved especially in Workpackages (WP) 4.5 and 5.5 which are concerned Self-Organizing Maps (SOM) techniques. We also are working with traditional methods, thus for WP's 4.1 and 5.1. Since the Danish Labour Force Survey only is concerned on imputation problems, this report is covering WP's 5.1 and 5.5, consequently. It is not fully clear what methods belong to WP 5.1, but we here present our results based on standard Solas procedures, on one hand, and on Regression-based Nearest Neighbour (RBNN) methods (see Laaksonen 2000), on the other. Pasi Piela (2002) has written a special report on SOM imputations, that includes similar points as this one.

In this report, we next in Section 2 present the short introduction to SOM methods and tests results using this methodology. Some comparisons using Solas are included in this section. RBNN methods and results, respectively, are included in Section 3.

2. Data set and TS-SOM Based Imputation

The research group on Software Engineering and Computational Intelligence (SECI) of the University of Jyvaskyla (JyU) has developed a software called the Neural Data Analysis environment (NDA). The software provides a generic application platform for computational intelligence with many proven examples of real world applications. The main emphasis of the software is to aid methodological development of knowledge discovery, data analysis, and modelling in general (see Hakkinen 2001). Techniques are mainly based on neural networks, but the system also includes a large data

The Danish Labour Force Survey (1996) consists of Danish population register records for individuals selected for interview. The synthetic version of this data set was given as training and development data for imputation methods. The data consist of only 14 variables with little information, four relating to type of response. Annual income (DKK) is the only variable needed to impute, the missingness rate being 26.8 %. The data are at person level having 200,000 observations, information about households is not available here. These 13 auxiliary variables are categorical, except person's age. They have 2-4 classes, except AREA (area of living), BUSINESS (last employment) and EDUCATION, for example, have 4 classes:

1 = Private Industry
1 = Primary school only
2 = Other private business
2 = Craftsman, Skilled labour, High school only
3 = Government employed
3 = Long education, school teacher, university, etc.
-9 = Not applicable
-9 = No information

The complete data vectors are used here as training data. That is, we select all 146,323 observations with known value of income. Another alternative would be selecting all the observations and training TS-SOM for these without the variable INCOME, but this is not necessarily appropriate in this simple case due to losing information of the relationship between INCOME and explanatory variables.

First we make data analysis by building TS-SOM for our training data set by NDA software and viewing its different layers. For example, it is easy to observe graphically the basic statistics of INCOME for different clusters and comparing them to the statistics of background variables.

Variables are scaled for nearest neighbour imputation (this thus assumes that each variable has the same weight), where for the missing component $\mathbf{x}_i(j)$ that belongs to a cluster b

$$\mathbf{x}_i(j) = \mathbf{x}_n(j) : n = \arg\min_{k \in cl_b} \|\mathbf{x}_i - \mathbf{x}_k\|, \tag{12}$$

so that all categorical variables are binarized. Thus, for variable l of m classes we have corresponding m (0/1)-variables. The continuous background variable, AGE, is scaled to [0, 1] based on min/max ranges:

$$\mathbf{x}' = \frac{\mathbf{x} - \min \mathbf{x}}{\max \mathbf{x} - \min \mathbf{x}}$$

However, in the case of several skewed distributed continuous variables there are naturally other appropriate methods for equalization. Also, when missingness of several components occurs the distances in (12) can be weighted by simply comparing the number of missing components and the number of variables for every data vector.

An obvious problem in this kind of hot-deck type of imputation is its computational complexity (related to computation time) that is $\mathcal{O}(N^2)$ due to the full search among N data vectors. By using the above method and TS-SOM the complexity can be reduced to $\mathcal{O}(N \log_p M + N^2/M)$ where the complexity of the TS-SOM algorithm is $\mathcal{O}(N \log_p M)$, p being the number of sons of each node (in two dimensional case: p = 4), and for the imputation within M clusters it is $\mathcal{O}(N^2/M)$. In practice, this was clearly observed for the example in question as follows:

Table 2: Computation time of the nearest neighbour imputation. Synthetic Danish Labour Force Data set, N = 200,000.

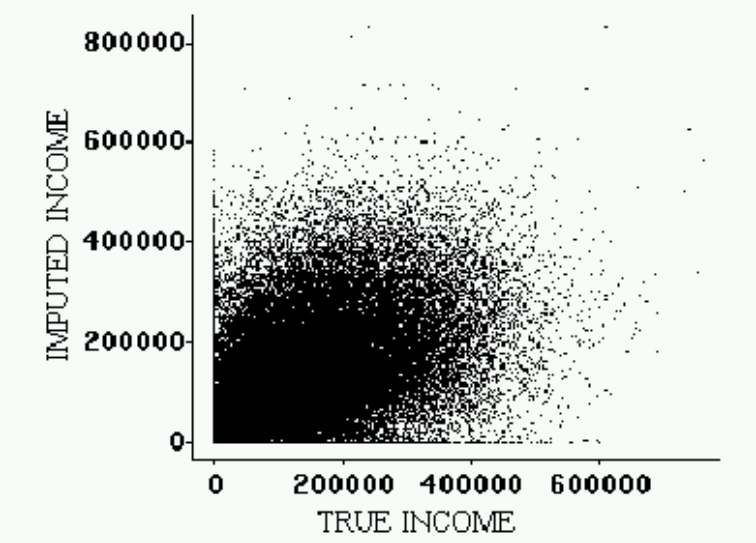
Nearest Neighbour Imputation	Computation Time Pentium(r) II Processor (500)
without TS-SOM mapping (layer = 0)	more than 12 hours
layer 2 as imputation layer (16 neurons)	59 minutes
layer 3 as imputation layer (64 neurons)	20 minutes

As shown in Figure 20, the imputation does not give good results at the unit level, which is expected because of lack of good background information. But for this case the method really seems to work at the aggregate (or data) levels. Specifically, the line for observations with $\hat{Y} = 0$ and $Y* > 0$ is quite similar to the line for observations with $\hat{Y} > 0$ and $Y* = 0$, but the lines are long including several wrong imputed values. Furthermore, the linear regression model (y = true income, x = imputed income) has surprisingly high estimate of the intercept parameter (102034) but when modelling without intercept quite a good model with high R-square is observed.

As Häkkinen (2001) points out, there are some obvious problems in this kind of traditional method together with TS-SOM. There might be clusters having only missing values and there is a risk to lose the nearest data vector to another cluster, which might be a problem here too.

Cluster centroid imputation, which replaces the missing values with corresponding values of the centroid of the same cluster, is not appropriate in this case. Totals are clearly over-estimated and variances

Chart 20: Scatterplot of imputed INCOME ($\hat{Y}$) and corresponding true values (Y*) from nearest neighbour imputation on the 3rd TS-SOM layer.



under-estimated. On the other hand, the structure of TS-SOM gives possibilities to solve problematic situations: the centroid can be interpolated from a parent neuron of the upper TS-SOM layer or it is possible to use neighbour clusters and neurons as well in finding the appropriate donor or the centroid needed.

There are, naturally, a number of ways to impute by MLP models. Table 3 presents results for two structurally different groups of local backpropagation MLP imputation models for modelling dependencies between background and INCOME variables. In method I, for example, we first build TS-SOM and on its first level we use the backpropagation algorithm to create four local MLP models with two hidden layers of four and six neurons connected sequentially by logistic sigmoid units, one for each neuron (see Table 3). But these kinds of MLP models are clearly problematic for the case in question; they do not give any 0s as estimates of INCOME, and variances are under-estimated.

3. Regression-based Nearest Neighbour (RBNN) Method for Imputation

This methodology first exploits a best possible regression and statistical modelling technique as imputation model, and in the second step, it takes advantage of the predicted values of this model and uses this metrics for searching the nearest or a near value of each missing value. In some sense, it is an old technique, but not much used. It is not well known, in which cases the methodology is advantageous, and in which cases it is not. Laaksonen (2000) has tested this method quite successfully in the case where the missingness rate is rather high, the distribution of a target variable is skew and the model fitting is not good. This technique was superior in that case to standard regression technique, for example. The DLFS data resemble in some sense the situation of the example of Laaksonen. Hence we made attempts to apply RBNN.

The DLFS data are explained already in Section 2. For constructing a regression model for INCOME we tried the two types of specifications for the dependent variable, linear and log-transformations. The latter one is often used in econometrics. The model fitting based on R-square is much smaller for log-transformations, but this is not the most important point, necessarily, for the goodness of imputation. The predictability of the model is more important, and this depends much on the auxiliary variables available. As in Section 2 noted, there are not any very good such variables in the data set. But: we tried to do our best and exploit all variables with somewhat different specifications: SEX, MARRIAGE, COHABITE, EDUCATIO, BUSINESS, UNEMPLOY, CHILDREN, AREA, AGE, LETTER

Table 3: Example results using linear regression (Solas 3.0^{TM} implementation) and various SOM (NDA Implementation) based methods of INCOME variable.

	True val.	Cen.	Lreg	NN,L0	NN,L1	NN,L2	NN,L3	MLP I	MLP II
Mean	158108	169177	170287	158068	158094	159799	159408	167354	166806
Stddv	107193	56806	61498	107856	107855	108380	108253	69143	68920
95%	362639	259743	272202	363148	362674	363374	363712	280030	277061
Q3	221423	204815	215989	220904	221534	224541	223804	225255	219764
Md	140971	173937	168978	140105	139715	141657	141616	173607	162920
Q1	76691	122951	122344	77004	76869	77914	77779	107494	109096
Q3-Q1	144732	81864	93645	143900	144666	146627	146025	117762	110668
DL1	0	73079	73678	90865	91072	91593	91639	67109	67279
DLmax	0	564345	520718	664950	664950	671202	664950	508255	518621

Notes:

i) The following shorthand notations are used: Cen. (Centroid, 4th SOM layer) Lreg (Lin. Regression) NN,L0 (Nearest Neighbour, 0th SOM layer) NN,L1 (Nearest Neighbour, 1st SOM layer) NN,L2 (Nearest Neighbour, 2nd SOM layer) NN,L3 (Nearest Neighbour, 3rd SOM layer) MLP I (TS-SOM Imp. layer 1 (4 clus.), 4 neurons on 1st Hidden Layer , 10 neurons 2nd Hidden Layer) MLP II (TS-SOM Imp. layer 3 (64 clus.), 6 neurons on 1st Hidden Layer, 10 neurons 2nd Hidden Layer)

ii) true mean / standard deviation of the missing incomes is 158,108/107,193 and true mean / standard deviation of the whole data set is 181,724/116,064.

iii) $d_{L1}(\hat{\mathbf{Y}}, \mathbf{Y}^*) = (\sum_{i=1}^{n} w_i |\hat{Y}_i - Y_i^*|) / \sum_{i=1}^{n} w_i$, where $w_i = 1$ if $i \in N$

AND PHONE. Some interactions between variables were attempted as well.

Table 4 gives some results. In all cases of Table 4, we made the final imputation at full data level, although this could be done at sub-data levels, the number of such levels could be in this case numerous. We tested some sub-data levels such as AREA, SEX * AGE and BUSINESS, but the results of these were not essentially better or worse, and hence we used this simpler formulation.

We cannot definitely say what method and specification is best. The better the model is, the better the imputation result is, however, when using the same transformation. Log-transformation does not help in general, but it gives the best result for standard deviation. Best nearest neighbour imputation models are a little bit preferable to RBNN as far as the mean and some distribution figures are concerned. On the other hand, the DL1 values for RBNN are lower, but still rather high.

All results may be compared with the last column of Table 4 which gives results from one single random hot decking (if other random numbers are used, the results are little varying). The distributional figures of that method are close to those of the respondents' data set, and show how biased are the estimates if any imputations have not been done. All imputation methods seem to give essentially better results than obtained from random hot decking. This is concerned the DL1 values too.

Table 4: Example results using RBNN

	True value	RBNN 1	RBNN 2	RBNN 3	RBNN 4	RBNN 4	Hot Deck
Mean	158108	164471	164486	159441	159448	160465	176056
Stddv	107193	109609	109686	107654	108347	107605	116310
95%	362639	369574	368487	360779	363389	362812	390072
Q3	221423	230046	230916	223921	224346	224090	248847
Md	140971	147920	147544	142718	142043	143236	160464
Q1	76691	81388	81176	77601	76785	79232	86722
Q3-Q1	144732	149658	149740	146320	147561	144858	162125
DL1	0	89853	89921	87592	88208	89334	124964
DLmax	0	701298	652273	672713	793438	645755	731064

Notes:

RBNN 1 Linear INCOME, no inter-actions, No random term in model, No PHONE
RBNN 2 Linear INCOME, some inter-actions, No random term in model, No PHONE
RBNN 3 Linear INCOME, some inter-actions, No random term in model, Yes PHONE
RBNN 4 Linear INCOME, some inter-actions, random term in model, Yes PHONE
RBNN 5 Log-linear INCOME, some inter-actions, No random term in model, Yes PHONE
Hot Deck Single Random Hot Decking Method for Bench-marking

References of appendix A

Häkkinen, E. (2001). Design, Implementation and Evaluation of the Neural Data Analysis Environment. PhD thesis. Jyvskyl University Library, Jyvskyl, Finland.

Koikkalainen, P. (1995). Fast Deterministic Self-Organizing Maps. In Fogelman-Souli, F. and Gallinari, P., eds., Proc. ICANN'95, Int. Conf. on Artificial Neural Networks, volume II, pages 63-68, Nanterre, France. EC2.

Koikkalainen, P. (1999). Tree Structured Self-Organizing Maps. In Oja, E. and Kaski, S., eds., Kohonen Maps, pages 121-130. Elsevier, The Netherlands.

Koikkalainen, P. and Oja, E. (1990). Self-Organizing Hierarchical Feature Maps. In Proc. IJCNN-90-Wash-DC, Int. Joint Conf. on Neural Networks, volume II, pages 279-285, Piscataway, NJ., IEEE Service Center.

Laaksonen, S. (2000). Regression-Based Nearest Neighbour Hot Decking. Computational Statistics.

Piela, P. (2002). Introduction to Self-Organizing Map Modelling for Imputation. Draft paper for Euredit.

B Preliminary Imputation Test Results For Euredit Using the UK SARS Data

Pasi PIELA
Statistics Finland
FIN-00022 STATISTICS FINLAND, FINLAND
E-mail: Pasi.Piela@stat.fi

1. Introduction

This report includes the first Statistics Finland test results based on the UK SARS Data. We present these results using the so-called development data set, that is, we know the real values for each missing one afterwards. Specifically, SARS data set needs some pre-processing to get imputation data set out without errors. Basically, the best specifications of these results will be applied when using the so-called evaluation data set. This kind of work has already been preliminarily done but any results of these are not included in this report.

We decided not to present our test results for each method, separately, but instead for each data set, separately. This gives easier opportunity to compare the results based on various new and traditional methods and techniques. Statistics Finland is involved especially in Workpackages (WP) 4.5 and 5.5 which are concerned Self-Organizing Maps (SOM) techniques. We also are working with traditional methods, thus for WP's 4.1 and 5.1. Since the Danish Labour Force Survey only is concerned on imputation problems, this report is covering WP's 5.1 and 5.5, consequently. Pasi Piela and Seppo Laaksonen (2001) have written a special report on WAID imputations, that includes SARS results from regression and classification trees. It will be modificated strongly in the near future. In this report every method is Tree-Structured Self-Organizing Map (TS-SOM) methodology based, but in the future we will also include results from purely classical methods. This methodology has been previously presented in the EUREDIT papers and will not be discussed here. See Koikkalainen (2002).

The software used is the current version of NDA developed by the research group on Software Engineering and Computational Intelligence (SECI) of the University of Jyvskyl, JyU. This version includes new user-interface with some specialities developed by Ismo Horppu with the EUREDIT JyU group leader Pasi Koikkalainen.

We next present the short introduction to data with notations from our point of view. Preliminary test results are presented in Section 4 and following tables.

2. Data set and variables

The EUREDIT development data set of the Household SARS (UK Census sample of anonymised records for individual households) includes 44703 observations, living in 19136 households. We selected four imputation variables from the UK Household data set to be presented here: TENURE (Tenure of household space), ROOMSNUM (Number of rooms), AGE and HOURS (Hours worked weekly). TENURE and ROOMSNUM are household level variables while AGE and HOURS have unit level values (see Table 5).

3. Practical notations

1) The so called Y2 SARS data set with missingness have been created by merging erroneous and the clean data together and replacing true values by missing ones if a corresponding value in the erroneous data is missing. We have not used any kind of editing rules to create more missingness. However, it has been noticed that there is also lot of editing failures in the clean data, which could implicate more missingness if taking into account.

2) The order of the observations inside the data is anything but random and can easily been used to

Table 5: Missingness in selected imputation variables. HOURS excl. -9 corresponds a data set without -9 values of HOURS.

	Unit level		Household level	
Variable	Available values	Missing values	Available values	Missing values
AGE	44077	3626 (7.6%)	-	-
HOURS	46147	1556 (3.3%)	-	-
HOURS excl. -9	24795	1556 (3.3%)	-	-
ROOMSNUM	44658	3045	17922	1214 (6.3%)
TENURE	45550	2153	18245	891 (4.7%)

get very good results for couple of variables at least. PNUM (Person number in household) makes imputation of variable RELAT (Relationship to household head) too easy. If PNUM = 1, it is very probable that RELAT = 0 etc. Furthermore, NUMBER (Person number) is a very good hot deck sorting variable for AGE. But we have tried not to take any advantage from any of these surrealistic connections between target variables and index variables.

3) There is naturally number of solutions to create practical data for household level variable imputation. We have always solved this simply by selecting only one observation from each household, namely household head. It is a natural choice, but we have not studied other solutions for this problem so far.

4) Value -9 means not applicable value. Especially in the SARS data, these values clearly have meaning and should be used with the other values of the selected explanatory variables.

5) The missing data and clean data include equal number of -9 observations for HOURS, and that why the data for imputation of HOURS have been reduced, see Table 5.

6) The data include four continuous variables: HOURS, AGE, PERSINHH (Person number within household) and ROOMSNUM. When used as an explanatory variable, HOURS and AGE have been classified into 6-10 classes, hence only ROOMSNUM and PERSINHH have been regarded as continuous ones.

7) Handling both continuous and categorical variables has been problematic from a scaling point of view; this is a real life problem. Binarization might give too much weight for categorical variables compared to scaling from 0 to 1 of continuous ones. This problem needs further studying, although handling continuous variables is not seen as a bad problem in the data set in question, because of their partly categorical nature (can be used as categorical ones) and connections to other variables.

8) Missing values of categorical variables have been considered as their own class and then corresponding binary sub variables have been simply deleted, because we have not found much practical meaning behind missingness in this experimental data set. Our tests have shown this to be quite a good solution.

9) The results in this paper are calculated only by using imputed values and their corresponding true values; there have not been any unimputed observations from any method mentioned. Unit level goodness measures of the imputation are the following simple ones, abbreviated as DL1 and DL2:

$$d_{L1}(\hat{\mathbf{Y}}, \mathbf{Y}^*) = (\sum_{i=1}^{n} w_i |\hat{Y}_i - Y_i^*|) / \sum_{i=1}^{n} w_i, \quad \text{where} \quad w_i = 1 \text{ if } i \in N$$

$$d_{L2}(\hat{\mathbf{Y}}, \mathbf{Y}^*) = \sqrt{(\sum_{i=1}^{n} w_i (\hat{Y}_i - Y_i^*)^2) / \sum_{i=1}^{n} w_i}, \quad \text{where} \quad w_i = 1 \text{ if } i \in N$$

4.1 Imputation of AGE

Lot of tests has been made in trying to get as good results as possible both at aggregate level and at unit level. Specifically number of imputed 0s seems to be one satisfactory indicator to the goodness of imputation. We have not yet been able to totally explain, why nearest neighbor imputation fails when compared to other methods; there are too much imputed 0s (see Table 6). 0- observations seem to be quite special, and because there are errors in the clean data it implicates existence of wrong nearest neighbors.

First, the best results came from the random imputation within the TS-SOM clusters at 4th level, which means SOM mapping of 256 clusters; there might exist few empty clusters. Picture presents nice visualization of the 4th level SOM map. Naturally, this is a very trivial example, but it shows that SOM algorithm has made a good discretization for the data.

However, in the method above we have not taken into account households except using household level variables. That why, we chose a special nearest neighbor method within the household - when possible - by using RELAT as a sorting variable. In this special method, first data are divided into two parts (part1: if RELAT in (3-6, 12)) and then if the nearest neighbor imputation within household is possible, in other words, if there are at least one known value of AGE for the missing one from the same household, we impute. This leaves 1261 observation unimputed. Those will be imputed by TS-SOM 4th level random imputation separately from these NN imputed values. This method gives the best results we have got so far, see last row * of Table 6.

Table 6 and other tables as well include also one trivial benchmarking method, namely, random overall imputation which measures the effect of randomness and degree of difficulty in one easy way. Surprisingly the estimates of mean and standard deviation are very close to the true one. But luckily values for DL1 and DL2 are far away. We know also non-trivial benchmarking methods are needed for the future reports.

4.2 Imputation of HOURS

Any special problems of replacing missing values of working hours have not been noticed. Imputation results are rather good as seen in Table 7 and mean deviations quite small. It should be noted that the true deviation of the HOURs is high which makes imputation more challenging - besides it clearly separates average based methods from the others.

Prediction from the normal distribution that is estimated for all the SOM clusters by calculated using variance and mean gives good results except that it fails to estimate deviations or percentiles. Random imputation within the clusters seems to be reasonable as well. But full random imputation without any other helping method give nice estimates too.

4.3 Imputation of ROOMSNUM

Household level variable, Number of rooms, have been tried to impute many times in all versions of the SARS data. TS-SOM based nearest neighbor imputation gives very good results when using five explanatory variables (in finding the correct donor!) mentioned in Table 8. Mean deviations are relatively small and distribution of the imputation data remains very well.

4.4 Imputation of TENURE

Because of the methodological and technical problems the imputation of the tenure of household space has been problematic. Only way to impute TENURE with the current version of NDA was to use it as a continuous variable. It is not actually totally unsuitable way to handle this variable, but imputation fails. These problems will be repaired to the next version. Note that SOM mapping has been made using most of the variables of the data. We expect good results from the future testing.

References of appendix B

Koikkalainen, P. (2002) : Description of The Imputation Methodology based on The Tree-Structured Self-Organizing Map. Draft Report for EUREDIT FP5 Project, University of

Jyvskyl, Finland.

Piela, P. & Laaksonen S. (2001) Automatic Interaction Detection for Imputation - Tests with The WAID Software Package. Contributed Paper for Federal Committee on Statistical Methodology Research Conference, Washington, DC Area.

Table 6: NDA test results for AGE.

Method	Mean	Std.	25 %	Med.	75 %	95 %	DL1	DL2	zeros (%)
True values (N=3626)	37.26	23.05	19	35	55	76	0	0	0.012
Random overall	37.27	22.90	19	35	55	77	13.90	14.49	0.015
SOM l=2, NN	39.96	27.37	15	41	61	87	13.16	18.89	0.084
SOM L=2, R	37.42	22.89	19	35	55	77	5.14	9.41	0.018
SOM l=3, R	37.57	22.84	19	36	55	77	4.80	9.48	0.013
SOM l=4, R	37.26	22.91	20	35	55	76	4.72	8.79	0.016
SOM l=5, R	37.71	23.04	19	36	56	77	4.98	8.84	0.014
NN: Re+SOM l=5, R	37.00	23.16	19	35	55	76	4.33	7.33	0.018

Notes: NN = nearest neighbor imputation, R = random imputation, TSSOM L = TSSOM clustering, at level L. L = k means that the data have been divided into 4^k clusters/subclasses. R = ROOMSNUM, E = ECONPRIM, M = MSTATUS, Re = RELAT, H = HHSPTYPE. DL1 and DL2: see notation 9 in 3.

Table 7: NDA test results for HOURS.

Method	Mean	Std.	25 %	Med.	75 %	95 %	DL1	DL2
True values (N=1556)	35.44	12.46	32	39	40	50	0	0
Random overall	35.61	12.09	32	39	40	54	12.07	17.08
TSSOM l=3,NN:I,E	32.72	14.15	24.5	38	40	50	9.21	15.05
TSSOM l=3,NN:I,E,A,S,Re	33.43	14.73	26.5	38	40	52	8.96	14.48
TSSOM l=4, Normal prediction	34.78	8.23	31	37	40	45	7.10	10.56
TSSOM l=5, Normal prediction	34.78	8.8	31	37	41	45	6.92	10.35
TSSOM l= 4, Random	35.11	11.62	32	38	40	50	8.59	13.18
TSSOM l=5, Random	34.94	12.50	32	38	40	50	8.75	13.76

Notes: NN = nearest neighbor imputation, Random = random imputation, Normal prediction = Normal curve prediction imputation, TSSOM L = TSSOM clustering, at level L. L = k means that the data have been divided into 4^k clusters/subclasses. I = ISCO1, R = ROOMSNUM, E = ECONPRIM, M = MSTATUS, Re = RELAT, H = HHSPTYPE, S = SEX, A =AGE. DL1 and DL2: see notation 9 in 3.

Table 8: Imputation Results for ROOMSNUM (number of rooms).

Method	DL1	DL2	Number of Rooms (%)					
			1-2	3	4	5	6	7+
True values (N=1214)	0	0	3.95	8.57	23.39	31.55	20.43	12.1
Random overall	1.55	2.13	4.78	8.24	22.89	32.37	18.78	12.85
TSSOM L = 2,NN:H,P	1.27	1.72	5.44	6.84	22.41	28.25	20.68	16.39
TSSOM L = 1, NN:E,H,P,T,M	1.25	1.73	4.86	9.14	21.33	27.35	21.33	15.98
TSSOM L = 2, NN: E H P T M	1.23	1.70	4.86	7.74	21.00	30.07	22.90	13.42
TSSOM L = 3, NN: E H P T M	1.21	1.68	3.79	8.65	19.85	31.71	21.91	14.07
TSSOM L = 5, Centroid	0.99	1.42	1.31	7.74	36.99	43.82	7.08	3.05

Notes: NN = nearest neighbor imputation, Random = random imputation, Centroid = cluster centroid imputation, TSSOM L = TSSOM clustering, at level L. L = k means that the data have been divided into 4^k clusters/subclasses. P = PERSINHH, E = ECONPRIM, M = MSTATUS, H = HHSPTYPE, T = TENURE. DL1 and DL2: see notation 9 in 3.

Table 9: Imputation Results for TENURE (Tenure of household space).

Method	DL1	DL2	Tenure of household space (%)					
			1	2	3	4	5-6	7
True values, N = 891	0	0	23.91	40.63	3.93	4.38	3.7	23.46
TSSOM L = 3, Centroid	1.49	2.16	8.31	48.37	21.77	11.45	10.37	0
TSSOM L = 2, NN: E H P R M	1.79	2.84	33.33	29.74	3.03	4.26	4.94	24.69
TSSOM L = 3, NN: E H P R M	1.72	2.77	31.76	31.31	3.25	5.27	4.72	23.68
TSSOM L = 2, Random	1.98	3.08	23.57	41.75	3.03	3.59	4.04	24.02

Notes: NN = nearest neighbor imputation, Random = random imputation, Centroid = cluster centroid imputation, TSSOM L = TSSOM clustering, at level L. L = k means that the data have been divided into 4^k clusters/subclasses. P = PERSINHH, E = ECONPRIM, M = MSTATUS, H = HHSPTYPE, R = ROOMSNUM. DL1 and DL2: see notation 9 in 3.