Otto-von-Guericke-University of Magdeburg

Faculty of Computer Science
Department of Knowledge and Language Engineering

Internship Report

Application of Growing Self-Organising Maps for Analysing Manufacturing Data

Author:

Georg Ruß

July 5, 2005

Academic Supervisors:

A/Prof Saman Halgamuge, Md Azharul Karim

University of Melbourne
Department of Mechanical and Manufacturing Engineering
Parkville, 3052, Victoria, Australia

Prof. Dr. Rudolf Kruse

Universität Magdeburg
Fakultät für Informatik
Postfach 4120, D–39016 Magdeburg, Germany

Ruß, Georg:
Application of Growing Self-Organising Maps for Analysing Manufacturing Data
Internship Report,
Otto-von-Guericke-University of Magdeburg,
2005.

Acknowledgements

I'd like to say thanks to all the people at the Mechatronics Research Group at the University of Melbourne, Department of Mechanical and Manufacturing Engineering; you've made my six months of training time worthwile and mind-opening. Special thanks go to my academic supervisors A/Prof. Saman Halgamuge, Md Azharul Karim and Prof. Rudolf Kruse from the University of Magdeburg, Germany; you have provided the necessary ideas and steps to make this research project a success. My very personal thanks are directed towards Genevieve Coath, for proof-reading not only this report, but also for being a true friend, taking care of WalTher and providing emotional support whilst being far, far away from Germany. The 2nd and 4th assumptions from the last sentence are also valid for Jessica Christel, being on the other side of the world. Last, but not least, I'd like to thank my parents and family for supporting my stay in Australia financially and emotionally.

Contents

List of Figures

List of Tables

List of Abbreviations

BMU	Best-Matching Unit
CQ	Clustering Quality
GP	Growing Phase
GSOM	Growing Self-Organising Map
GT	Growth Threshold
LL	Learning Length
LR	Learning Rate
MLP	Multi-Layer Perceptron
NS	Neighborhood Size
SF	Spread Factor
SOM	Self-Organising Map
SP	Smoothing Phase
SVM	Support Vector Machine

Chapter 1

Introduction

1.1 Purpose of this document

The intended purpose of this document is to provide my academic supervisor Md Azharul Karim with a detailed documentation of the work done during my 24 weeks of job training at the Mechatronics Research Group, Department of Mechanical and Manufacturing Engineering, University of Melbourne, Australia. It is also being authored in support of my supervisor's final PhD thesis.

Furthermore, as the mentioned job training is required within my conditions of study at the Faculty of Computer Science, University of Magdeburg, Germany, this document also provides the mandatory project report which is supposed to be written accompanying the training time.

As my research work continues during the time of writing this document, certain results from it may find their way into various journal papers.

1.2 Motivation

Imagine working at a large semiconductor company and being responsible for quality control at the end of multiple assembly and production lines. Since there is no fuzzy classification of chips, that is, there is only a working product or non-working 'waste', you are definitely interested in discovering where possibly faulty steps might be situated and with which amount they contribute to the final product. The only input data you possess to perform your work consists of an enormous amount of values from the assembly line meters; tedious working hours have been invested by colleagues to find out manually whether the final product has been a working exemplar or one that was out of order. Those other people, however, relied on their knowledge about the production process and did not target relationships inside the data which might have led to faulty products. Your task as a data analyst is to detach from the physical background and instead to work on the given data exclusively. Right from the beginning, based on the

high dimensionality of the data, you are sure *not* to use statistical, descriptive methods to find distinguishing attributes. Since neural network learning algorithms are known to operate well under these input data conditions, you decide to use the extension of the Self-Organising Map, the *Growing* Self-Organising Map, to try to find the (presumed) distinguishing attributes (= possibly faulty production line points).

1.3 Given task

Given high-dimensional, two-class, manufacturing data, evaluate without further knowledge about the data itself the feasibility of applying the Growing Self-Organising-Map (GSOM) algorithm to it and try to find distinguishing attributes within the given dataset.

Certainly this condensed description needs some explaining: The 'high-dimensional' and the 'manufacturing' keywords will be explained in the following section; the 'two-class' keyword refers to the given dataset as a discrimination problem between two classes, namely 'good' and 'bad'[1] products. Now, the task is to use the GSOM algorithm and to ascertain whether it is suitable to solve this discrimination problem and to automatically separate the input dataset into two classes. The term 'without further knowledge' is self-evident, with an important exception: the data distribution between 'good' and 'bad' is known beforehand, a fact that makes benchmarking of the GSOM possible.

1.4 Manufacturing data

To exemplify the manufacturing data's impact and, in conclusion, the certain need for computer-aided processing see table 1.1 from which the data dimensionality and layout can be obtained. The given raw data contains several fields that are either irrelevant to the data mining process (such as identifiers or constant values) or need further handling as to not mislead the chosen neural network into wrong conclusions. Therefore, preprocessing actions had to be taken, see Section 3.1 for details.

From table 1.1 the dimensionality of the data can be derived as having more than 16,000 inputs (or 'vectors') with 133 attributes each. The only additional fact given to estimate the outcome of the respective data mining technique was a threshold of 8750 referring to the attribute labeled 'REF' which discriminates between good and bad products.

[1] The words 'good' and 'bad' have been used to describe the two possible classes throughout the author's training time at the University of Melbourne; they do not constitute any valuation and could have been chosen arbitrarily. As mentioned in the introduction, this report also serves as a documentation to my Australian supervisor and is, in that regard, compliant to his thesis.

	NAME	REF	C1	C2	...	C92	X93	...	X131
1	J546040_12_1	9628	1.00E-09	2.21E-02	...	15	SMO	...	Dec-18
2	J546040_12_1	9496	1.20E+00	2.21E-02	...	33	SMO	...	Dec-19
3	J546040_15_1	9336	1.05E-09	2.34E-02	...	33	SMO	...	Dec-19
4	J548375_01_3	9148	1.36E-09	2.26E-02	...	14	SMO	...	Dec-19
5	J548375_04_6	9480	-1.00E-08	2.50E-02	...	23	SMO	...	Dec-19
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
16381	F606617_20_3	9728	-2.69E-08	2.24E-02	...	17	SMO	...	Feb-26

Table 1.1: Data dimensionality and layout before preprocessing

1.4.1 Proposed solution / Document structure

To understand the underlying principles of the GSOM, this special technique within the neural network technologies will be explained in detail in Chapter 2. Now that you know about the basics, Chapter 3 starts with the preprocessing as the first action that has to be taken, then describes a quality measure, both as prerequisites to invoking the GSOM. Appropriate parameter sets for the GSOM are established next, a sampling method will be developed and the final simulation setup is described. Numerical, formatted results as well as resulting maps from this setup will be described in Chapter 4. Chapter 5 finalises and summarises this document and points out possible further work.

However, the second chapter follows and starts with a short introduction to artificial neural networks.

Chapter 2

Basic techniques

2.1 Short introduction

Artificial neural networks

Artificial neural networks are adaptive models that can learn from input data and generalise (therefore simplify) learned things. They extract the essential characteristics from the numerical data as opposed to memorising all of it. This offers a convenient way of reducing the amount of data (neural networks have been used to process millions of inputs [5]) as well as to form an implicit model without having to manually form a traditional, physical or logical model of the underlying phenomenon. In contrast to traditional models, which are *theory-rich and data-poor* the neural networks are *data-rich and theory-poor* in a way that little or no a-priori knowledge of the problem is present and also that certain properties are hard to prove but easily to accept based on empirical observations.

Self-Organising Maps

Self-Organising Maps have been widely used in data mining – or knowledge exploration – to visualise high-dimensional data and to reduce its dimension to just a few representative prototypes. It is therefore crucial to sustain the highest possible accuracy during the data mining process. Many variants extending the conventional SOM's capabilities were proposed (one of which is the *Growing* Self-Organising Map) to allow more flexibility and adaptiveness by introducing controllable and consecutive growth during the training process. This chapter will explain the SOM and GSOM algorithms in detail to provide the necessary basics for understanding the rest of this report.

2.2 SOM

The Self-Organising Map developed by Prof Teuvo Kohonen [6] is one of the most popular neural network models. The SOM algorithm is based on unsupervised competitive learning causing the training to be entirely data-driven and the neurons on the map to compete with each other. Supervised algorithms like the Multi-Layer Perceptron or Support Vector Machines require the target values for each data vector to be known in advance whereas the SOM does not have this limitation. The SOM has been invoked in a large variety of tasks ranging from process modeling [3] over large textual document collections [4] to multimedia feature extraction [9].

The important distinction from Vector Quantisation techniques is that the neurons are organised among a regular grid and that along with the selected neuron (the Best-Matching Unit) also its neighbors are updated, by means of which the SOM performs an ordering of the neurons. In this respect the SOM is a scaling method projecting data from a high-dimensional input space onto a typically two-dimensional map (see figure 2.1). Hence, similar input vectors get mapped to neighboring neurons on the output map.

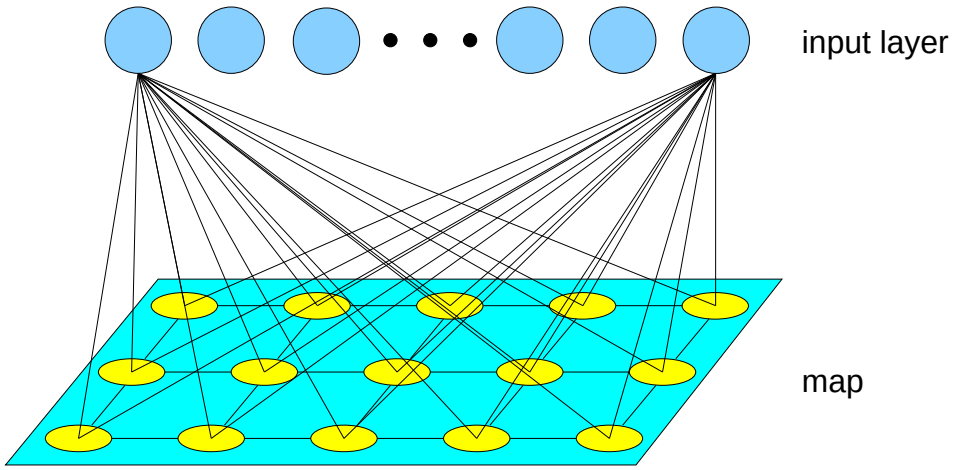
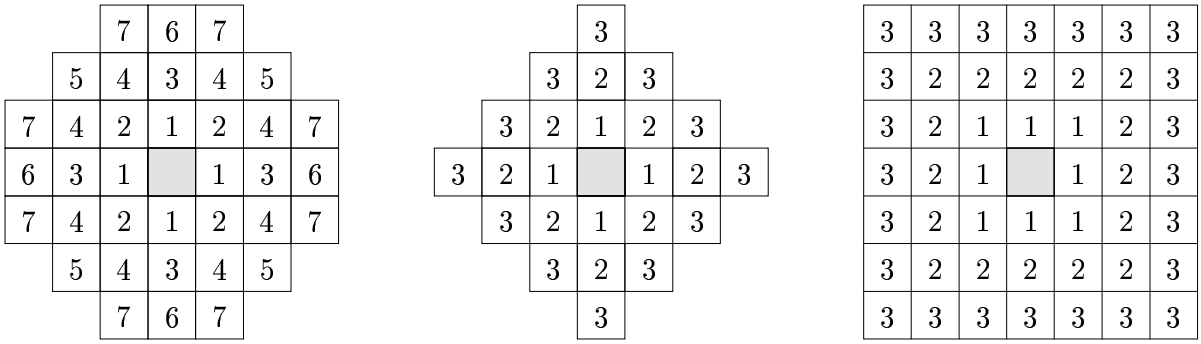


Figure 2.1: Neighborhood preserving mapping from input layer to two-dimensional map

2.2.1 Structure

A SOM is formed of neurons located on a usually 2-dimensional grid having a rectangular or hexagonal topology. Each neuron of the map is represented by an n-dimensional weight vector $m_i = [m_{i1}, \cdots, m_{in}]^T$, where n is equal to the respective dimension of the input vectors. Higher dimensional grids might be used but their visualisation is not as straightforward as it is for two-dimensional maps.

The map's neurons are connected to adjacent neurons by means of a neighborhood relation which superimposes the structure of the map. Immediate (directly adjacent) neighbors belong to the 1-neighborhood $N_{i,1}$ of neuron i. As mentioned, in the 2D-case the neurons of the map can be arranged either on a rectangular or a hexagonal grid, for an illustration including the neighborhood relation see figure 2.2.



a. b. c.

Figure 2.2: Equidistant SOM neighborhoods on a rectangular grid with (a) Euclidean, (b) city-block, (c) chessboard distance. [7]

The number of neurons naturally determines the granularity of the resulting mapping, which in turn influences the accuracy and the generalisation capability of the SOM.

2.2.2 Initialisation

In this section's basic SOM algorithm the number of neurons and the topological relationship are fixed from the beginning, as opposed to the GSOM algorithm explained in Section 2.3. The number of neurons should usually be selected as large as possible with the neighborhood size affecting the smoothness and generalisation capability of the mapping. The mapping does not suffer considerably even when the number of neurons exceeds the number of given input vectors; selecting the neighborhood size appropriately seems to be much more important. As the size of the map increases to e.g. thousands of neurons the training phase becomes computationally almost infeasible for practical applications; in this term the SOM does not exhibit much difference from the GSOM algorithm whose (exemplary) computation times can be obtained in detail from Chapter 4.

An initialisation of the weight vectors has to be provided before starting the training phase. Since the SOM is robust in regards to the choice of the initialisation it will finally converge, albeit the proper choice of initial values can save some computational effort. Two often-used initialisation procedures are the following:

- random init: weight vectors are initialised with small random values,

- sample init: weight vectors are initialised with random samples drawn from the input data set.

2.2.3 Training

In each training step one sample vector $\mathbf{x}$ from the input data set is chosen (in order of appearance or randomly) and a similarity measure is calculated between it and all the weight vectors of the map's neurons. The Best-Matching Unit (BMU), denoted as c, is the unit whose weight vector possesses greatest similarity with respect to the input sample $\mathbf{x}$. The distance measure used to define the similarity is typically a Euclidean distance; formally the BMU is the neuron for which

$$\|\mathbf{x} - \mathbf{m_c}\| = \min_i \{\|\mathbf{x} - \mathbf{m_i}\|\} \tag{2.1}$$

holds, where $\|\cdot\|$ is the chosen distance measure.

Having found the BMU the weight vectors of the SOM are updated as follows: The weight vectors of the BMU and its topological neighbors are moved closer to the input vector from the input space. This adaptation procedure stretches the BMU and its neighbors towards the presented sample vector. For an illustration see figure 2.3. The

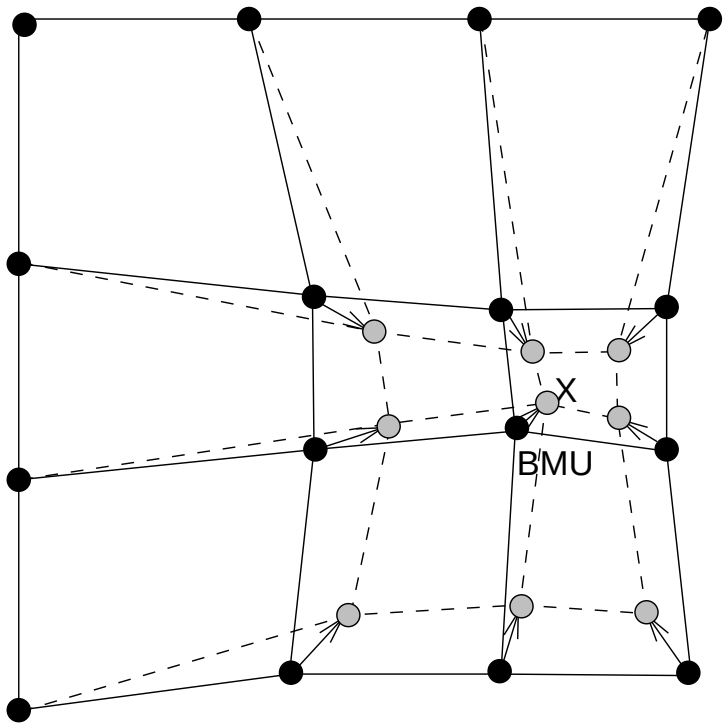


Figure 2.3: Updating the BMU and its neighbors towards the input sample [10]

update rule for changing the respective weight vectors of unit i of the SOM is:

$$\mathbf{m_i}(t+1) = \mathbf{m_i}(t) + h_{ci}(t)[\mathbf{x}(t) - \mathbf{m_i}(t)], \tag{2.2}$$

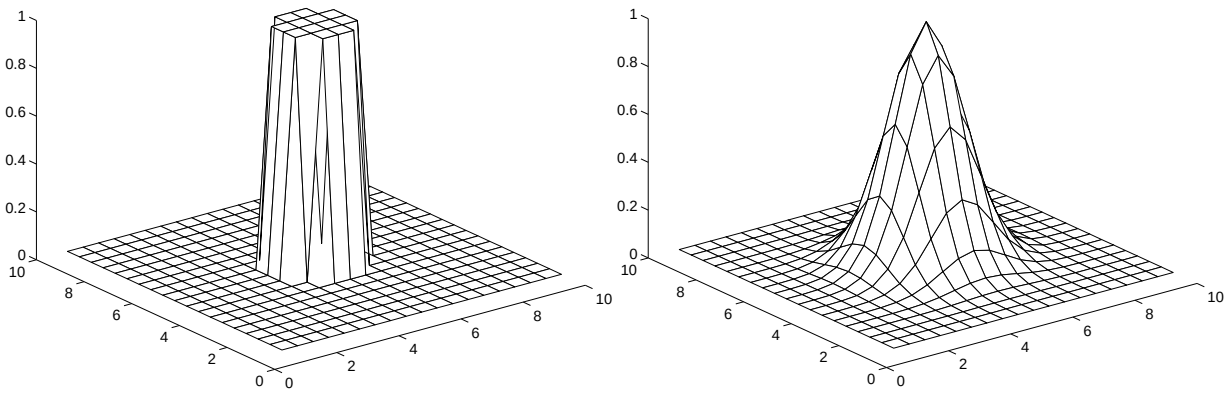
where t denotes the time step. $\mathbf{x}(t)$ is the input vector chosen from the input data set at time t and $h_{ci}(t)$ the neighborhood kernel around the winner unit c at time t. The neighborhood kernel is a non-increasing function of time and of the distance of unit i from the BMU c. Colloquially expressed, it defines the region of influence that the input sample has on the SOM. The kernel is formed of two parts: the neighborhood function $h(d,t)$ and the learning rate function $\alpha(t)$:

$$h_{ci}(t) = h(\|\mathbf{r_c} - \mathbf{r_i}\|, t)\alpha(t) \tag{2.3}$$

where $\mathbf{r_i}$ is the location of unit i on the map grid.

There are many possible types of neighborhood functions, the simplest and also most commonly-used of which are the following:

- Bubble: constant among the neighborhood of the winner unit and zero elsewhere (see figure 2.4 (a)),
- Gaussian: $\exp(-\frac{\|\mathbf{r_c}-\mathbf{r_i}\|^2}{2\sigma^2(t)})$ (see figure 2.4 (b)).



(a) (b)

Figure 2.4: Commonly used neighborhood functions [10]

Using the Gaussian neighborhood function yields slightly better results but is also heavier in computational terms. When the SOM is applied, the neighborhood radius is normally larger at first (resulting in fast adaptation of the map to the inputs) and is constantly decreased throughout the training process.

As well as the neighborhood also the learning rate $\alpha(t)$ is a function decreasing over time. Two common forms are:

- a linear decreasing function: $\alpha(t) = At$,

- a function inversely proportional to time: $\alpha(t) = \frac{A}{t+B}$,

with A, B chosen appropriately, respectively.

The training is usually carried out in two phases, the first of which features relatively large initial *alpha* and neighborhood radius values, whereas in the second phase both values are relatively small right from the beginning. This happens in pursuit of tuning the SOM to the same space as the input data and then fine-tuning the map. A more elaborate version of those 'tuning' and 'smoothing' phases can be seen in Section 2.3.2.

There are many variants to the basic SOM, such as neuron-specific learning-rates and neighborhood sizes or using a *Growing* SOM (GSOM), which will be explained in detail in Section 2.3 and has been used for the simulations explained in Chapter 3. These modifications target some of the inherent disadvantages of the basic SOM principle, such as the hard-coded, unflexible map size and enable the SOM to better represent the topology of the input data set without losing its advantages en route, most important of which is the instant comprehensibility through visualisation of the input data.

2.2.4 Important properties

As detailed in the last chapter, the SOM algorithm itself is very simple, but desirable mathematical proofs of its properties are still outstanding. However, in most practical applications good results can be obtained, one of the more exciting examples of which is [5], where about 6.8 million patent abstracts were preprocessed, represented by 500-dimensional vectors and then clustered using a SOM algorithm. SOM properties relevant to the given task are listed below.

Voronoi regions

The SOM partitions the input space into convex Voronoi regions, with each neuron being responsible (in analogy to the biological case) for one of these regions. The Voronoi region of a neuron i is the union over all the vectors $\mathbf{x}$ to which it is closest:

$$V_i = \{\mathbf{x} | \|\mathbf{m_i} - \mathbf{x}\| < \|\mathbf{m_j} - \mathbf{x}\|, i \neq j\}. \tag{2.4}$$

The reference vector of the respective Voronoi region is placed according to the local expectation of the data weighted by the neighborhood kernel:

$$\mathbf{m} = \frac{\int h_{ci} p(\mathbf{x}) \mathbf{x} \mathbf{dx}}{\int h_{ci} p(\mathbf{x}) \mathbf{dx}} \tag{2.5}$$

Quantisation and projection

In searching for good reference vectors and ordering them on a regular grid at the same time, the SOM combines the properties of data projection and vector quantisation techniques. The grid can be thought of as a 2-dimensional elastic network following the

original data's distribution. However, the SOM does *not* try to preserve the distances directly but instead focuses on representing the topology of the input data which makes it inherently useful for visualisation of large data sets.

There is an important tradeoff to be made between the two competing goals of (exact) quantisation and topology preservation; this can be controlled by setting the radius of the neighborhood kernel appropriately. Obviously, the SOM reduces to a plain vector quantisation algorithm when the neighborhood radius is set to zero.

Errors in data / missing data

Naturally, even the SOM cannot be prevented from suffering through incomplete or otherwise faulty data fed into it. However, outliers in the input data can be easily detected as their distance from other vectors in the same unit that it was mapped to is large. In practice, we often have to deal with data that has missing values, represented by vectors with missing components. This problem can be overcome by leaving out the missing component from the distance calculation during the training and updating process; if there is still a sufficient number of complete vectors to train the map with, the missing values can even be filled in with e.g. mean values from the unit it got mapped to. Obviously, 'Errors in data' does not refer to errors introduced during the preprocessing; if the input data is somehow biased or has other systematic or inherent errors, those cannot be detected and/or might simply result in bad maps.

2.2.5 Visualisation

There are a variety of different visualisation techniques once the SOM has been computed, two of which are relevant to the current problem and therefore mentioned here.

Vector component planes

The reference vectors of the SOM can be visualised via the component plane representation [3]. The computed SOM can be thought of as multi-tiered with the components of the vectors describing horizontal layers themselves and the reference vectors being orthogonal to these layers. In this planar, sliced representation it is easy to (a) see the distribution of the component values, and (b) recognise correlations between components. This technique might be used in a later stage of the current solution to the task from Section 1.3.

Clusters

The main techniques used for evaluating the results in Chapter 4 were to (a) visualise the computed map and manually look for distinguishable clusters and (b) introduce a quality measure as a further indicator of the map's grade. Clusters are groups of vectors which

are close to each other, relative to their distance to other vectors on the map. Therefore, there are different maps (through different criteria being visualised) to be shown and the ones to be used here are either the distance map or the hits map. Examples for types of maps can be found in Section 2.3.3 and their manual evaluation in Chapter 4.

2.3 GSOM

The characteristic feature distinguishing neural maps from other neural network paradigms and other regular vector quantisers is the preservation of neighborhoods. This desirable feature obviously depends on the choice of the output topology: the proper dimensionality of the output space is usually not known a-priori, but yet has to be specified prior to learning in the SOM algorithm. This knot has to be cut to be able to optimise the neighborhood preservation.

Of course, one could easily compute structures in a brute-force approach, starting from different map sizes and evaluating the degree of neighborhood preservation to choose the best map possible. Yet, this strategy is computationally heavy and does not necessarily yield the map with the best possible neighborhood preservation. Therefore, in [2] a new algorithm was proposed which extends the SOM in a very straightforward way. A more advanced description can be found in [8] or in [1].

Since the GSOM is an extension to the SOM, the latter's basic principles (see Section 2.2.1 and 2.2.4) also hold for the GSOM. For the sake of clarity this section is structured similarly to the SOM section.

New variables will be introduced:

- H_{err}: accumulated error throughout the map,
- *Spread Factor*: allows user to control growth,
- *Growth Threshold*: derived from Spread Factor, see Equation 2.6.

2.3.1 Structure and Initialisation

The basic structural assumptions of the GSOM are the same as those of the SOM, except that the initial grid size of the output space usually lies in the range of *two* to *four* neurons, depending on topology (rectangular, hexagonal). See Section 2.2.1 for more details regarding the choice of topology.

However, the initialisation is usually performed randomly by selecting values from the input vector's value range and, since every initial node is a boundary node, no restrictions are imposed on the direction of growth. To determine where growth occurs (i.e. where nodes are added) a new variable H_{err} is initialised to 0 and will keep track of the highest accumulated error value in the network. Furthermore, a *spread factor* (SF) value has to be specified by the user enabling him to control the growth of the SOM.

The mentioned *Growth Threshold* (GT) derives from the SF as in Equation 2.6 (where D stands for the data dimensionality) and is used internally as a threshold value for initiating node generation: a high GT means less spread and a low GT leads to a larger, more spread-out map. Note that the outer boundaries of the spread factor (0 and 1) can not be included since $ln(0) = undefined$, and $ln(1) = 0$ renders the growth threshold useless since the computed error of the map will exceed it in every step and therefore grow continuously.

$$GT = -D \times ln(SF) \tag{2.6}$$

2.3.2 Phases

Growing phase

As with the SOM, the set of neuron vectors W_i in the GSOM can be considered as a vector quantisation of the input space so that each neuron i is used to represent a Voronoi region V_i (see Section 2.2.4). A winner neuron is found as per the algorithm listed below. If a neuron contributes significantly to the total error of the network then its Voronoi region is said to be underrepresented by the assigned neuron. Hence, a new neuron is generated in the immediate neighborhood to achieve a better representation of the region, or, if growth is not possible, the error gets distributed to the neighboring neurons, raising their error and therefore their growth probability.

The (simplified) GSOM algorithm for the growing phase presents as follows:

- Present input to the network.
- Determine the winner neuron i, using the chosen distance measure (e.g. Euclidean).
- Adapt weight vectors of winner and its respective neighborhood.
- Increase the error value of the winner.
- If total error of neuron is larger than the growth threshold: grow, if i is a boundary neuron; or distribute weights to neighbors if i is a nonboundary neuron.
- Repeat these steps until all inputs have been presented.

Smoothing phases

The smoothing phases occur after the new neuron growing in the preceding growing phase. The growing phase stops when new neuron growth gets saturated, i.e. if the frequency of new neuron additions falls below a certain threshold. Once the neuron growing phase is complete, the weight adaptation is continued at a lower adaptation rate. No new neurons are added during this phase, whose purpose it is to smooth out any existing quantisation error, particularly in the neurons grown in the later stages of the growing phase.

During the smoothing phases, inputs to the network are the same as those of the growing phase. The starting learning rate (LR) in this phase is smaller than in the growing phase since the weight values should not fluctuate too much without converging. The input data vectors are repeatedly presented to the network until convergence is achieved or a certain specified number of inputs have been presented. The smoothing phase is stopped if the error values of the neurons in the map fall below a threshold. Multiple smoothing phases with different (consecutively decreasing) LR and neighborhood sizes can be applied.

2.3.3 Maps

Average map The left map in Figure 2.5 shows similarities between neighboring weight vectors. For each weight vector the distance to each of the neighboring weight vectors is calculated, those values are averaged and a color from a black-and-white palette is assigned according to the determined value. White or light grey colors show similarities (low distances) whereas black and darker colors show dissimilarities (high distances). If the distances are not averaged, but accumulated instead, you get to the distance map, which will be described now.

Distance map The right map in Figure 2.5 exemplifies a distance map, where the distances that each neuron has to its neighboring neurons are accumulated and depicted with a color palette ranging from blue (low distance) over green (medium distance) towards red/brown (high distance). This map can help in finding borders between existing clusters, since cluster borders are small regions where distances are relatively high compared to the rest of the map or the inside of a single cluster.

Error map The error map, to be seen on the left of Figure 2.6, shows the accumulated error for every neuron at the current stage of training. A low or zero error is illustrated as a violet or dark blue color, whereas the error raises over red/brown to greenish colors for neurons with the highest error. Since the map grows from a small initialisation stage to larger sizes, the error is expected to propagate towards the borders of the map, where neurons with highest error cause map growth to better represent that particular region of the input space.

Hits map This map, shown on the right of Figure 2.6, is the type of map that will mainly be worked with when running simulations. It shows the number of inputs which were mapped to the individual neurons during the GSOM training phase. The color palette ranges from blue (no mappings) over shades of green to red (high number of mappings). In the depicted case, it is to be expected to have a high number of mappings in the middle cluster (neurons 1, 3, 5, 6, 7, 15), a medium number assigned to neuron 62 (bottom left) and a cluster with less mappings (neurons 60 and 64), with the rest of the input references scattered throughout the remaining neurons.

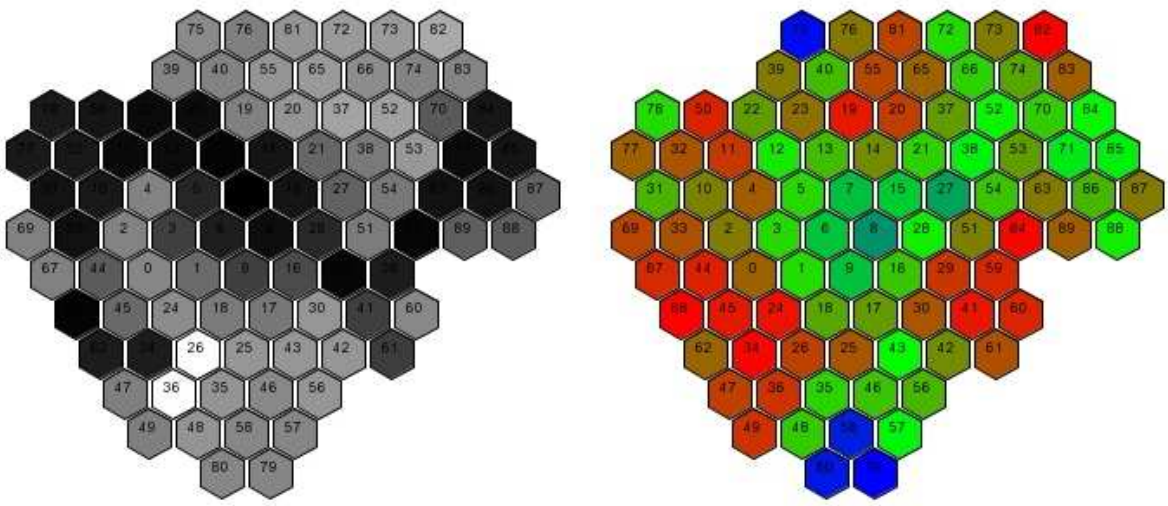


Figure 2.5: Average map and Distance map

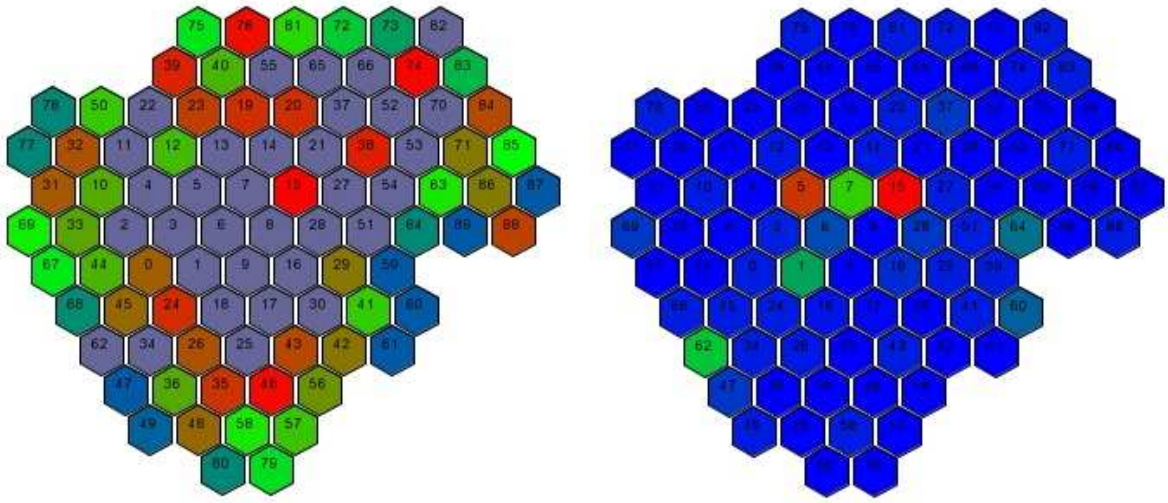


Figure 2.6: Error map and Hits map

2.4 GSOM implementation

A sophisticated graphical user interface to the GSOM algorithm has been provided and the essential user configuration parts can be found here. Every aspect that has been mentioned theoretically beforehand is implemented in JAVA; therefore rendering the operation intuitively. Furthermore, the implementation details are of no interest in the scope of this report, so they are left out and only some schematic screen shots are shown below.

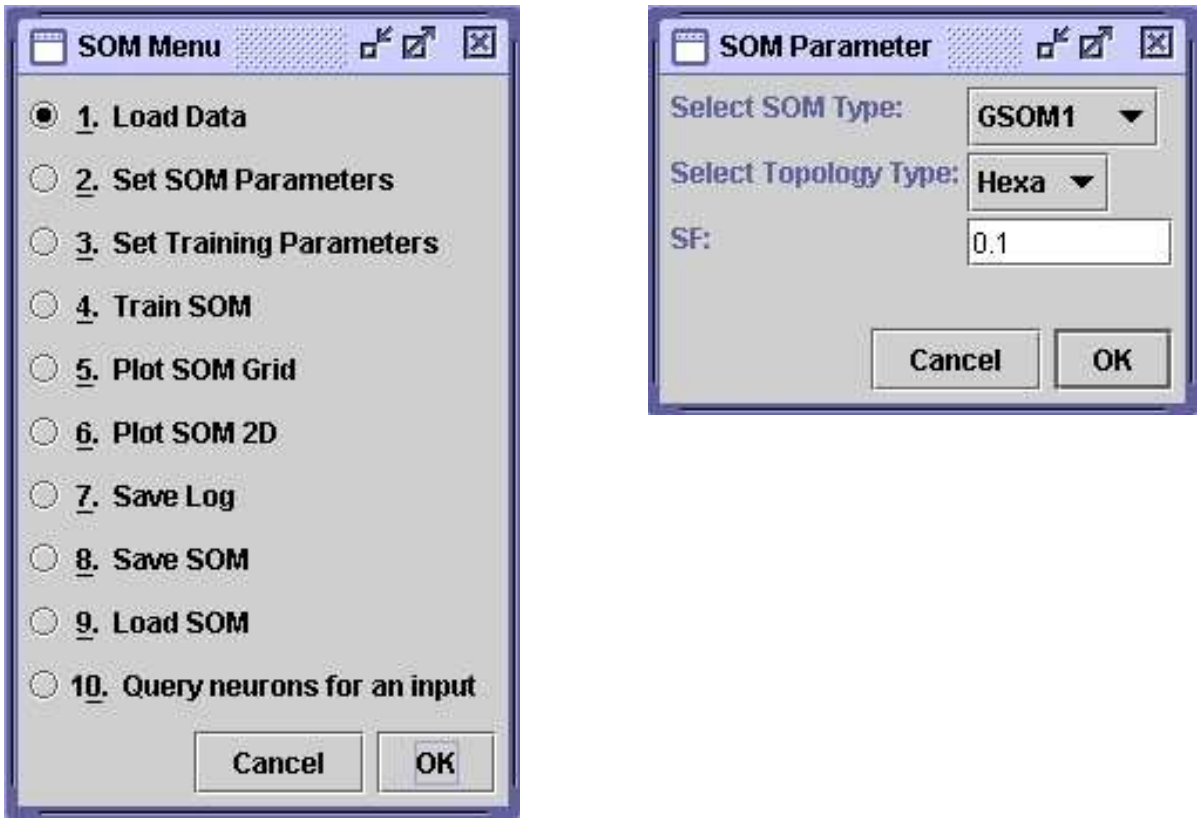


Figure 2.7: Basic menu choices of GSOMpak JAVA implementation

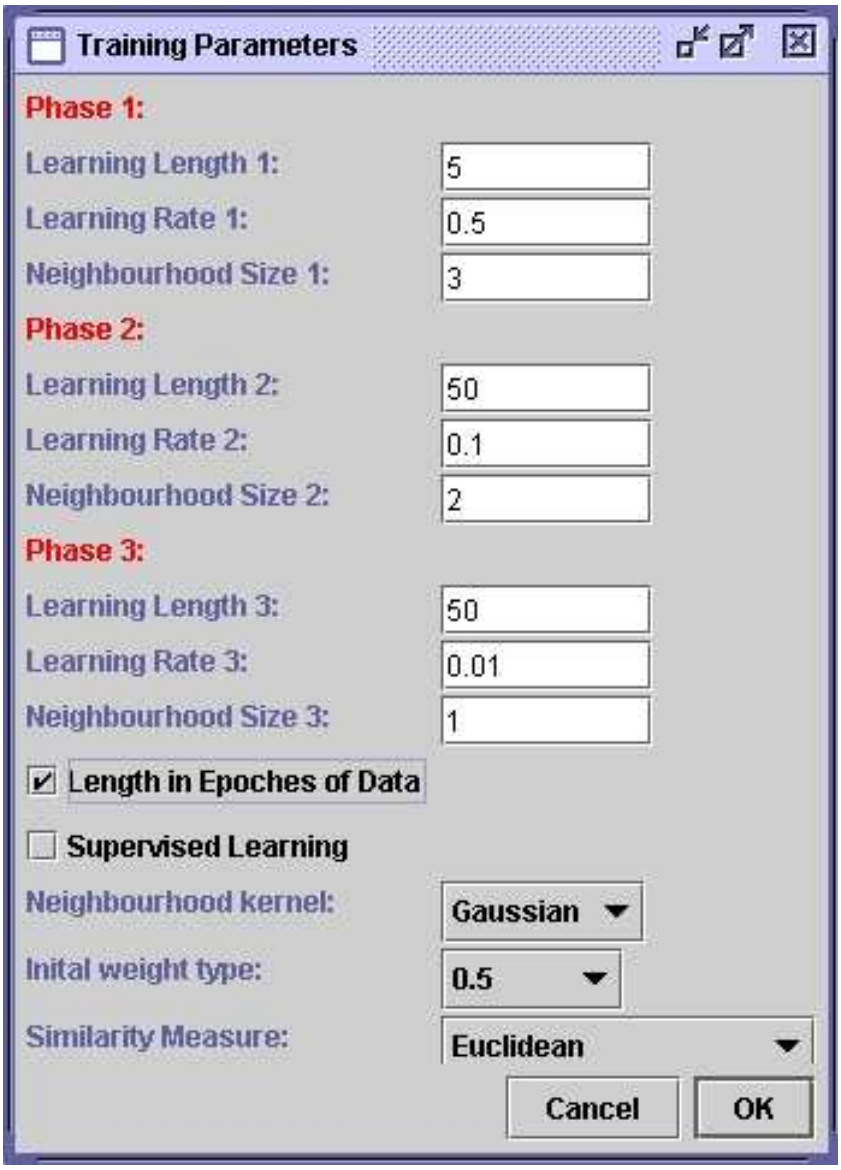


Figure 2.8: Training menu options of GSOMpak JAVA implementation

Chapter 3

Simulation and Evaluation Framework

Having explained the basic techniques in Chapter 2, this chapter deals with the generated simulation framework and introduces a quality measure for evaluating the simulation results. However, since preprocessing of data is inherently important, it will be explained first.

3.1 Data preprocessing

The first attempt at preprocessing was done by a fellow research team member which later proved to be incomplete and is to be supplemented in Section 3.1.2. The newly preprocessed data which was used as input to the data mining process is schematically depicted in Table 3.1.

	C1	C2	...	C808	REF
1	1.00E-01	2.21E-02	...	1	9628
2	0.00E-00	2.21E-02	...	0	9496
3	1.05E-01	2.34E-02	...	0	9336
4	1.36E-01	2.26E-02	...	1	9148
5	1.00E-01	2.50E-02	...	1	9480
⋮	⋮	⋮	⋮	⋮	⋮
15980	2.69E-01	2.24E-02	...	1	9728

Table 3.1: Data dimensionality and layout *after* preprocessing

3.1.1 Original preprocessing

Pruning

As can be seen from table 1.1 (original data) there are certain types of data fields that are irrelevant to the data mining process, such as dates and times of data acquisition. There are also constant fields which therefore bear no distinguishing information at all. These can safely be pruned to reduce the computational load afterwards without having influence on the data mining result.

Expansion of categorical data

Categorical data has to be expanded to suit the assumption of equal distances among the data. For each column of categorical values, the number of different attribute values n is calculated; subsequently, this column is replaced by n columns (one for each category) which are filled with *1's* where the column's category matches the original category and with *0's* otherwise. This preprocessing step can dramatically change the dimensions of the input data and might also lead to sparsely populated input tables. See table 3.2 for a comprehensive example.

Category		A	B	C	D	E
A		1	0	0	0	0
B		0	1	0	0	0
C	⇒	0	0	1	0	0
A		1	0	0	0	0
D		0	0	0	1	0
E		0	0	0	0	1

Table 3.2: Expansion of categorical data

3.1.2 Own preprocessing

After several months of simulations an important observation caused to believe that the original order of steps might be incomplete: even with several well-chosen parameter sets, the GSOM algorithm did not converge towards a satisfactory separation of the two input classes on the map. Since the GSOM itself as well as the parameter sets are less likely to generate this behaviour, the preprocessing seemed to be the cause, therefore additional preprocessing steps had to be undertaken in order to improve on simulation results. To be able to assess the difference in results due to changes in preprocessing, a quality measure has also been developed.

Elimination of outliers

Among the original dataset in Table 1.1 it can be seen that there are certain outliers in the data which are most probably caused by incorrect measurements from the production lines (e.g. faulty meters or errors during transmission); which manifest by being several orders of magnitude different from the remaining majority of the data. Assuming that the normalisation is carried out in the usual way, even a single outlier is sufficient to skew the ensuing data mining process. A more intuitive explanation can be obtained from Figures 3.1 and 3.2, where the immediate effect of the outliers is obvious, scaling the majority of values to nearly 0 and the few outliers to 1. This renders the data mining process nearly useless since floating point arithmetics' accuracy is limited.

Finding the outliers was done manually as follows:

- For every numerical attribute:
  1. Sort input vectors by current attribute
  2. Calculate attribute median
  3. Calculate attribute mean
  4. Compare median to mean
  5. For large differences:[1]
     (a) Check both ends of sorted list for outliers
     (b) Delete corresponding input vectors
     (c) Repeat from Step 1
  6. For small differences: Go to next attribute.

It shall be noted that the distribution of "good" and "bad" input vector references did not undergo a major change after outliers were eliminated; the percentage of "good" product references changed from 85.92% to 86.04% whereas the percentage of bad product references was reduced by the respective amount.

Normalisation

Every data attribute was scaled to $[0, 1]$ by using Equation 3.1, where x_{min} and x_{max} denote the minimum and maximum value of the attribute, respectively.

$$x_{new} = \frac{x - x_{min}}{x_{max} - x_{min}} \tag{3.1}$$

[1] For each of the numerical attributes under consideration in this context, one or more outliers could be found that were at least three orders of magnitude different from the rest of the attribute values. This method works in this special case because of the attributes' having a distribution similar to the one shown in Figure 3.1.

Figure 3.1: Data distribution before normalisation including outliers

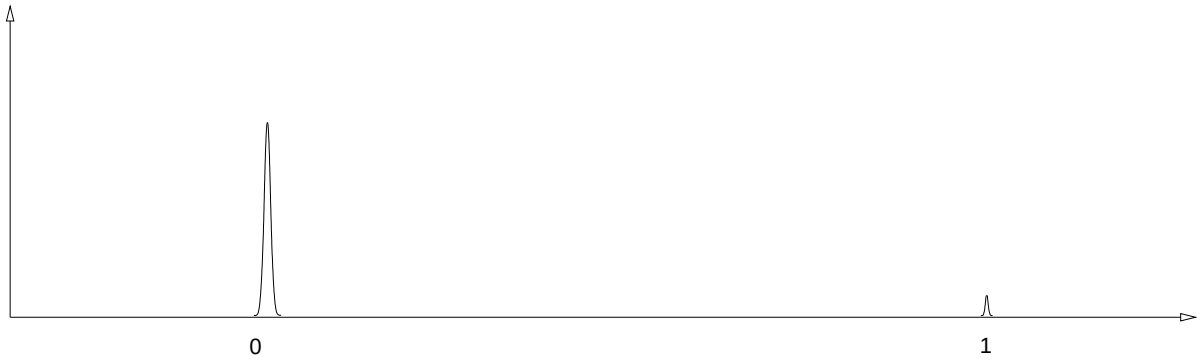


Figure 3.2: Data distribution after normalisation including outliers

Expansion of categorical data

Expansion of categorical data was performed in the same manner as in the original preprocessing chain (Table 3.2).

3.2 A quality measure

In order to compare the results of running simulations with different parameters on the same dataset, or with the same parameters on differently preprocessed data, this section will introduce a quality measure which takes the complete map into account and can be generated automatically as an objective, i.e. non-human-biased, quantifier.

Assumptions

As mentioned in Section 1.3 the given task is to invoke the GSOM algorithm to produce an automatic classification (clustering) of high-dimensional input data. The desired clustering is known in advance from the data distribution and would be a binary clustering at best, i.e. separating the "good" from the "bad" products on the GSOM. The data available for developing the benchmarking method are (a) the original data's statisti-

cal values and its distribution between "good" and "bad", and (b) the data from the generated map, i.e. the distribution of input vector references among the map's neurons.

Desired benchmark properties

The benchmark (named 'clustering quality') should yield a number between 0 and 1 as a quality measure of the input vector separation among the neurons. 0 means no difference in data distribution, i.e. no clustering at all. 1 means a high probability of good clustering, albeit visual map exploration is still necessary. The developed benchmark is supposed to aid the user in evaluating the result, therefore a value of 1 does not guarantee good clustering but can instead be seen as an indicator of potentially good clustering. A simple example for a high benchmark rating in combination with an obviously bad map could be where a benchmark value of 1 exists for an overtrained, non-clustered map with as many neurons as input vectors and every input vector being assigned to exactly one neuron. The algorithm's block diagram is depicted in Figure 3.3.

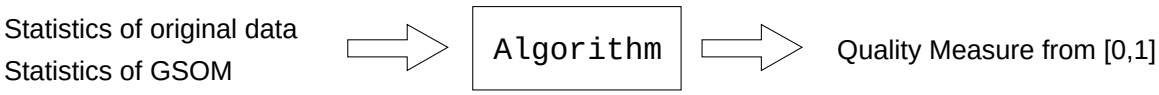


Figure 3.3: Quality-measuring algorithm

A good separation of input vectors is achieved if every neuron contains only input vectors from one class (i.e. "good" or "bad") or no input vectors at all, in which case the benchmark should yield 1. Bad separation (or *no* separation) should be benchmarked with a 0.

Conventions

Variable names referring to the complete input dataset (constants):

- N – total number of inputs
- G – total number of good products
- B – total number of bad products
- $\frac{G}{N}$ – percentage of good products in total number of inputs
- $\frac{B}{N}$ – percentage of bad products in total number of inputs

And similarly for inputs mapped to respective neurons:

- n_i – number of inputs mapped to neuron i
- g_i – number of good products in neuron i
- b_i – number of bad products in neuron i
- $\frac{g_i}{n_i}$ – percentage of good products in inputs mapped to neuron i
- $\frac{b_i}{n_i}$ – percentage of bad products in inputs mapped to neuron i

Calculation

Finally, the clustering quality value CQ is computed as follows:

$$CQ = \sum_i \max\left\{\frac{\frac{g_i}{n_i} - \frac{G}{N}}{1 - \frac{G}{N}} * \frac{n_i}{N}, 0\right\} + \sum_i \max\left\{\frac{\frac{b_i}{n_i} - \frac{B}{N}}{1 - \frac{B}{N}} * \frac{n_i}{N}, 0\right\} \tag{3.2}$$

From Equation 3.2 it can easily be seen that the maximum CQ value of 1 will be achieved if good and bad product references are mapped to separate clusters. On the other hand, the minimum benchmark value of 0 will be returned if the data partitioning on the map is the same as that amongst the input dataset. Colloquially expressed, CQ gives a measure of how far the clustering on the map deviates from the original data distribution.

Testing

Two simple test datasets were generated to show the correctness of the developed clustering quality measure (and, simultaneously, the functionality of the used GSOM implementation as well). As the interests lie in assessing *binary* clustering quality, two samples which represent the original dataset's two different classes were taken from the normalised dataset, slightly shortened (less attributes), and multiplied by a number of 50 and 500, respectively. The resulting test datasets have dimensions of 59x100 and 59x1000 and, due to the synthetic generation, consist of two easily distinguishable classes. These two datasets were then fed into the GSOM and showed the correctness of CQ. The generated hits maps can be found in Figure 3.4. Both maps were computed with the parameter set [SF=0.5, GP=3, SP1=SP2=0] and show the desired CQ value of 1 since both classes are perfectly separated on the map (one class per red cluster).

3.3 Dealing with the computational load

The GSOM algorithm tends to be computationally heavy with the number of inputs growing. Preliminary computation times on up-to-date workstations using the complete, preprocessed dataset ranged from one hour up to two weeks of continuous simulation time. This section deals with different approaches to reduce or at least adjust the computational burden accordingly.

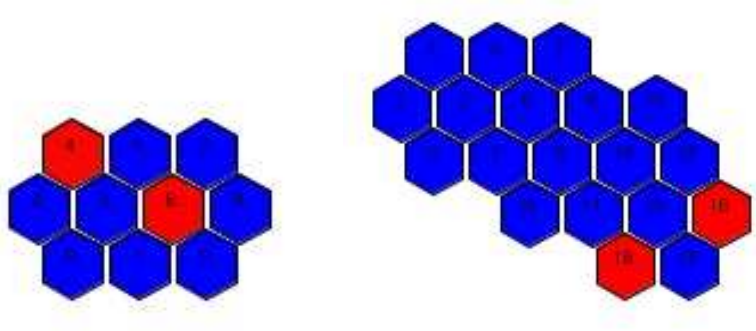


Figure 3.4: Generated maps with test dataset, left: 59x100, right: 59x1000

3.3.1 Adjustable parameters

Table 3.3 depicts the richness of adjustable parameters whose effects were evaluated and a decision was made on which of them to include during the actual simulations. [2] [3] [4] Important and used parameters are described here:

Number of inputs: As most important factor determining the result of the GSOM algorithm, the number and dimension of inputs can be chosen arbitrarily, although data has to be normalised beforehand. Special care has to be taken regarding the size of the input data since the running time of the GSOM algorithm grows exponentially with the size of the input. Simulations were run on (a) sampled 808x1638-, (b) 808x16380-, and (c) 808x15980-dimensional datasets.

Spread factor: This parameter determines the final size of the map by providing the user with a convenient, high-level way to influence the map's internal growth accordingly [8]. A larger spread factor means a more spread-out and therefore larger final GSOM. It was varied from 0.1 to 0.995 during simulations and its range is in $]0, 1[$. It is also of great influence on the length of the training period.

Topology: Either hexagonal or rectangular, as explained in Chapter 2. A hexagonal grid is used by default and stayed unaltered as it also states a balanced tradeoff between neighborhood size and quantisation of the input space.

Kernel: The actual implementation provided the Gaussian kernel as well as the 'bubble' kernel. Again, see Chapter 2 for details. The Gaussian kernel is used in most of the scientific literature and therefore went unaltered.

Similarity measure: Inside the GSOM algorithm, this measure is used in the calculation of the Best-Matching Unit. The distance between the prototype and every

[2] 'tested': the effects of the respective parameter were evaluated in preliminary experiments
[3] 'adapted': the parameter has been chosen for a systematic adaptation in later simulations
[4] default options are denoted in *emphasised* font

weight vector of the map is calculated using the chosen similarity measure and the unit with the smallest distance is denoted as BMU. As the Euclidean distance measure is best suited to most of the GSOM tasks, it was used here. Another implemented method was Pearson's correlation. Depending on the complexity of the similarity measure, its choice can also have an impact on computation time.

Options per algorithm phase: learning length, learning rate, neighborhood size

- learning length:
  During the course of the GSOM calculation, this parameter sets how many times the input data is fed into the algorithm. As an integer multiplier, it determines the final size of the map and therefore also the running time of the algorithm. It was varied from 1 to 100 during preliminary experiments and from 1 to 10 during the actual simulations. However, in the current task the second and third phases had no actual effect on mappings and therefore only increased computation time dramatically.
- learning rate:
  This parameter is used while updating the winner's and its neighbors' weight vectors and can be used to control the speed of the map's adaptation to inputs. It went unchanged from its defaults of 0.5 (1st phase), 0.1 (2nd phase), 0.01 (3rd phase). A large value is desired at first to keep the map flexible towards inputs, whereas smaller values are used during the smoothing phases.
- neighborhood size:
  Controls how many layers of neighboring neurons are affected by a weight change of the winner unit. Decreases during the course of the algorithm from 3 (1st phase) to 1 (3rd phase).

3.3.2 Sampling method

A straightforward approach to deal with the exponential behavior in terms of computation time is to let the GSOM work on sampled data only. Two ways of sampling are proposed below and their implementations were integrated into the JAVA user interface. It should be noted that the actual GSOM algorithm remains unchanged since the sampling reads the input datafile and generates the respective output datafile(s) to be fed into the program later on.

The first approach consists of drawing a sensible number of *randomly chosen inputs* from the full dataset and save them into a new input file. The second approach which also reduces algorithm running time is to *partition* the dataset, but instead of taking the data blockwise and output them into new files, it is chosen randomly from the input data and deleted afterwards so that every input vector will show up exactly once among the union of generated subsets, i.e. the subsets are disjoint. A comparison of both methods can be found in Figure 3.5 and the respective JAVA source code is to be found in Appendix A.

Parameter	**tested**	**adapted**	**options**
number of inputs	+	+	integer; full or sampled dataset (see text)
spread factor	+	+	$\in]0,1[$
topology	-	-	*hexagonal*, rectangular
kernel	-	-	*gaussian*, bubble
similarity measure	-	-	*euclidean*, pearson's correlation
options per phase			
1 - learning length	+	+	(small) integer * number of inputs
1 - learning rate	+	-	*0.5*
1 - neighborhood size	-	-	*3*
2 - learning length	+	+	(small) integer * number of inputs
2 - learning rate	+	-	*0.1*
2 - neighborhood size	-	-	*2*
3 - learning length	+	+	(small) integer * number of inputs
3 - learning rate	-	-	*0.01*
3 - neighborhood size	-	-	*1*

Table 3.3: Adjustable parameters

3.4 Simulation setup

The ideas in Section 3.4.1 led to the simulation plan stated in Sections 3.4.2 and 3.4.3. Certain time and computing equipment restrictions had to be taken into account and are mentioned, if necessary. The terms 'old data' and 'new data' refer to the different preprocessing methods explained in sections 3.1.1 and 3.1.2, respectively. Table 3.4 gives a condensed list of simulation plans; results can be found in chapter 4. The basic ideas are described first.

3.4.1 Ideas

In [8] three entangled ideas are proposed on how to proceed with the simulations after having preprocessed the data. An additional, fourth, idea incorporates the generated quality measure.

1. Varying the spread factor

Certainly the most basic approach to adapt the map to the input data is to vary the map's size by varying the spread factor appropriately. Since SF takes values from the interval]0,1[(see Equation 2.6) it is useful to start with small spread factors and increase them

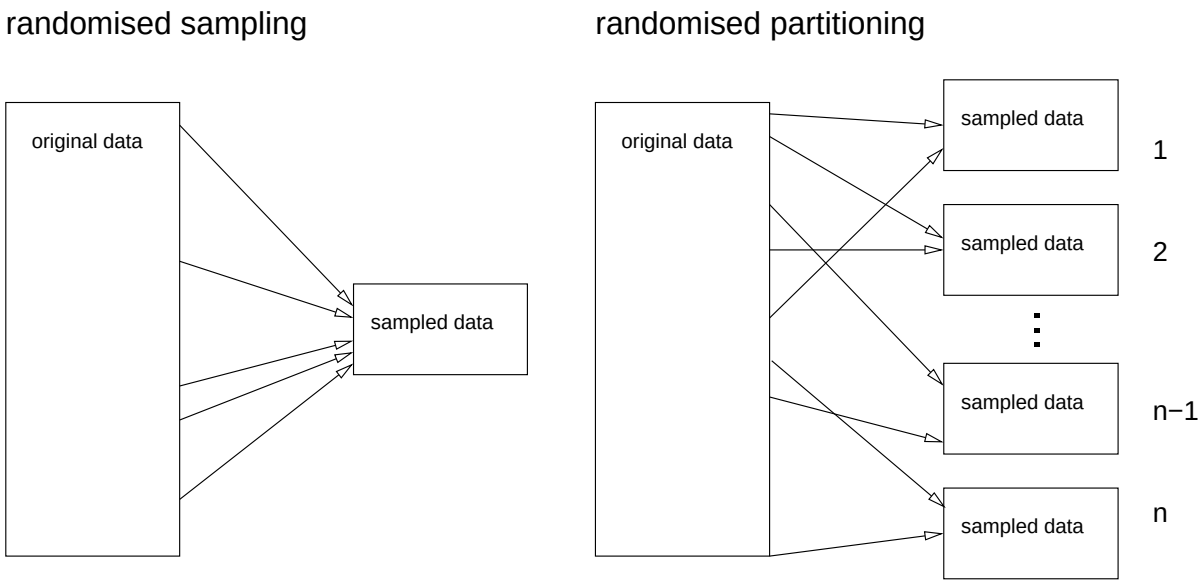


Figure 3.5: Randomised Sampling vs. Partitioning

continuously in discrete steps. The user should then be able to examine the generated maps and check for existing clusters of data. This approach was pursued systematically, among others.

2. Leaving out attributes

It may be necessary and of great insight to study the effect of removing several attributes from the input dataset and check its impact on the generated maps. This will generate and/or confirm knowledge about possible non-contributing attributes. Preliminary work has already been done in pursuing this approach; figures 3.6 and 3.7 show promising discriminatory distances among several attributes whereas numerous other attributes seem to be less distinguishing. Simultaneously, the effect of removing outliers can be recognised along the y-axis, which shows the difference between the average values of 'good' and 'bad' classes; due to a few outliers, this difference had been skewed without preprocessing.

Unfortunately, the manufacturer's requirement was not to leave out any attributes from the data mining process, hence this approach has not been pursued any further.

3. Automating the process

The process of systematically growing maps with varying spread factors is best suited to being automated for saving computation time, using the dimensionality-independent spread factor to name and compare different maps. However, since most of the maps took several hours and even days to compute, the time for evaluating these maps manually and starting new simulations with different parameters could be neglected, especially

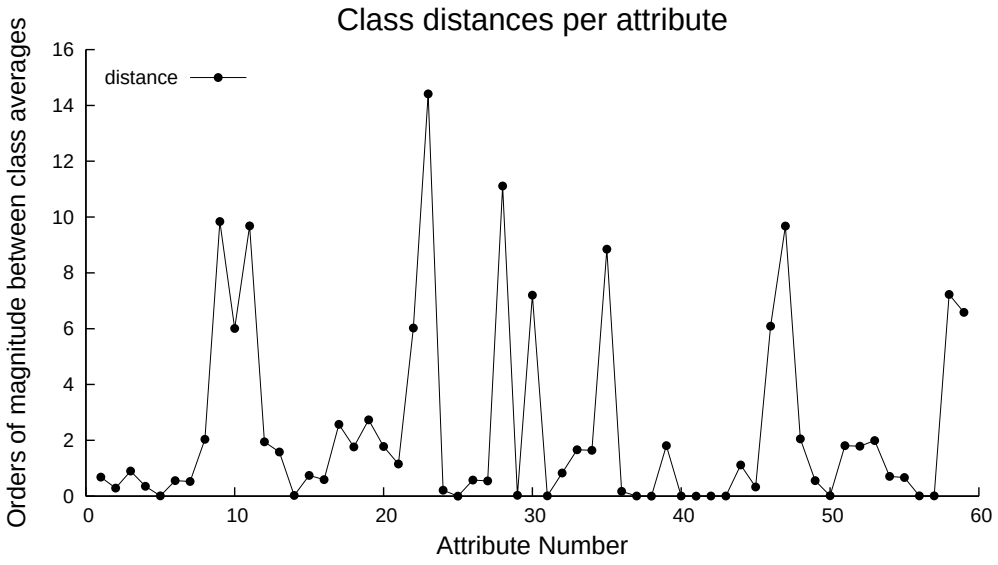


Figure 3.6: Distances between logarithmic attribute averages, *before* eliminating outliers

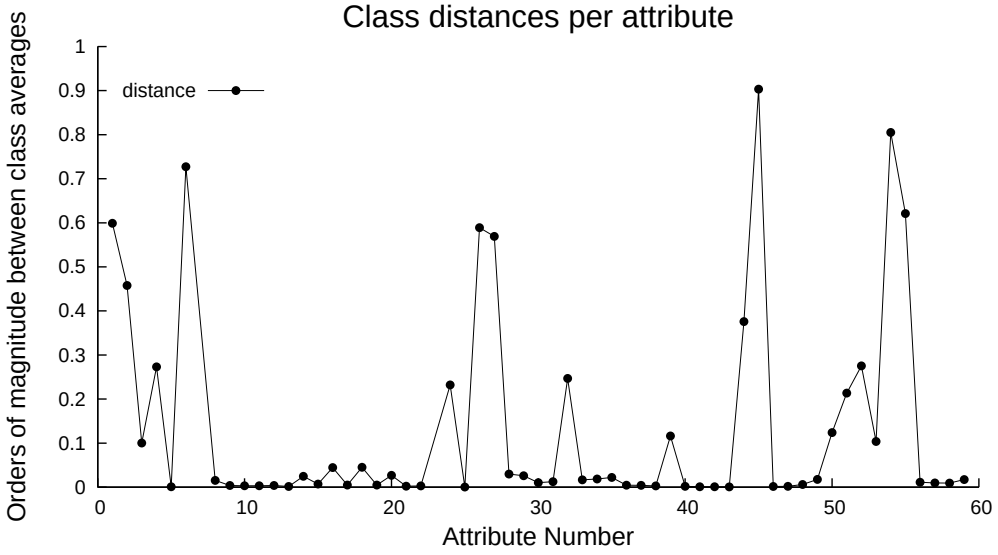


Figure 3.7: Distances between logarithmic attribute averages, *after* eliminating outliers

since remote-control technologies were employed to minimise time losses. Nevertheless, this can be a future improvement.

4. Including the quality measure

To easily assess the objective quality of the generated maps, the benchmark formula introduced in section 3.3 was applied to the maps and indicates the quality of the clustering compared to the known original data distribution as a single percentage.

3.4.2 Old data

Sampled data The original data was sampled using the implemented methods, a sample size of 10% was found to be a good trade-off between reducing computation time and still having a sufficient amount of data left to generate meaningful maps. Preliminary experiments resulted in an appropriate spread factor of 0.7 for the GSOM algorithm. The data was partitioned into ten disjoint sets and five of these sets were randomly chosen to be used for applying the GSOM algorithm to them. However, the quality measure has not been applied here, reasons for which will become obvious from the results.

Complete data Simulations with the same parameters as with the sampled data were run on the complete dataset. Additional simulations were run with constant spread factors of 0.8 and 0.9 and variations in the learning length during the growing phase being in the range from 1 to 10. The quality measure was applied to the latter simulations.

3.4.3 New data

Sampled data Time restrictions did not permit to run simulations on samples taken from the re-preprocessed data. However, the simulation results from section 4.1.1 show that working on sampled data is probably not justified at all.

Complete data As with the sampling approach, prohibitive time restrictions caused not to run simulations in the same, extensive manner. For comparative results spread factors of 0.8 and 0.9 were chosen and the learning length varied in analogy to the later simulations run on the old data. However, to assess the influence of different spread factors on the map growth, additional simulations were run with SF varying from 0.1 to 0.9.

3.4.4 Tabular simulation schedule

Table 3.4 serves as an overview about the simulations planned and run. The asterisk (*) denotes one simulation that could not be finished after it had been running for more

Dataset	Part	SF	GP, LL	SP1, LL	SP2, LL	Dimensionality
OLD	sampled (5 sets)	0.7	1-10, 100	0	0	808x1638
			5	10	[0,10]	
			10	10	[0,10]	
	complete	0.7	1-10, 100*	0	0	808x16380
			5	10	[0,10]	
			10	10	[0,10]	
		0.8	1-10	0	0	
		0.9	1-10	0	0	
NEW	complete	0.8	1-10	0	0	808x15980
		0.9	1-10	0	0	
		0.1-0.9	1	0	0	

Table 3.4: Performed simulation schedule

than two weeks.

Chapter 4

Simulation Results

This chapter gives explanations of obtained simulation results. The results themselves are graphed in condensed form at the end of this chapter, whereas the underlying numerical data as well as additional maps can be found in the appendix.

4.1 Results for old data

Before the errors in data-preprocessing were discovered, simulations had already been run extensively. Their results might therefore be invalid, but, however, the generated results seem to be quite congruous to the simulations which were run after re-preprocessing the data. These results might also be useful for other researchers encountering similar issues. This section tries to explain those results.

4.1.1 Sampled data

The complete dataset was sampled into ten disjoint datasets, using the partitioning approach explained in Section 3.3.2. Five of these ten samples were randomly chosen and simulations were run according to the laid-out schedule. Simulation results on all of these sample sets were inside a small-deviation interval, therefore Table B.1 contains this data in averaged form.

As can be seen from the graphs in Figure 4.1 the computation time grows much faster than the map size, which seems to stagnate after several iterations. Another effect of extremely long growing phases is 'overlearning', which can clearly be seen from the last line in Table B.1: the training error declines quickly to zero after nine iterations and starts to rise steadily after some more iterations. On the opposite, the quantisation error QE declines constantly and reliably, which is natural since it is roughly inversely proportional to the map size; hence, larger map sizes allow the GSOM to better partition the input space on the map. From the tabular data it can also be seen that the smoothing phases do affect only TE and QE, but not the final map size; since they don't change input

vector mappings at all (which is what we are focused on), these phases were neglected in subsequent computations.

Two exemplary hits maps are depicted in Figure C.1 and show no apparent clustering at all, but instead exhibit a behaviour of aligning mappings regularly throughout the map grid.

4.1.2 Complete data

SF = 0.7, different GP, SP1, SP2

Results very similar to those for the sampled data above were obtained for the complete dataset; they are graphed in Figure 4.2. Additionally, it can be seen that the sampling approach would save a large amount of computation time. Unfortunately, the longest simulation (denoted with '*' in Table B.2) could not be finished due to administrative restrictions at the simulation laboratory. However, by that time it had already been running for 15 consecutive days, clarifying the inherent computational complexity of the GSOM algorithm

Unexpectedly, the training error TE exhibits an abnormal behaviour, which will manifest again in subsequent results; QE shows the expected decline with the number of growing phases rising.

To keep the resulting maps comparable to the ones depicted for sampled data, maps resulting from the same parameter set are shown in Figure C.2. Those apparently seem to be much better than the ones for sampled data, but closer examination of the data distribution will reveal that no clustering has been performed by the GSOM algorithm.

SF = 0.8, 0.9, GP = [1...10]

Having finished simulations with SF=0.7, and having established that varying the SP1 and SP2 lengths does not change the mapping of input vectors to map neurons, only SF and the number of GP were changed systematically; the obtained results are depicted in Figure 4.3. Numerical results can be found in Table B.3, from which it can also be seen that an increase in SF from 0.8 to 0.9 roughly doubles computation times and map sizes. QE behaves as expected, declining steadily, and is therefore not graphed; TE again shows non-converging behavior with both spread factors used.

The current parameter set also firstly features the quality measure CQ introduced in Chapter 3 and shows rising clustering quality towards higher SF and GP, (Figure 4.3, right). As explained in the definition of this quality measure, a higher obtained CQ does not guarantee a higher subjective clustering quality but can indicate which parameters to change towards achieving better results or, to put it another way, to compare maps generated with different parameters.

Four maps for the current parameter set can be found in Figures C.3 and C.4. The depicted maps seem to be of higher quality than the ones in the last section; they also

yield a higher CQ value, but they are still quite far away from showing perfect separation of the two classes.

4.2 Results for new data

After re-preprocessing the data with an additional step, simulations were run again with comparable parameters. An evaluation of the obtained results can be found below. Due to certain time limitations and negative experiences with the sampling approach, simulations were only run on the complete dataset.

4.2.1 Complete data

SF = 0.8, 0.9, GP = [1...10]

Again, the left graph of Figure 4.4 shows non-converging behaviour of TE throughout different parameter sets. On the other hand, CQ reliably rises with higher SF and towards longer GP, to be seen in the right graph of Figure 4.4. Four maps with the same parameters as in preceding sections have been depicted in Figures C.5 and C.6; direct comparison to the maps generated from the old dataset shows significant subjective improvements for those maps generated with low spread factors whereas the aforementioned 'regular alignment of inputs' on the grid can be seen on the right of Figure C.6.

SF = [0.1...0.9], GP = 1

Figure 4.5 shows the effects of leaving GP constant at a low value and varying SF from 0.1 to 0.9. TE as well as QE should be declining with larger SF, but TE seems to alternate whereas QE roughly behaves as expected. Similar unexpected results were obtained for CQ [right graph], which is alternating uncontrollably when it was expected to rise steadily.

The current parameter set is perfectly suitable to demonstrate the growth process of the GSOM with different spread factors; an example of the hits maps for SF = 0.1...0.9 is depicted in Figure C.7. The faster the map grows, the faster different input clusters become recognisable on the self-organising map.

4.2.2 Effects of preprocessing

From Figure 4.6 the effects of different preprocessing steps can clearly be seen. After eliminating outliers from the data, CQ has risen by 3...12% (SF 0.8) and 2...8% (SF 0.9). However, overall CQ peaks at about 0.5 when perfect clustering quality should be close to 1.

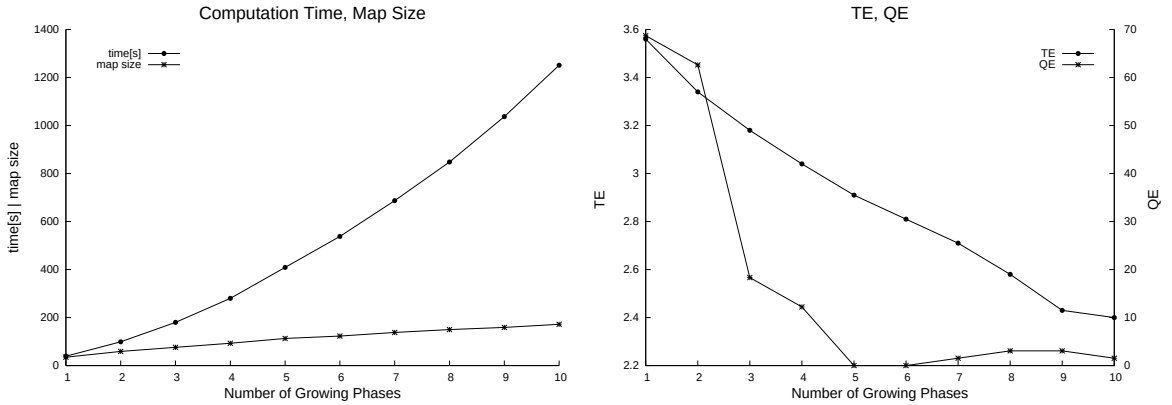


Figure 4.1: Sampled *old* data, SF=0.7, time/size and TE/QE

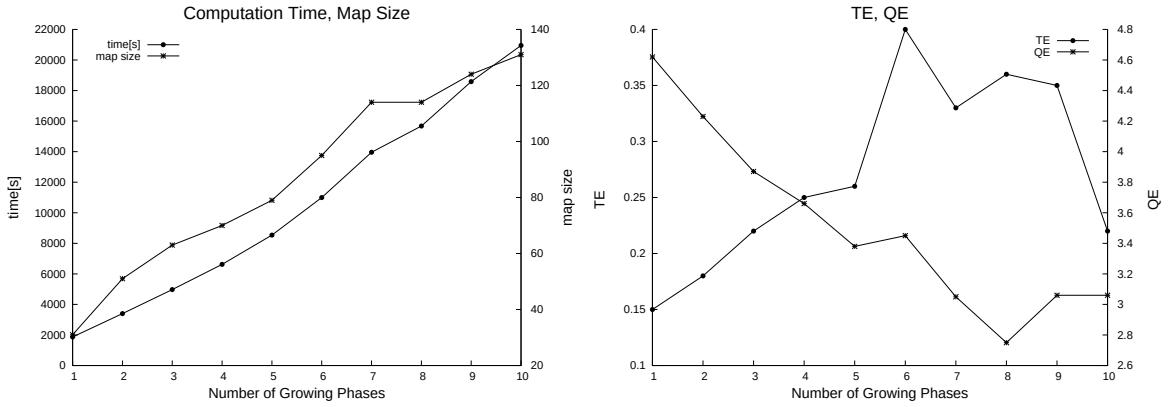


Figure 4.2: Complete *old* data, SF=0.7, time/size and TE/QE

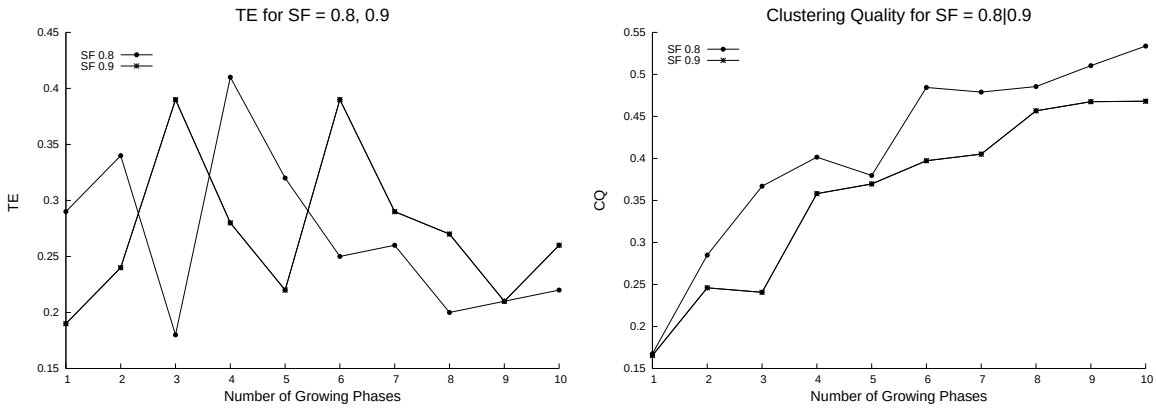


Figure 4.3: Complete *old* data, SF=0.8, 0.9, TE and CQ

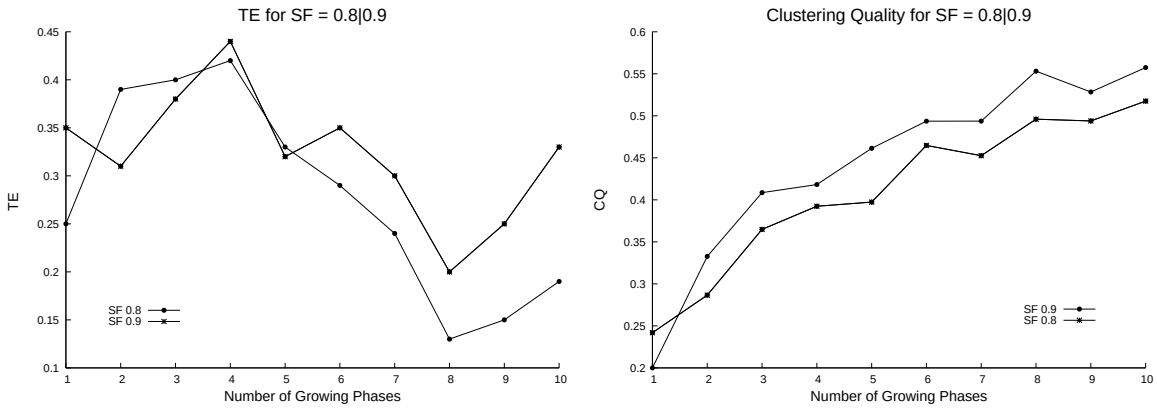


Figure 4.4: Complete *new* data, SF=0.8, 0.9, TE and CQ

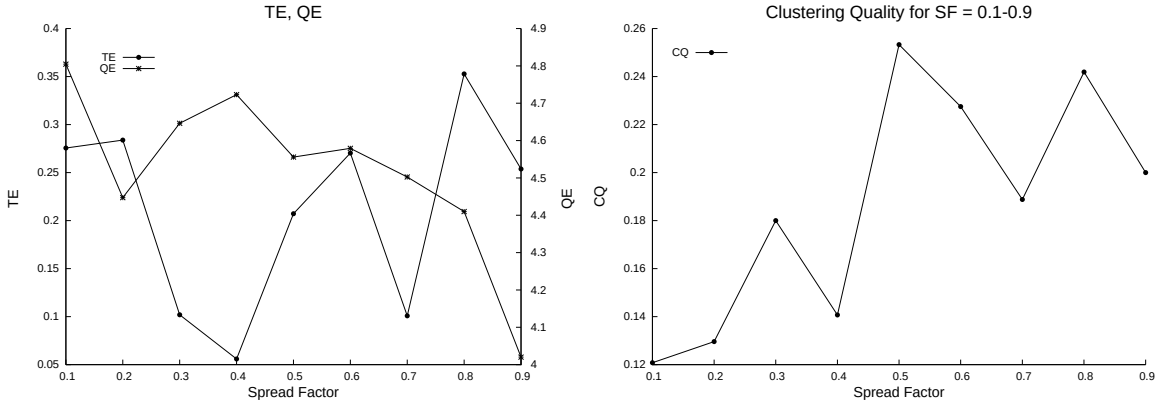


Figure 4.5: Complete *new* data with GP = 1 and SF = [0.1...0.9], TE/QE and CQ

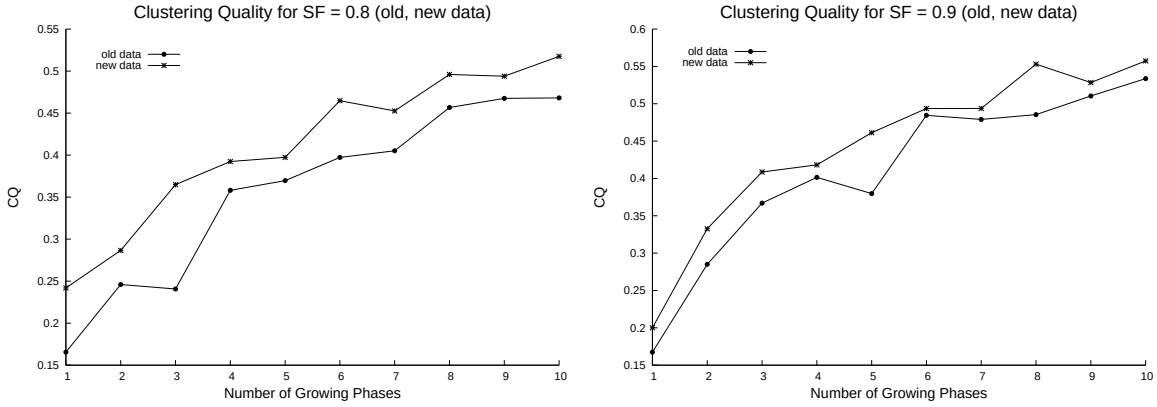


Figure 4.6: Comparison of old and new data CQ, with SF = 0.8 and 0.9

Chapter 5

Summary and Conclusion

5.1 Work done

5.1.1 Summary

Extensive efforts have been devoted to evaluating the feasibility of applying the GSOM to the given manufacturing dataset. Figure 5.1 shows the consecutive process, split up into the three principal steps of *preprocessing*, *processing* and *evaluation*.

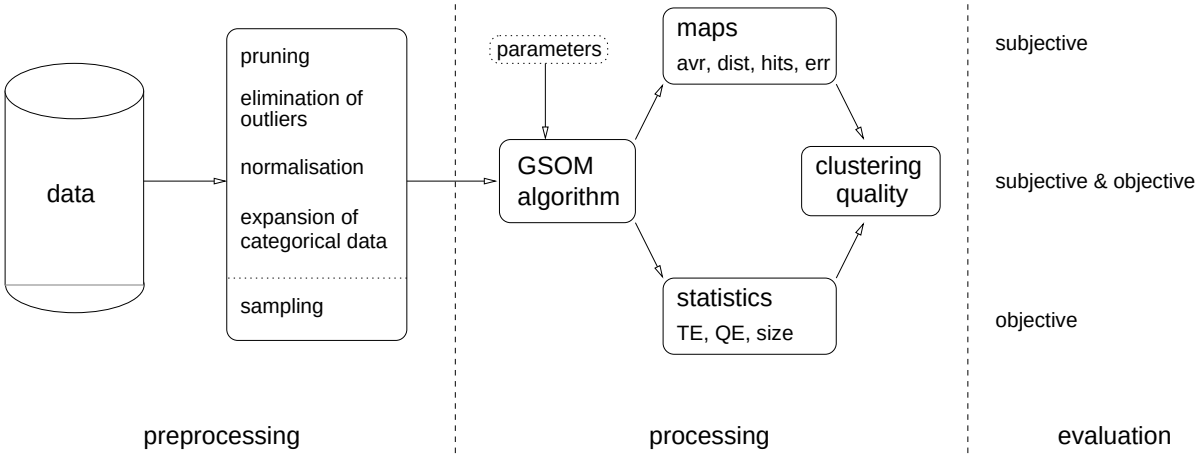


Figure 5.1: Data mining steps

Preprocessing: It starts, of course, with the data itself of which we have only little or no previous knowledge, a fact that is to be remedied by the ensuing data mining. Thorough preprocessing has been performed upon the dataset, including *pruning*, the *elimination of outliers*, *normalisation* and *expansion of categorical data*. A straightforward *sampling* approach to reduce inherent computational complexity has been implemented and tested.

Processing: This step can be held accountable for a fraction of 95% of the overall computing time involved in this work. Important GSOM parameters have been sieved out systematically and have later been applied according to a self-generated schedule. The algorithm's output consists, in one part, of the generated maps, which are neighborhood-preserving projections of the high-dimensional input space onto two-dimensional maps. On the other hand, during the computation of the maps, useful measurements (*TE*, *QE*, *map size*) are being generated. Furthermore, a measure aiding in evaluating the maps, *CQ*, has been introduced, justified, implemented, and tested.

Evaluation: The evaluation of the obtained results from the GSOM algorithm finishes the data mining process. It depends on the user to draw his conclusions from the data, but his *subjective* skills in evaluating the maps are supported by *(semi)-objective* widgets such as TE, QE and CQ from the processing step. Yet and above all, the evaluation step needs one thing: experience.

5.1.2 Conclusion

Summary: Overall, it can justifiably be assumed that *the GSOM algorithm* is able to deal with high-dimensional datasets and *can be used to find distinguishing attributes*. Regarding the extensive set of simulations which were run systematically and the numerous approaches to dealing with the given data, it can also be said that the resulting maps are a good start to follow-up improvements.

Limitations: Since the results are similar throughout the different parameter sets, the error is probably *not* situated inside the *processing* part of Figure 5.1. Having taken expert human knowledge into consideration in the *evaluation* step, this part can also be taken as correct. Therefore, with the evident improvement in CQ due exclusively to different preprocessing steps (Figure 4.6) kept in mind, the most probable cause of not fully satisfactory results seems to be the *preprocessing* part.

5.1.3 Recommendations

Even with the results being rather distant from satisfactory, there are a number of recommendations to be given for future GSOM simulations with the manufacturing dataset used throughout this report. It certainly cannot be avoided to run numerous, computationally heavy experiments with an algorithm which is inherently of near-exponential demands. Therefore, all of the recommendations below target the reduction of computation time, enabling the user to run more experiments in the same amount of time.

Reduce dimension of inputs: Since the running time is inherently dependent on the number and dimension of inputs, try to reduce the number of attributes. This can be done by *sampling* and *pruning*, as demonstrated in Section 3.1.

Use adequate parameter sets: Start with small spread factors and short growing phases. Continually raise SF and GP until sufficient map sizes have been achieved. For the manufacturing dataset, spread factors between 0.7 and 0.9 and the number of growing phases between 1 and 5 could be shown to be adequate. Contingent, questionable benefits of larger parameter sets do not pay off in terms of computation time and, more importantly, subjective map quality.

Get powerful computing equipment: For the full manufacturing dataset under consideration in conjunction with the introduced JAVA implementation of the GSOM algorithm, try to get hold of at least 1024MB of RAM and a fast Intel Pentium-IV (or similar) CPU.

5.2 Possible improvements

This section describes two approaches that could be pursued towards achieving better results. Yet, both of them have not been implemented nor tested; they are ideas developed in conjunction with Arthur Hsu from the Department of Mechanical and Manufacturing Engineering at the University of Melbourne, Australia. [1]

5.2.1 Leaving out attributes

Even with the manufacturer's requirement not to leave out any attributes that might be important, it could still be necessary and insightful to try and remove certain attributes which are probably less distinguishing than others; e.g. with the help of distances between attribute mean values, as shown in Figure 3.7. A positive side-effect, if attributes were removed, would be the reduction of computation time. After all, data mining is supposed to find information and correlation in data that one has not been aware of before doing so.

First, proximate experiments would consist of running the GSOM with the preprocessed numerical attributes only, totally omitting the categorical data which might itself lead to complex changes (see following section).

5.2.2 Weighting of categorical data

During the preprocessing of the input data, data in categorical form are expanded (see Section 3.1). This expansion leads to a thorough change of dimensionalities of the input data, in this case from 80 attributes to roughly 800 attributes. Before the expansion took place, every attribute was equally important, therefore had the same weight. After the expansion, there are roughly ten times as many attributes as before which still have

[1] http://www.mame.mu.oz.au/∼alhs

uniform weights. This might lead to an overvaluation of the categorical attributes which have been expanded.

Hence, it is proposed to introduce some changes to the distance calculation in the GSOM algorithm and add a weighting scheme as follows:

Introduce weight vector for attributes: This weight vector **w** will contain weights for every attribute: **1** for numerical attributes and $\frac{\mathbf{1}}{\mathbf{n}}$ for each attribute of an expanded category, where **n** is the number of distinct attributes of the respective category.

Change calculation of BMU: Equation 2.1 will be changed into the following version, reducing the excess influence of categorical attributes by weighting them:

$$\|\mathbf{w} * \mathbf{x} - \mathbf{m_c}\| = \min_i \{\|\mathbf{w} * \mathbf{x} - \mathbf{m_i}\|\} \tag{5.1}$$

Table 5.1 extends the example from Table 3.2 and introduces the weights as an additional vector.

Category		A	B	C	D	E
1	weight	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$
A		1	0	0	0	0
B		0	1	0	0	0
C	$\Rightarrow$	0	0	1	0	0
A		1	0	0	0	0
D		0	0	0	1	0
E		0	0	0	0	1

Table 5.1: Expansion of categorical data, introduction of weights

Appendix A

Java source code for sampling

A.1 Code for random sampling

```
public void DrawSampleSet(int sample_size)

 /*Draws a set of samples from the input file
  and saves it as a different file */

{
 System.out.println("DrawSampleSet, value of n" +sample_size);

 if ( ilayer == null ) return;

 Vector indices = new Vector();
 Random rand = new Random( System.currentTimeMillis() );

 for (int i = 0; i < ilayer.length; i++)
     indices.add( Integer.toString(i) );

 PrintWriter pw = null;
 try
  {
   String oldFile = ilayer.getFilename();
   String newFile = oldFile.substring( 0, oldFile.length()-4 ) +
    "_sample_" + sample_size + oldFile.substring( oldFile.length()-4 );
   System.out.println("new filename ought to be: " +newFile);
   pw = new PrintWriter( new FileOutputStream(newFile) );
  }
 catch (Exception e)
  {
```

```
   e.printStackTrace();
  }

 String[] colHeader = ilayer.getAllColumnHeaders();

 for (int i = 0; i < colHeader.length; i++)
     pw.print( colHeader[i] + "\t" ); /*output header first*/

 pw.println();
 int c=0;
 while (indices.size() > 0 && c < sample_size)
   {
    int randIdx = rand.nextInt(indices.size());
    int sampleIdx = Integer.parseInt((String)indices.elementAt(randIdx));
    float[] sample = ilayer.getRow(sampleIdx);
    for (int i = 0; i < sample.length; i++)
        pw.print( sample[i] + "\t" );
    pw.println( ilayer.getClassName(sampleIdx) );
    indices.removeElementAt( randIdx );
    c++;
   }
 pw.close();
}
```

A.2 Code for partitioning

```
public void PartitionInputRandomly(int number_of_partitions)

/*partitions the input file into n equally-sized output files
 with the respective contents chosen randomly from the input file
  example: input file has 2,000 inputs
           choose n as 10
           the method creates ten files with 200 inputs each */

{
 if ( ilayer == null ) return;
 Vector indices = new Vector();
 Random rand = new Random( System.currentTimeMillis() );

 for (int i = 0; i < ilayer.length; i++)
     indices.add( Integer.toString(i) );
```

```
 PrintWriter pw = null;
 int rem = ilayer.length % number_of_partitions;
 int length_minus_rem = ilayer.length-rem;
 int samples_per_part = (int)(length_minus_rem/number_of_partitions);
 int c=0;

 try
    {
     String oldFile = ilayer.getFilename();
     for (int j = 0; j < number_of_partitions; j++)
         {
          String newFile = oldFile.substring( 0, oldFile.length()-4 )
         + "_sample" +j + oldFile.substring( oldFile.length()-4 );
          pw = new PrintWriter( new FileOutputStream(newFile) );

          if ((number_of_partitions-j)==1)
             c=rem*(-1);
          else
             c=0;  /*If last sample file is to be written,
                     include ALL remaining samples in it.
                     Realized in this case by altering the initial
                     value of the counter c in while loop below*/

          while ((indices.size() > 0) && (c < samples_per_part))
                {
                 int randIdx = rand.nextInt(indices.size());
                 int sampleIdx =
                     Integer.parseInt((String)indices.elementAt(randIdx));
                 float[] sample = ilayer.getRow(sampleIdx);
                 for (int i = 0; i < sample.length; i++)
                     pw.print( sample[i] + "\t" );
                     pw.println( ilayer.getClassName(sampleIdx) );
                     indices.removeElementAt( randIdx );
                     c++;
                }
          pw.close();
         }
    }
 catch (Exception e)
}
```

Appendix B

Tabular simulation results

SF	GP	SP1	SP2	time[s]	map size	TE[*10^4]	QE
0.7	1	0	0	39	35	68.7	3.510679
0.7	2	0	0	99	59	62.6	3.328941
0.7	3	0	0	180	76	18.3	3.177149
0.7	4	0	0	280	93	12.2	3.055456
0.7	5	0	0	408	113	3.1	2.917654
0.7	6	0	0	537	123	3.1	2.832479
0.7	7	0	0	687	138	1.5	2.734429
0.7	8	0	0	848	150	1.5	2.658500
0.7	9	0	0	1037	159	0.0	2.508615
0.7	10	0	0	1251	172	0.0	2.439344
0.7	5	10	0	1501	113	0.0	2.704890
0.7	10	10	0	3116	172	0.0	2.194546
0.7	5	10	10	2659	113	0.0	2.683838
0.7	10	10	10	5013	171	0.0	2.150550
0.7	100	0	0	53228	400	250.5	0.044048

Table B.1: Simulation results for sampled data, SF=0.7, varying GP/SP1/SP2

SF	GP	SP1	SP2	time[s]	TE	QE	map size
0.7	1	0	0	1868	0.150549	4.621684	31
0.7	2	0	0	3402	0.184934	4.234978	51
0.7	3	0	0	4970	0.218926	3.874527	63
0.7	4	0	0	6621	0.253907	3.660109	70
0.7	5	0	0	8537	0.257631	3.382912	79
0.7	6	0	0	10994	0.400183	3.447568	95
0.7	7	0	0	13957	0.329182	3.045930	114
0.7	8	0	0	15679	0.357570	2.749231	114
0.7	9	0	0	18584	0.352625	3.064933	124
0.7	10	0	0	20953	0.217888	3.061046	131
0.7	5	10	0	26634	0.124969	3.188203	79
0.7	10	10	0	51040	0.081197	2.726300	131
0.7	5	10	10	40669	0.000488	3.105994	79
0.7	10	10	10	79059	0.008364	2.717812	131
0.7	100	0	0	*	*	*	*

Table B.2: Simulation results for *old* complete data, SF=0.7, varying GP/SP1/SP2

SF	GP	time[s]	TE	QE	map size	#nwm	CQ
0.8	1	1202	0.185287	4.416494	39	23	0.1655
0.8	2	2468	0.237546	3.892676	71	35	0.2460
0.8	3	3489	0.393407	3.580527	85	44	0.2407
0.8	4	5218	0.283578	3.084385	117	63	0.3581
0.8	5	6215	0.224237	3.046406	117	62	0.3696
0.8	6	8162	0.385836	2.832159	151	72	0.3972
0.8	7	9525	0.294383	2.482823	157	79	0.4051
0.8	8	11685	0.266606	2.572165	158	77	0.4567
0.8	9	13119	0.210439	2.270554	167	92	0.4675
0.8	10	14614	0.257509	2.239801	176	91	0.4681
0.9	1	2368	0.294628	4.034194	78	36	0.1675
0.9	2	3496	0.338095	3.510404	88	50	0.2850
0.9	3	6527	0.182173	2.771639	164	79	0.3670
0.9	4	8340	0.405678	2.687389	179	80	0.4015
0.9	5	10717	0.318926	2.160446	202	89	0.3797
0.9	6	13905	0.247924	1.912891	245	104	0.4845
0.9	7	16006	0.260684	1.899147	226	97	0.4790
0.9	8	17512	0.203236	1.724007	244	112	0.4855
0.9	9	23218	0.212698	1.175136	301	121	0.5104
0.9	10	26201	0.219169	1.308389	334	115	0.5337

Table B.3: Simulation results for *old* complete data, SF=0.8 and 0.9, GP=[1...10]

SF	GP	time	TE	QE	size	#nwm	CQ
0.8	1	1602	0.35275	4.40988	55	32	0.2419
0.8	2	2959	0.30519	3.77222	82	43	0.2866
0.8	3	5002	0.38304	3.46210	124	68	0.3649
0.8	4	6310	0.44093	3.14806	126	79	0.3924
0.8	5	9187	0.31986	2.93144	179	93	0.3973
0.8	6	10928	0.35019	2.32438	179	103	0.4648
0.8	7	13598	0.30476	2.34424	207	96	0.4526
0.8	8	16457	0.19931	2.02021	232	112	0.4960
0.8	9	18459	0.25419	2.08403	229	121	0.4939
0.8	10	21841	0.32991	1.75955	257	117	0.5175
0.9	1	6061	0.25375	4.02021	105	49	0.2000
0.9	2	9520	0.38779	3.37584	129	81	0.3327
0.9	3	15520	0.40357	2.89653	185	99	0.4086
0.9	4	19372	0.41902	2.66622	194	96	0.4182
0.9	5	30581	0.33023	2.27741	246	123	0.4613
0.9	6	36020	0.28805	1.76875	289	128	0.4936
0.9	7	43535	0.24318	1.80860	331	129	0.4937
0.9	8	51009	0.12854	1.19258	352	157	0.5531
0.9	9	60620	0.14737	1.09385	378	154	0.5284
0.9	10	68975	0.19136	1.03298	391	156	0.5574

Table B.4: Simulation results for *new* complete data, SF=0.8 and 0.9, GP=[1...10]

SF	GP	time	TE	QE	size	#nwm	CQ
0.1	1	713	0.27559	4.804709	9	8	0.1208
0.2	1	1741	0.28385	4.447151	21	17	0.1296
0.3	1	572	0.10187	4.646301	18	17	0.1800
0.4	1	586	0.05594	4.722953	19	15	0.1407
0.5	1	1705	0.20719	4.555871	29	22	0.2533
0.6	1	1033	0.27033	4.579150	35	25	0.2275
0.7	1	2362	0.10081	4.502526	40	24	0.1888
0.8	1	1602	0.35275	4.409880	55	32	0.2419
0.9	1	6061	0.25375	4.020213	105	49	0.2000

Table B.5: Simulation results for *new* complete data, SF=[0.1...0.9], GP=1

Appendix C

Resulting maps

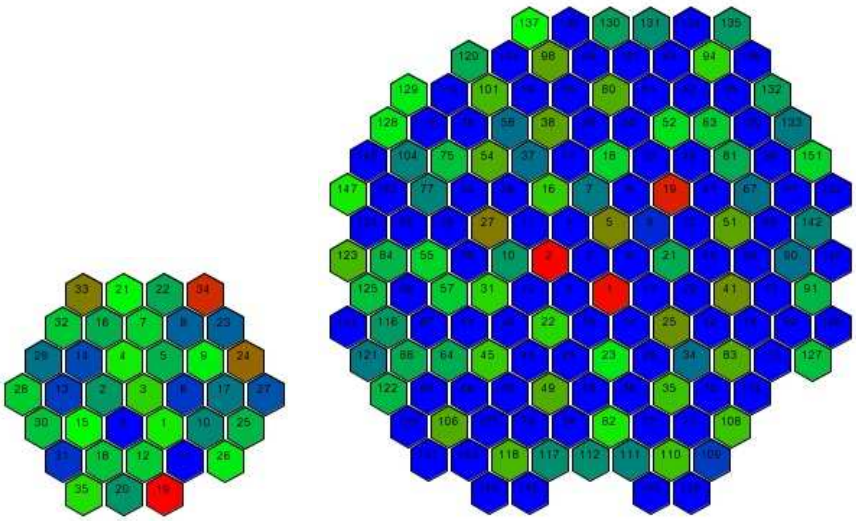


Figure C.1: Hits maps, sampled *old* data, SF=0.7, GP = 1 and 8, SP1 = SP2 = 0

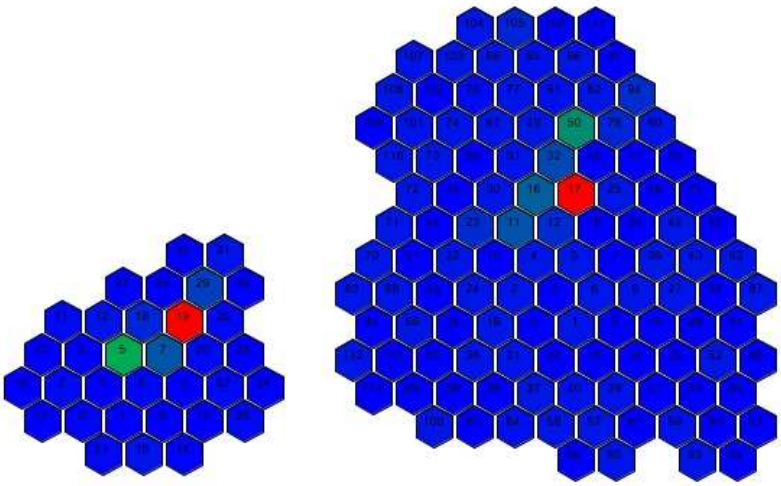


Figure C.2: Hits maps, complete *old* data, SF=0.7, GP = 1 and 8, SP1 = SP2 = 0

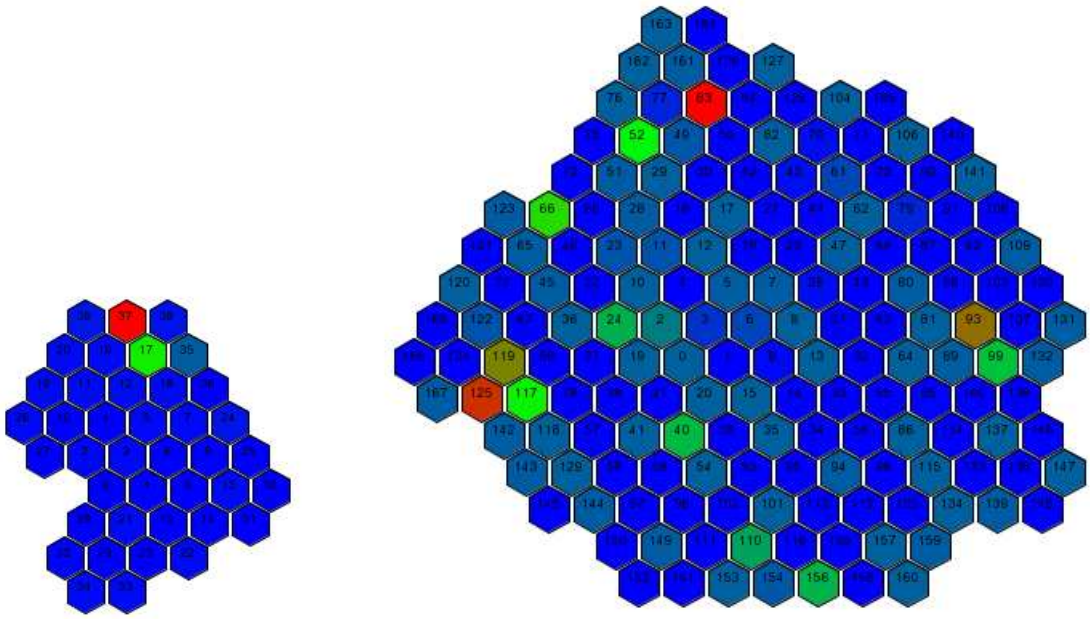


Figure C.3: Hits maps, complete *old* data, SF=0.8, GP=1 and 8, SP1=SP2=0

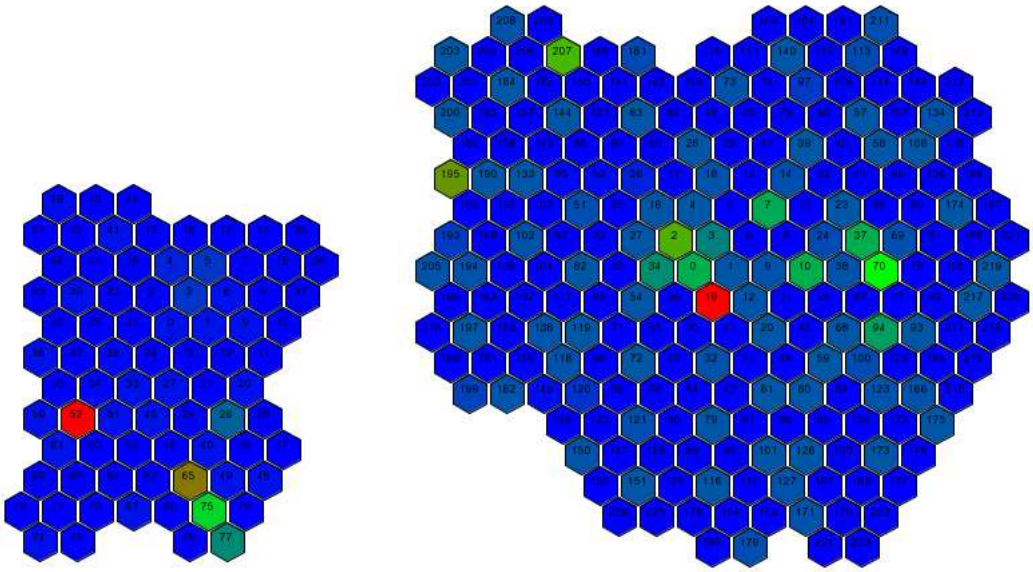


Figure C.4: Hits maps, complete *old* data, SF=0.9, GP=1 and 8, SP1=SP2=0

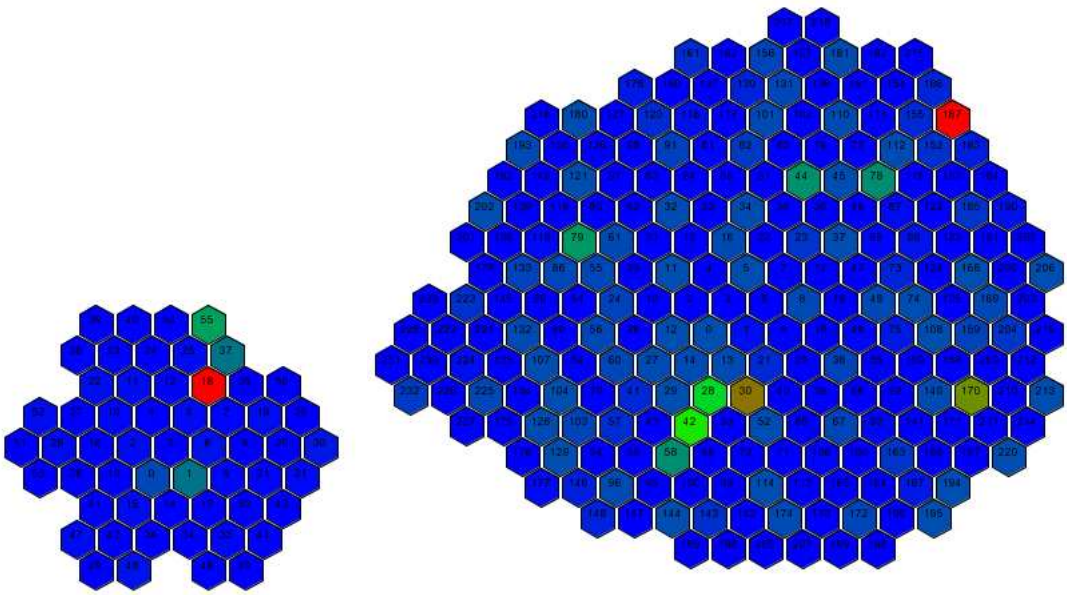


Figure C.5: Hits maps, complete *new* data, SF=0.8, GP=1 and 8, SP1=SP2=0

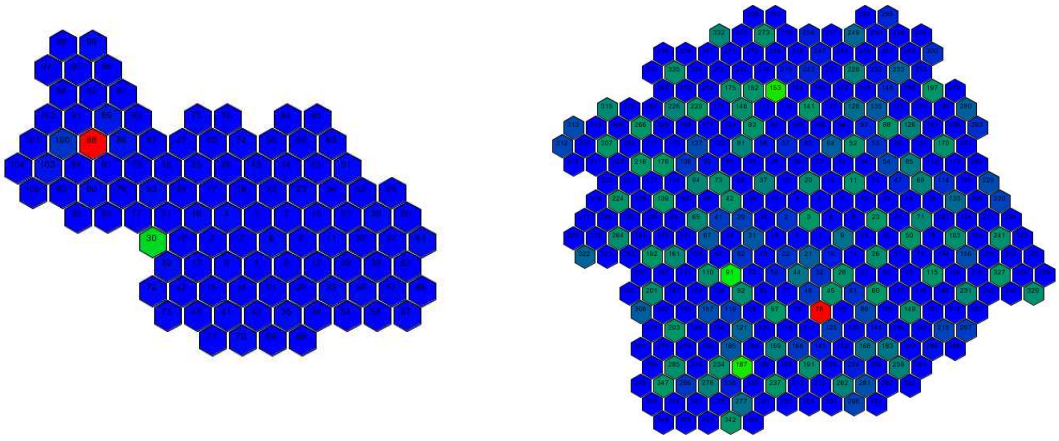


Figure C.6: Hits maps, complete *new* data, SF=0.9, GP=1 and 8, SP1=SP2=0

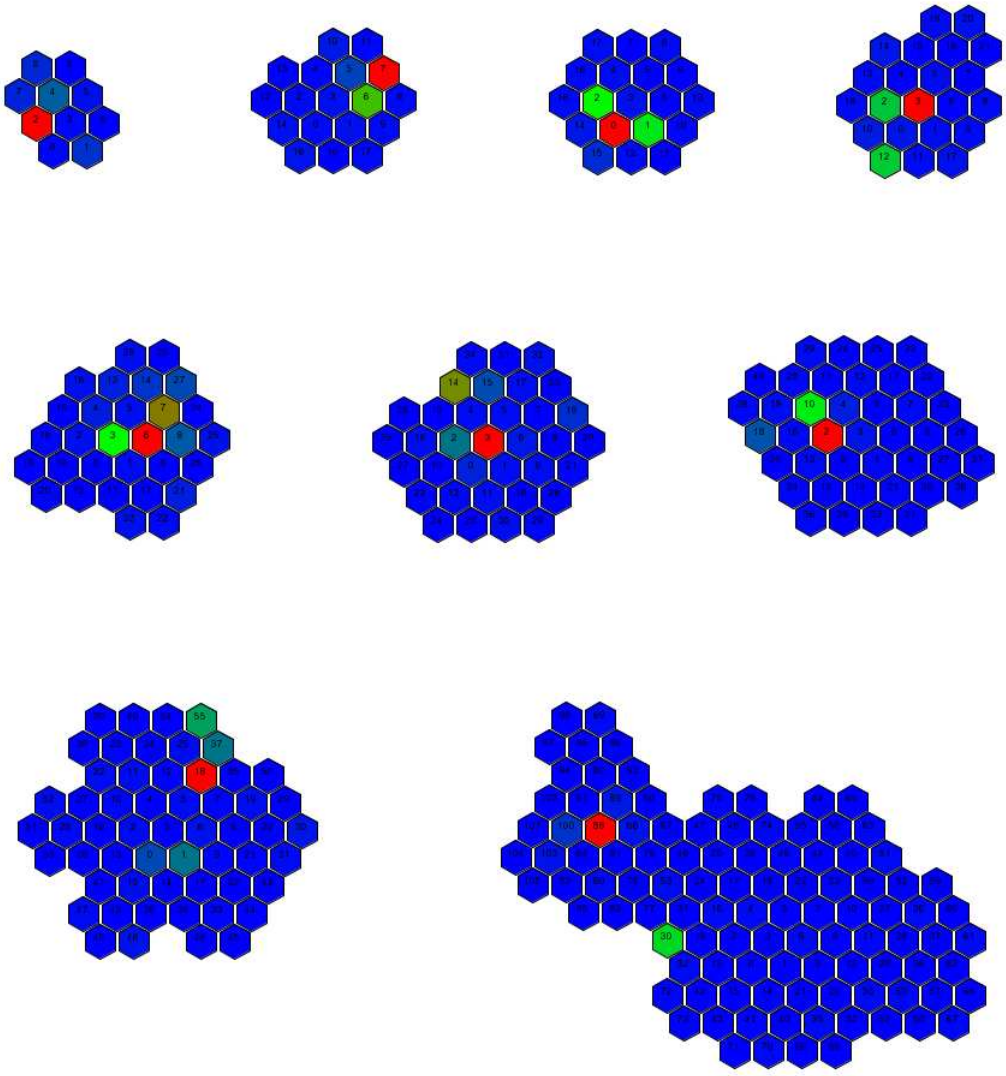


Figure C.7: SOM growing process, SF=0.1...0.9, GP=1, SP1=SP2=0

Selbständigkeitserklärung

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbständig und nur mit erlaubten Hilfsmitteln angefertigt habe.

Magdeburg, 06. Juli 2005

Georg Ruß

Bibliography

[1] Damminda Alahakoon. Controlling the spread of dynamic self organising maps. *Neural Computing and Applications*, 13:168–174, 2004.

[2] Th. Villmann; H.-U. Bauer. Growing a hypercubical output space in a self-organizing feature map. Technical Report TR-95-030, International Computer Science Institute, 1947 Center St., Suite 600, Berkeley, California 95704-1198, July 1995.

[3] Jaakko Hollmën. Process modeling using the self-organizing map. Master's thesis, Helsinki University of Technology, February 1996.

[4] Timo Honkela, Samuel Kaski, Krista Lagus, and Teuvo Kohonen. WEBSOM—self-organizing maps of document collections. In *Proceedings of WSOM'97, Workshop on Self-Organizing Maps, Espoo, Finland, June 4-6*, pages 310–315. Helsinki University of Technology, Neural Networks Research Centre, Espoo, Finland, 1997.

[5] T. Kohonen, S. Kaski, K. Lagus, J. Salojrvi, J. Honkela, V. Paatero, and A. Saarela. Self organization of a massive document collection, 2000.

[6] Teuvo Kohonen. *Self-Organizing Maps*, volume 30 of *Springer Series in Information Sciences*. Springer, Berlin, Heidelberg, New York, third extended edition, 1995, 1997, 2001.

[7] M. Koskela. Content-based image retrieval with selforganizing maps. Master's thesis, Helsinki University of Technology, 1999.

[8] Damminda Alahakoon; Saman K. Halgamuge; Bala Srinivasan. Dynamic self-organizing maps with controlled growth for knowledge discovery. *IEEE Transactions on Neural Networks*, 11(3):601–614, May 2000.

[9] Markus Törmä. Self-organizing neural networks in feature extraction. In *International Archives of Photogrammetry and Remote Sensing ISPRS XVIIIth Congress in Vienna*. Austria, 1996.

[10] Juha Vesanto. Data mining techniques based on the self-organizing map. Master's thesis, Helsinki University of Technology, May 1997.