HOCHSCHULE MITTWEIDA
UNIVERSITY OF APPLIED SCIENCES

MASTER’S THESIS

Ms.
Evgeniya Nemchinova

ML-based classification of problems occurring on SAP supported DB, OS and cloud platforms

2020

Faculty of **Applied Computer Sciences & Life Sciences**

MASTER'S THESIS

ML-based classification of problems occurring on SAP supported DB, OS and cloud platforms

Author:
Evgeniya Nemchinova

Study Programme:
Applied Mathematics for Digital Media (M.Sc.)

Seminar Group:
MA16W1-M

First Referee:
Prof. Dr. Thomas Villmann

Second Referee:
Dr. Maximilian Schmidtke

Mittweida, March 2020

I. Acknowledgements

I wish to express my sincere appreciation to my supervisors in SAP, Dr. Christian Cop, Dr. Arndt Effern, Dr. Maximilian Schmidtke who convincingly guided me to be professional and kept moving along with me through the whole way of this project. Without their persistent help, the goal of this work would not have been realized.

I wish to acknowledge my supervisor in HS Mittweida, Prof. Dr. Thomas Villmann, for his professional insights and experienced advice.

I thank my family, Tomas, Paulina and my grandfather for their encouragement and belief in me.

Bibliographic Information

Nemchinova, Evgeniya: ML-based classification of problems occurring on SAP supported DB, OS and cloud platforms, 65 pages, 28 figures, 6 tables, Hochschule Mittweida, University of Applied Sciences, Faculty of Applied Computer Sciences & Life Sciences

Master's Thesis, 2020

Typeset by LaTeX

Abstract

Anomaly Detection is a very acute technical problem among various business enterprises. In this thesis a combination of the Growing Neural Gas and the Generalized Matrix Learning Vector Quantization is presented as a solution based on collected theoretical and practical knowledge. The whole network is described and implemented along with references and experimental results. The proposed model is carefully documented and all the further open researching questions are stated for future investigations.

II. Contents

III. List of Figures

IV. List of Tables

1 Introduction

Artificial Intelligence has been drawing the attention of computer scientists from all over the world for decades [31]. People always wanted to build up modern powerful machines and computers capable of imitating human's physical and mental abilities. Thus, that could ease many life problems we face every day: cleaning, cooking, collecting and sorting rubbish, the list is endless. All of this led out into a new discipline in 1956 which we now know as AI [31]. Many different approaches have been developed and tried out seeking the solution to one goal - teach machines to act, make a choice and learn like a human being[24].

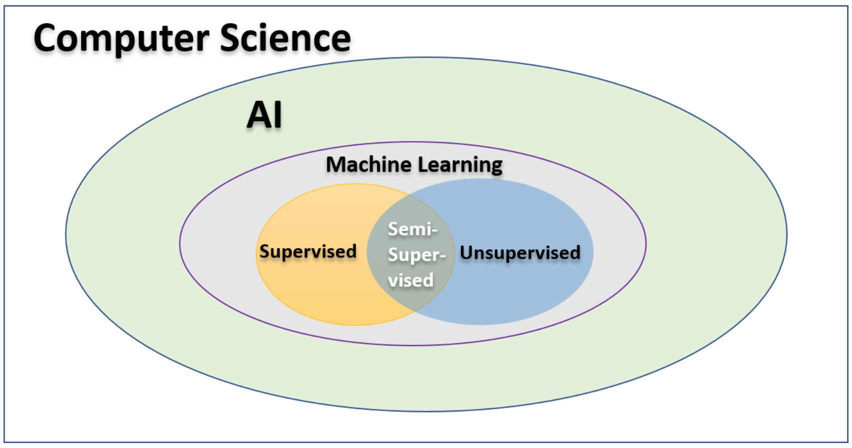


Figure 1.1: Computer Science sub-fields

Before any of the approaches can be applied to teach a machine, one needs to analyze data and transform it into information that can be processed by the machine. Thus, there was developed a huge sub-field of AI, known as **Machine Learning**. It, in turn, includes supervised, unsupervised and semi-supervised learning methods [31]. For a better understanding of the working scope, the sub-sequence of disciplines is depicted in Figure 1.1.
Nowadays, a lot of problems in Machine Learning are related to Anomaly Detection. For example, how to teach a machine to distinguish healthy patients' state and unhealthy one according to given biometrics; or talking about business operating systems, how a machine could monitor the normally functioning state and automatically report about disturbances of any cause. A more detailed explanation about Anomaly Detection is described in [24]. The most widely used approaches to treat such problems are classification and clustering. Classification belongs to supervised training, when one gives an algorithm some input data and teaches the algorithm to return an in advance known output (learning with a master). Whereas, clustering (or learning without a master) is a type of Machine Learning which is used when an end-user is not aware of possible output [11]. Algorithms learn data on their own and split them into groups in accordance

with a similarity measure. Normally, one does not always know the results which might be shown after data is processed. For instance, a doctor collects patients' anamnesis, urine and blood samples, measures pulse and pressure. After gathering all the information about a patient it is not always clear which diagnosis to report, as the same symptoms may signalize to different health problems. Here we need to manage to get an output that we can not predict beforehand, so we can not teach our algorithm to perform a particular, known output. That is why Unsupervised Learning finds application in different areas more and more often. Also, sometimes we do have some available information, that is probably not enough for solving an anomaly detection problem completely, but could be used as a start point. For instance, if we are given with examples of healthy data, we can teach an algorithm how to distinguish this healthy state from an unhealthy one. After a successful classification we can continue with unsupervised learning, trying to make the machine see the differences among variety of anomalies. Such learning approach is known as semi-supervised, because the first step includes classification (an anomaly or not an anomaly) and the second one uses clustering (what kind of anomaly). For more detailed explanation about semi-supervised technique see [12]. All in all, Anomaly Detection finds its application in many spheres such as banks, insurance companies, sales, business. People are trying to catch and patch an outlier behavior as soon as possible in order to keep their business running, otherwise, the enterprises may meet financial and client loss. That is why automatic anomaly detection is essential for middle and big ventures. If there was such a method to identify an anomaly in a huge volume of data and even give some feedback about its root cause, many companies would profit sufficiently. In spite of the topicality of the task, it is not completely solved so far. There is plenty of algorithms, but all of them have their drawbacks, such as poor scaling onto the bigger amount of data, the necessity of knowledge about input data or hyper-parameters pre-definition, which are often to be chosen heuristically [37]. Although it seems unlikely that one can find a universal method that would solve arbitrary clustering or classification tasks (see [11]), we can keep investigating the algorithms and their combinations in order to come up as close as possible to multipurpose solutions.

1.1 Motivation

In this thesis we investigate how Anomaly Detection can help in Root Cause Analysis (RCA) for monitoring applications, and for that we use SAP systems to build and evaluate an anomaly detection and root cause identification model. A SAP system is a business application that consists of application servers connected to SAP HANA databases. The database tracks records of metrics (attributes), such as CPU load, memory usage, and others, see Appendix A. The records are written in a time-series manner. Each timestamp stores an array of 32 attribute values, they are being monitored and analyzed manually by SAP experts when they try to find a root of an appeared problem. For that, the SAP environment is equipped with the Technical Monitoring Cockpit (TMC) which is a built-in tool of SAP application server ABAP to quantify the impact and locate the source of a reported problem. TMC collects data from all above mentioned stacks and also from underlying host. That is, given that we already know that a problem exists because of end-user complaints or of a system landscape alert. To find the root cause of a present problem, the TMC has access to and allows the correlation of data from all technical stacks of an SAP system (Operating System (OS), Database (DB) and Application Server (AS)). When facing a present problem, an expert from SAP (support engineers) usually addresses the problem to a certain category as for example CPU, Memory, IO, network, a.s.o. In order to help the expert in further analyzing the RC of the present problem, the TMC provides a number of analysis scenario screens. However, it is still difficult to find the root cause without anomaly detection, the user just browses through the raw data, whereas having anomaly detection tool, we could offer pre-configured analysis scenarios for a presumed anomalies class. Such an automatic analyzing tool could support the experts in their daily routine.

1.2 Statement of the problem

It would be desirable to develop a *support algorithm* which can help SAP-experts detect and classify deviant behavior of the system. With that goal in mind, we want to use a large amount of healthy data in order to teach the healthy state of the system to an algorithm. As soon as productive system behavior is learned, we are going to separate healthy and abnormal data samples. Furthermore, it shall be evaluated to which degree an automated, data-driven process can support the expert in determining the problem category. For the investigation, we were provided with labeled data of healthy states and anomalies of different classes, so that there are many more healthy states, those are to be learned first in order to create prototypes that could optimally represent the healthy data samples. Thus, the semi-supervised two-step approach was suggested. First, the topology of the healthy data is to be learned with Growing Neural Gas[15]. Then, the binary problem is to be solved: automatically distinguish normal and abnormal data signals. Our final goal is to detect different anomaly states and here we want to investigate how well the Generalized Matrix Learning Quantization (GMLVQ)[32] can perform in our case.

In Figure 1.2 we can see the structure of the Anomaly Detection Tool we want to build. As a healthy state is extensive and unhealthy signals are sparse, we first find a representation of that class with GNG.

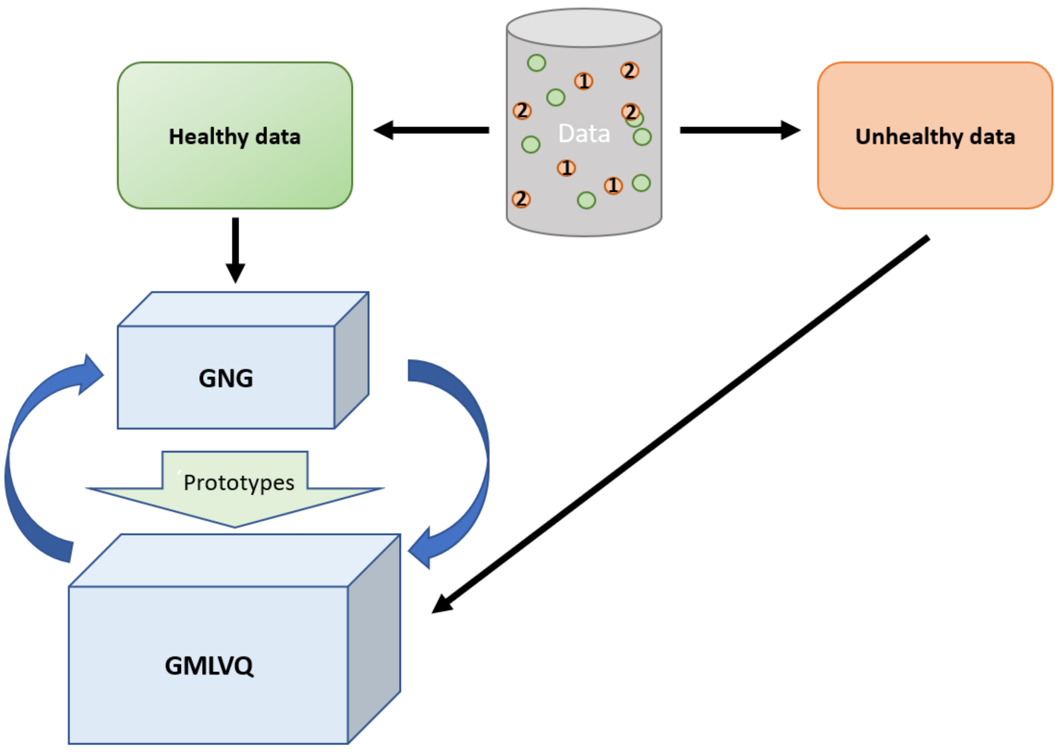


Figure 1.2: The architecture of the Anomaly Detection tool.

Then the prototypes retrieved from the Neural Gas are used as data samples for GMLVQ and compared with anomalous samples.

1.3 Thesis structure

The further part of this thesis is organized as follows: Chapter 2 gives some basic definitions in ML that will be used in this work. Chapters 3,4 describe basic concepts of the GNG and the GMLVQ. In Chapter 5 we describe our data workflow in the GNG and evaluate the results. Chapter 6 describes he the workflow of the GMLVQ classifier with final evaluation. This work is finished by Chapter 7, where we discuss the results and propose a scope of open questions for possible future research.

2 Basics

This chapter explains the main terms and concepts that we use in the following chapters and helps to understand the following content.

2.1 Outliers

Outliers are data signals in well-structured data that deviate from the normal behaviour [38]. In this work we will also use term **anomalies** because it is intuitively more clear and used as a main term in SAP.

2.2 Root Cause Analysis

The process of discovering the ground cause of a problem, that appears in an alert system or is reported by a user, is called RCA. The information about the root cause helps to understand what to update in the system to prevent the occurring of the same problem in future.

2.3 Technical Monitoring Cockpit

The TMC is is an RCA tool in SAP that shows users the full stack of SAP applications such as databases, operating systems and application servers. Users can monitor data in each of the connected systems in order to catch the moment of interruption and indicate the RC of it.

2.4 The Self-Organizing Maps

In 1982 a Finnish professor Dr. Teuvo Kohonen introduced an unsupervised approach that realize dimensionality reduction, mapping high-dimensional data onto 2D or 3D space [4]. The approach is known as a self-organizing map (SOM). SOMs maintain a topology of input and output spaces, guided by the fact that the input vectors that are close in high dimensional space are also turned out to be neighboring neurons after mapping onto the 2D space. A valuable characteristic of the learning process is that the neurons are being organized unsupervisely (a self-organization). They group according to the similarity between each other. The following Figure 2.1 depict the architecture of a SOM, [4].

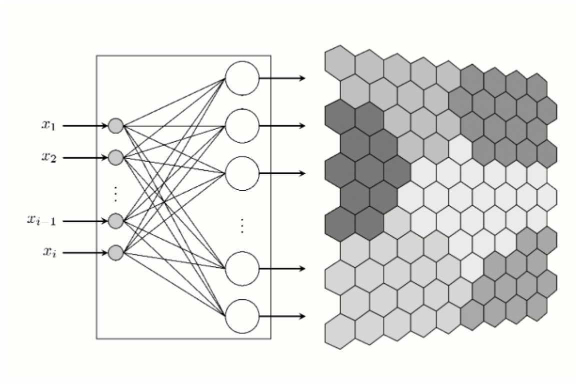


Figure 2.1: Kohonen Architecture

A SOM, additionally, uses competitive learning to adjust its neurons. Only one neuron is activated at each iteration in which the features of an input signal are presented to the neurons network, as all vectors compete for the right to respond to the input. The winning neuron - **the Best Matching Unit (BMU)** - is chosen in accordance with the similarity, between the current input values and all the neurons in the projected space. To get a better understanding of the concepts of SOMs, see [22]. In this thesis we applied an unsupervised algorithm that was driven by SOM concepts: Growing Neural Gas. It is explained in Chapter 3.

2.5 Confusion Matrix

A confusion matrix is used as a performance measure of ML classification algorithms. For example, we are given with some data sample and we know to which class it belongs to. Now we want our algorithm to match it to a particular class. After the algorithm run, we can construct a confusion matrix (Table 2.1), where True Positive and True Negative outcomes mean that our algorithm made a correct prediction about the class for a given data signal. Otherwise, we get a miss-classification (False Negative or False Positive). For further details, see [1].

		Actual Values	
		$+$	$-$
Predicted Values	$+$	$TruePositive$	$FalsePositive$
	$-$	$FalseNegative$	$TrueNegative$

Table 2.1: Confusion Matrix

2.6 ROC and AUC

When we want to visualize the performance of a multi-class classification problem, we can use Receiver Operating Characteristics, which is a curve and the Area Under the Curve (AUC). Thus, AUC-ROC curve is another evaluation measure for classification. It shows how well our algorithm can distinguish among different classes. In Figure 2.1 we can see that the ROC curve is plotted with True Positive Rate against False Positive Rate values.

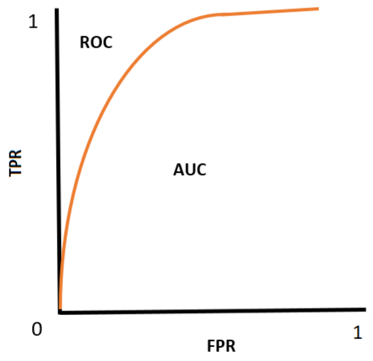


Figure 2.2: ROC-AUC curve

Where we calculate

$$TPR(Sensitivity) = \frac{TP}{TP+FN},$$

$$Specificity = \frac{TN}{TN+FP},$$

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN}.$$

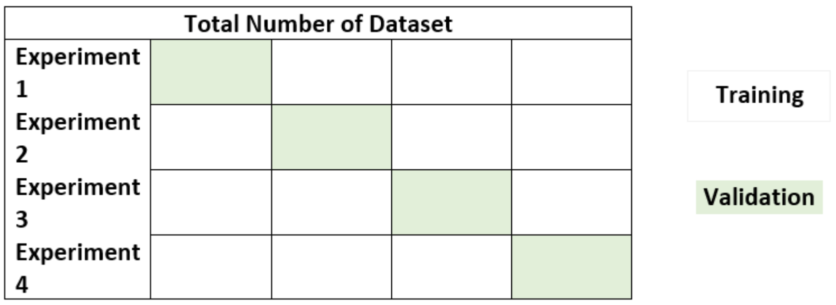
The most accurate model is the one that has AUC close to 1, meaning that the algorithm performs good measure of separability. For more detailed interpretation of the ROC-AUC see [3].

2.7 Principal Component Analysis

A PCA is a statistical method that converts a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables or so-called *principal components* [21]. The conversion takes place due to an orthogonal transformation. PCA can be used for dimensionality reduction and visualization. In this work, we will use it to picture our 5-by-32 dimension input signals in a 2-dimension plot.

2.8 K-Fold Cross Validation

When we need to evaluate a ML algorithm accuracy, we can use CV technique [2]. First, the input data is split into *k*-sets. The *k* subsets are called *folds*. In the second step, we iteratively fit the model *k* times, each time training the data on *k-1* of the folds and evaluating on the *k*-th fold (called the validation data). In the Figure we show an example on 4-folds validation. The first iteration we train on the first four folds and evaluate on the fifth. The second time we train on the first, second, third, and fifth fold and evaluate on the fourth. We repeat this procedure 2 more times, each time evaluating on a different fold. At the very end of training, we average the performance on each of the folds to come up with final validation metrics for the model.



	Total Number of Dataset			
Experiment 1	🟩			
Experiment 2		🟩		
Experiment 3			🟩	
Experiment 4				🟩

⬜ Training

🟩 Validation

Figure 2.3: 4-Fold Cross Validation [2]

This approach is used for evaluation of the GNG in Chapter 5. For more information about K-Fold CV see [2],[30].

3 Growing Neural Gas (GNG)

In many unsupervised algorithms learning aims at optimal representation of an input data structure: **topology learning**[20], [13]. In other words, we want to discover and represent a fundamental structure of the given data with less number of data representers *prototypes*.
Assuming that we are given with $P(x)$ - the distribution of some data (high-dimensional), the task is to design such a topological structure that would portray the given data distribution topology. The algorithm was driven by SOMs and was described by Martinetz and Schulten in 1993 [26]. The approach uses the synergy of Neural Gas (NG) [36] and 'competitive Hebbian learning' CHL [25]. Further, an advanced approach was described by Fritzke where NG was replaced with GNG [15]. Growing Neural Gas is a derivative algorithm from NG and is also based on CHL.

3.1 Growing Neural Gas Algorithm

Growing Neural Gas Algorithm (GNG) is a self-organized neural net that learns the topology of input data on the flow along with new input samples insertion. In this Master thesis, the code was written based on the GNG algorithm given in the article [15], where it was assumed that we have a network, (or topologically - a graph) that consists of two sets:

- Ω - a set of *reference vectors* (nodes) $\omega_c \in \mathbb{R}^n$.
- ***N*** - a set of non-weighted edges among pairs of the neurons.

The algorithm starts with the initialization of two randomly chosen neurons. The term *'neurons'* means some abstract objects, whilst the real numerical data that represents the corresponding neurons in $\mathbb{R}^n$) is defined as *'reference vectors'*. But from now on the two terms *'neurons'* and *'reference vectors'* in GNG will be used as equal for convenience. So, each reference vector is assumed to have an accumulated error variable and a set of edges emanating out of it. Error measure is aimed to regulate the neurons' birth. The edges reflect topological neighbors of every existing node and are assigned with another statistical measure - an age variable. The age of a topological neighborhood (node-connection) is meant to regulate removing old links, keeping a network topology updated and preserving Delaunay triangulation [19]. Thus, we have:

In the preamble of the algorithm one sets 8 hyper-parameters which are constant in time and whose sole role is to moderate values of errors and moving rates of the nodes.

N – maximum number of nodes;

$MaxIt$ – maximum number of itterations;

Ł – if the index of the current data point is multiple of L, then insert a new node;

$\varepsilon_b, \varepsilon_n$ – local eror measures for BMU and its neighbours;

α – moderate a local error;

δ – moderate the global error;

T – maximum age of an edge;

Further, there is an *n*-dimensional space of input signals spread according to a probability density function $P(x)$. One signal is randomly chosen in each iteration.
The following algorithm describes an *'advanced'* variation of NG, so-called Growing Neural Gas approach which allows to gradually generate the network structure due to CHL without concern about the dimensionality of input data.

GNG step-by-step:

1. Insert two randomly located neurons ω_{c_1} and ω_{c_2} in $\mathbb{R}^n$.
2. Get a random data signal x from your input set.
3. Calculate distances (a chosen similarity measure) between the neurons and x; determine two *best matching units*: s_1, s_2.
4. Age all the edges flowing out of s_1.
5. Update the squared error of the winner-unit s_1 according to the chosen dissimilarity measure (e.g. Euclidean distance):

$$\Delta E = d^2(\omega_{s_1}, x). \tag{3.1}$$

6. Shift s_1 and all the neurons that are directly connected to it towards the input sample x. The adaptation rule is given below:

$$\Delta\omega_{s_1} = \varepsilon_{s_1} * (x - \omega_{s_1}), \tag{3.2}$$

$$\Delta\omega_n = \varepsilon_n * (x - \omega_n), \tag{3.3}$$

   where ε_{s_1} and ε_n - learning rates for ω_{s_1}, ω_n accordingly.
7. Unless the 2 best matching units s_1 and s_2 are connected, join them with a new zero-aged edge. Otherwise, set the age of the already existed connection to 0.
8. The edges of age larger than the predefined value T are removed. After that, all alone neurons that might appear after edge deleting also have to be removed from the network.

9. If the amount of existing neurons is not exceeded and the index of the input is a multiple of Ł, then add a new reference vector in accordance with the following steps:

   - Let w_p be the neuron in the network with the maximum accumulated error. Consider a subset B_p , a sub-graph of the net where all the neurons are connected to w_p with an edge, we call such neurons topological neighbors of w_p and B_p its neighborhood. Determine its topological neighbor $r \in B_p$, also with the highest accumulated error. So we get ω_p, ω_r such that:
   - Create a new reference vector ω_q:

   $$\omega_q = \frac{1}{2} * (\omega_p + \omega_r) \tag{3.4}$$

   - Manage the links: connect nodes $q - p$ and $q - r$ pairwise and disconnect $p - r$
   - Decrease the age of the edge $p - r$ and initialize the error variable of q in this manner:

   $$E(p) = \alpha * E(p),$$
   $$E(r) = \alpha * E(r),$$
   $$E(q) = E(q).$$

10. Through multiplying by δ decrease all the *error variables*

   $$\Delta E = E * \delta. \tag{3.5}$$

11. The stopping criterion may vary. It is achieved if, for example, the topology of the network gets stable (new nodes neither appear, nor move).

All in all, with the GNG one can construct a general structure of the input data that helps to get a better understanding of the data topology. The neurons that are created in the net may serve as prototypes of the original data and further be used to support different clustering techniques, for example as it is explained in [13].

4 Learning Vector Quantization

In this Chapter we consider different LVQ approaches, their evolution and differences. LVQ is quite a mighty classification scheme due to its easy implementation, controllable classifier, multi-class problems management, and explicit intuitive interpretation [33]. Therefore, the algorithm has been widely employed in different academic and commercial fields, among which bio-informatics, robotics, bio-engineering and image analysis. Despite its huge popularity among data scientists LVQ does still suffer from a number of shortcomings, for instance, slow convergence and unstable training behavior, resulting in unpredictable performances [35]. Thus, there have been derived several modifications of the original technique in order to beat the drawbacks up. As an example, Sato and Yamada in their work [32] presented an approach that allows obtaining a scheme for minimizing the cost function of the learning. It is known as the GLVQ. Then a matrix version (the GMLVQ) was elaborated [34], that we consider in Section 4.4 in more details.

4.1 The concepts of Learning Vector Quantization

LVQ was introduced by T. Kohonen in 1997 and described as a novel prototype-based approach which belongs to supervised learning [22]. The sole purpose of the algorithm is to outline class regions in the given data space according to the closest-neighbor rule. Let $\mathscr{X} \subseteq \mathbb{R}^n$ be input data, $\mathscr{C} = \{1,...,C\}$ - classes, then training data X is defined as a graph:

$$\mathrm{X} = \{(\mathbf{x}, c(\mathbf{x})) \mid \mathbf{x} \in \mathscr{X}, c(\mathbf{x}) \in \mathscr{C}\}.$$

Where $\mathbf{x}$ - a data sample, $c(\mathbf{x})$ - a class label. One also defines a set of code-book vectors $\mathscr{W} = \{\mathbf{w}_1,...,\mathbf{w}_N\}$, $\mathscr{W} \subseteq \mathbb{R}^n$ such that, for each $c \in \mathscr{C}$ there exists $\mathbf{w}_j$ with $c = c(\mathbf{w}_j)$. Then, introducing new data $\mathscr{X}'$ we manage the class assignment

$$c(\mathbf{x}) = c(\mathbf{w}_{s(x)}), \tag{4.1}$$

where

$$s(\mathbf{x}) = \underset{j}{argmin}(d(\mathbf{x},\ \mathbf{w}_j)), \tag{4.2}$$

the approach is known as *winner-take-all competition (WTAC)* and $\mathbf{w}_{s(x)}$ is the winner prototype. Here d is a chosen distance, normally Euclidean $d_E(\mathbf{x}, \mathbf{w}_k) = (\mathbf{x} - \mathbf{w}_k)^2$, but depending on the original problem other dissimilarity measures can be chosen instead [29]. Then the task transforms into the class assignment with the following quadratic variant as a more general form

$$d_\Omega(\mathbf{x},\ \mathbf{w}_j) = (\Omega\mathbf{x} -\ \mathbf{w}_j)^2, \tag{4.3}$$

we denote $\Omega \in \mathbb{R}^{n_p \times n}$ as dimension data projection matrix where n_p the projection dimension[9]. In case when $\Omega = id(x)$ with $n_p = n$, we get the Euclidean distance.

Learning scheme is realized by the prototypes distribution onto the projection space $\mathbb{R}^{n_p}$ in order to depict the class distributions. If we set $n_p = n$, the prototypes will live in the same dimensional space as the original data signals. The idea of the learning, though, is to find a lower-dimensional representation of our data (the quantization). That is why we avoid such an assignment.

4.2 Milestones of learning scheme for LVQ algorithms

The LVQ is presented by 3 separate algorithms: LVQ 1, LVQ 2, LVQ 3, which are described in [23] by T. Kohonen. In this section, I describe some main steps and ideas behind the general workflow of the algorithms. In order to perform an approximation of a Bayesian classifier, T.Kohonen proposed a heuristic learning construction. Each randomly chosen data sample is assigned to a class label according to 4.2. In a second step the winning code-book vector $\mathbf{w}_{s(x)}$ is to be either shifted closer to the data vector $\Omega\mathbf{x}$ or repelled from it. Thus, the updating process is defined by

$$\Delta\mathbf{w}_{s(x)} = \varepsilon\psi(\mathbf{x}, s(x)) \cdot (\Omega\mathbf{x} - \mathbf{w}_{s(x)}), \tag{4.4}$$

where

$$\psi(\mathbf{x}, s(\mathbf{x})) = \begin{cases} 1, & c(\mathbf{x}) = c(\mathbf{w}_{s(x)}) \\ -1, & c(\mathbf{x}) \neq c(\mathbf{w}_{s(x)}), \end{cases}$$

and $0 < \varepsilon \ll 1$ is the ***learning rate***.

4.3 Generalized LVQ algorithm

LVQ severely suffers from reference vectors divergence. To overcome this obstacle and to minimize the cost function, GLVQ was proposed. As before we consider labeled training data $(\mathbf{x}, c(\mathbf{x}))$, where $c \in \mathscr{C} = \{1, ..., c\}$ and a set of labeled prototypes $(\mathbf{w}_j, c(\mathbf{w}_j))$ with $\{\mathbf{w}_j\}_1^N = \mathscr{W}$.
Notation:
$\mathbf{w}^+$ is the best matching prototype for $(\mathbf{x}, c(\mathbf{x}))$ among all prototypes $\mathbf{w}_j$ with $c(\mathbf{w}_j) = c(\mathbf{x})$;
$\mathbf{w}^-$ is the closest prototype for $(\mathbf{x}, c(\mathbf{x}))$ among all prototypes $\mathbf{w}_k$ with $c(\mathbf{w}_k) \neq c(\mathbf{x})$ (the best incorrect prototype);
$D^+ = d(\mathbf{x}, \mathbf{w}^+)$;
$D^- = d(\mathbf{x}, \mathbf{w}^-)$.
Then, we define the *relative distance dissimilarity* $\mu(x)$:

$$\mu(\mathbf{x}) = \frac{D^+ - D^-}{D^+ + D^-}, \tag{4.5}$$

where $-1 \leq \mu(\mathbf{x}) \leq 1$ and $\mu(\mathbf{x}) < 0$ means $\mathbf{x}$ is correctly assigned to the class, whereas $\mu(\mathbf{x}) < 0$ signals about an incorrect classification. Under the assumption that $d(\mathbf{x}, \mathbf{w})$ w.r.t. $\mathbf{w}_{+-}$ differentiable, $\mu(\mathbf{x})$ is a differentiable classifier function.
Now we need a method to decrease error rates, so that $\mu(\mathbf{x})$ shall go down for all input

signals. Keeping this thought in mind the following learning criterion is deployed:

$$\mathscr{F} = \sum_{k=1}^{N} \phi(\mu(\mathbf{x}_i)), \tag{4.6}$$

we set the number of training data samples N, and a monotonically increasing function $\phi(\mu)$. Normally, the identity function $f(z) = z$ or the sigmoid function $f(z) = 1/(1+e^{z\theta})$ is chosen as $\phi(\mu)$. The goal is to minimize $\mathscr{F}$, therefore, the units $\mathbf{w}^+$ and $\mathbf{w}^-$ are to be updated with the mean of the *steepest descent method*, where we additionally introduce $\alpha = const, 0 < \alpha \ll 1$, then we perform updating:

$$\triangle \mathbf{w} = -\alpha \frac{\partial \mathscr{F}}{\mathbf{w}} \tag{4.7}$$

Precisely,

$$\triangle \mathbf{w}^+ = +\alpha \frac{\partial \mathscr{F}}{\partial \phi} \frac{\partial \phi}{\partial \mu} \frac{\partial \mu}{\partial D^+} \frac{\partial D^+}{\partial \mathbf{w}^+} \tag{4.8}$$

And analogically

$$\triangle \mathbf{w}^- = -\alpha \frac{\partial \mathscr{F}}{\partial \phi} \frac{\partial \phi}{\partial \mu} \frac{\partial \mu}{\partial D^-} \frac{\partial D^-}{\partial \mathbf{w}^-} \tag{4.9}$$

Taking $d_\Omega = (\Omega \mathbf{x} - \mathbf{w}_j)^2$, one gets the following layout:

$$\triangle \Omega = \frac{\partial \mathscr{F}}{\partial \phi} \frac{\partial \phi}{\partial \mu} \cdot \frac{\partial \mu}{\partial \Omega} \tag{4.10}$$

$$\frac{\partial \mu}{\partial \Omega} = \frac{\partial \mu}{\partial D_\Omega^+(\mathbf{x})} \cdot \frac{\partial D_\Omega^+(\mathbf{x})}{\partial \Omega} + \frac{\partial \mu}{\partial D_\Omega^-(\mathbf{x})} \cdot \frac{\partial D_\Omega^-(\mathbf{x})}{\partial \Omega} \tag{4.11}$$

where

$$\frac{\partial \mu}{\partial D_\Omega^+(\mathbf{x})} = \frac{+2D_\Omega^-(\mathbf{x})}{(D_\Omega^+(\mathbf{x}) + D_\Omega^-(\mathbf{x}))^2} \tag{4.12}$$

and

$$\frac{\partial \mu}{\partial D_\Omega^-(\mathbf{x})} = \frac{-2D_\Omega^+(\mathbf{x})}{(D_\Omega^+(\mathbf{x}) + D_\Omega^-(\mathbf{x}))^2} \tag{4.13}$$

Thus, we obtain the following GLVQ learning:

$$\triangle \mathbf{w}^+ = +\alpha \frac{\partial \phi}{\partial \mu} \frac{D^-}{(D^+ + D^-)^2} (\mathbf{x} - w^+) \tag{4.14}$$

$$\triangle \mathbf{w}^- = -\alpha \frac{\partial \phi}{\partial \mu} \frac{D^+}{(D^+ + D^-)^2} (\mathbf{x} - w^-) \tag{4.15}$$

We obtained (4.12) and (4.13) as components of the derivative $\frac{\partial \mu}{\partial \Omega}$ in (4.10). This version of the GLVQ algorithm got known as GMLVQ which we consider further in the next section.

4.4 Generalized Matrix Learning Quantization

The ***"Generalized Matrix Learning Quantization"*** is introduced as an important concept of LVQ, which uses a full matrix of relevance in the similarity measure [34]. The adaptation of code-book vectors $w_k \in \mathbb{R}^n$ to training data is being handled in regards to the class distribution among the training data vectors.
Mathematically speaking, the algorithm is built on a general distance form (a dissimilarity evaluation).

$$d_\Lambda(\mathbf{x},\, \mathbf{w}_j) = (\mathbf{x} - \ \mathbf{w})^T \Lambda (\mathbf{x} - \ \mathbf{w}), \tag{4.16}$$

The full $N \times N$ matrix Λ declares correlations between the features. We deploy a GSE distance in a suitably transformed space enforcing the matrix Λ to be positive semi-definite and symmetric. Here, we come up with the approach below

$$d_\Omega(\mathbf{x},\, \mathbf{w}_j) = (\mathbf{x} - \ \mathbf{w})^T \Omega^T \Omega (\mathbf{x} - \ \mathbf{w}). \tag{4.17}$$

The substitution $\Lambda = \Omega^T \Omega$ guarantees that the above-mentioned essential constraints on Λ are satisfied. In such a way a general form of squared Euclidean distance can be defined in *a suitable transformed space*. According to well-known linear algebraic transformations, it is shown

$$u^T \Lambda u = u^T \Omega^T \Omega u = (\Omega^T u)^2 \leq 0$$

for all u. The initializing of Ω one may choose a random $N \times N$ matrix. After the above-mentioned substitution of Λ, one achieves the following SED representation

$$d^\Lambda(\mathbf{w},\mathbf{x}) = \sum_{lmn} (\mathbf{x}_l - \mathbf{w}_l) \Omega_{nl} \Omega_{nm} (\mathbf{x}_m - \mathbf{w}_m). \tag{4.18}$$

The adaptation process is done through the chain of equations below and includes the computing of derivatives with respect to **w** and Ω

$$\nabla_{\mathbf{w}} d^\Lambda(\mathbf{w},\mathbf{x}) = 2\Lambda(\mathbf{x} - \mathbf{w}) = -2\Omega^T \Omega (\mathbf{x} - \mathbf{w}) \tag{4.19}$$

as well as

$$\begin{aligned} \frac{\partial d^\Lambda(\mathbf{w},\mathbf{x})}{\partial \Omega_{pq}} &= \sum_m (\mathbf{x}_q - \mathbf{w}_q) \Omega_{pm} (\mathbf{x}_m - \mathbf{w}_m) + \sum_l (\mathbf{x}_l - \mathbf{w}_l) \Omega_{pl} (\mathbf{x}_q - \mathbf{w}_q) \\ &= 2 \cdot (\mathbf{x}_q - \mathbf{w}_q) [\Omega(\mathbf{x} - \mathbf{w})]_p, \end{aligned} \tag{4.20}$$

the indexes *p,q* declare vectors components and one gets the equations for prototypes adaptation:

$$\Delta \mathbf{w}^+ = \varepsilon \cdot 2 \cdot \phi'(\mu(\mathbf{x})) \cdot \mu^+(\mathbf{x}) \cdot \Lambda \cdot (\mathbf{x} - \mathbf{w}^+), \tag{4.21}$$

$$\Delta \mathbf{w}^{-} = -\varepsilon \cdot 2 \cdot \phi'(\mu(\mathbf{x})) \cdot \mu^{-}(\mathbf{x}) \cdot \Lambda \cdot (\mathbf{x} - \mathbf{w}^{-}). \tag{4.22}$$

In order to update the matrix elements we apply

$$\begin{aligned} \Delta\Omega_{pq} = -\varepsilon \cdot 2 \cdot \phi'(\mu(\mathbf{x})) \cdot \\ \left(\mu^{+}(\mathbf{x}) \cdot \left((\mathbf{x}_q - \mathbf{w}_q^{+})[\Omega(\mathbf{x} - \mathbf{w}^{+})]_p\right) - \right. \\ \left. \mu^{-}(\mathbf{x}) \cdot \left((\mathbf{x}_q - \mathbf{w}_q^{-})[\Omega(\mathbf{x} - \mathbf{w}^{-})]_p\right)\right). \end{aligned} \tag{4.23}$$

In 4.21 - 4.23 we can observe the conventional Hebbian rules of LVQ, according to which, the true-closest code-book vector is to be moved towards the considered data sample while the false-closest prototype is shifted away from it. Therefore, the parameters of the matrix in 4.23 are updated accordingly to squeeze the distance between a current data vector and its nearest prototype and enlarge the gap to the prototype which represents another class.
The learning rates for the code-book vectors and for the metric do not depend on each other, thus, are chosen heuristically.
Note that in order to avoid the degeneration of the algorithm Λ is to be normalized after each adaptation step. In [34] it is achieved enforcing

$$\sum_i \Lambda_{ii} = 1$$

by dividing all components of the matrix Λ by the *raw value* of $\sum_i \Lambda_{ii} = 1$ after each iteration. Thereby, the sum of diagonal elements is fixed and amounts to the corresponding sum of eigenvalues. Thus, one obtains the generalized normalization of relevance $\sum_i \lambda = 1$ for a simple diagonal metric. The eigen directions of Λ can be considered as an interim coordinate system with respect to the relevance which coincide with the corresponding eigenvalues. As

$$\Lambda_{ii} = \sum_k \Omega_{ki} \Omega ki = \sum_k (\Omega_{ki})^2, \tag{4.24}$$

here we maintain normalization by multiplying all components of Ω by $1/\sqrt{\sum_{ki}(\Omega_{ki})^2} = 1/\sqrt{\sum_i [\Omega^T \Omega]_{ii}}$ after each adaptation. This technique is a kind of analogy to a Standard Gradient procedure. More detailed explanation is given in [7]. The approach described by (4.21) -(4.23) approach is termed Generalized Matrix LVQ or in short GMLVQ. In [34] it is shown that this algorithm is faster than unsupervised fuzzy-clustering techniques that apply a similar metric form, but requires a matrix inversion in each iteration. In addition, the metric in GMLVQ is chosen in a supervised manner, such that the optimization of the parameters takes place according to the given classification task.

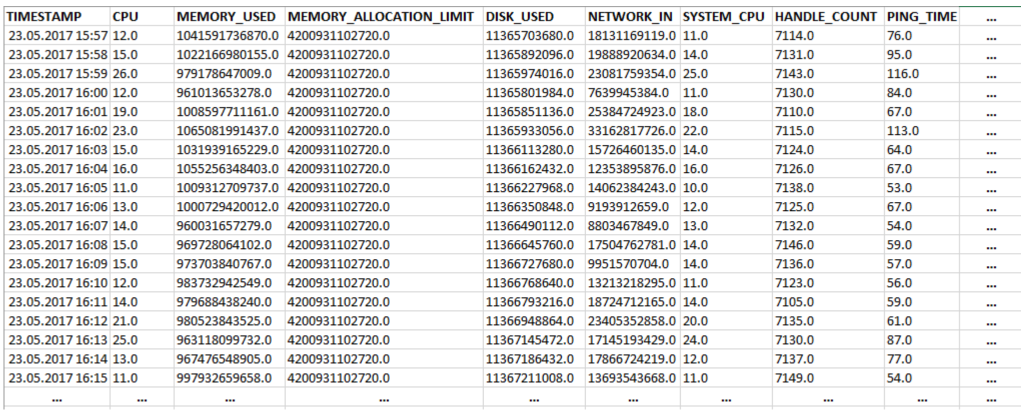
5 Data Preprocessing. Healthy prototypes with the GNG.

This part of the work describes the implementation and evaluation of the approaches from the 3rd and the 4th chapters working on the data taken from SAP IT departments. We designed an anomaly classification model that is a combination of GNG and GMLVQ. Some evaluating and parameter tuning techniques were used during learning, such as random search, k-cross-fold-validation, PCA. The corresponding references are noted as needed, because the additional methods and their understanding effect the accuracy of learning, thus, play a crucial role in the whole model.

The first section runs about the data structure, the second one introduces the created anomaly detection tool. The third section describes the normalization of the given data. The implementation of GNG and GMLVQ, as well as, visualization and evaluation part of the thesis is explained in the last two sections of the chapter. For the data pre-processing steps python frameworks scikit-learn, pyod and pandas were used, the topology of healthy data was maintained in MATLAB with all the necessary references.

5.1 Data set

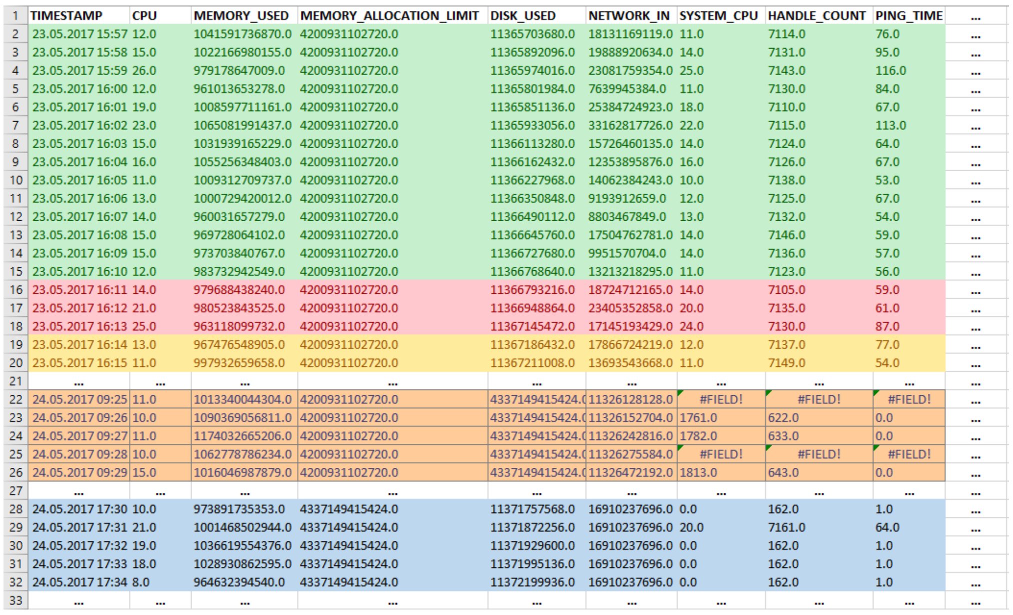
The data of a productive system was derived from SAP IT experts. The data was being collected from May 2017 till October 2018 and represents a time-series with different values of 32 system metrics, see Figure 5.1 for an example fragment of the records. The first column shows the timestamp, the other 32 columns store values for different system attributes such as CPU load, used memory, disk usage, and others.
In a production environment, SAP experts get incidents from end users (clients), where they attach records of the 32 attributes within some time range, claiming that the system stopped working over the mentioned period (incidents). The experts, in turn, look through the values (e.g. CPU, MEMORY_USED, etc.) in order to capture the moment of an anomaly and its root cause.



TIMESTAMP	CPU	MEMORY_USED	MEMORY_ALLOCATION_LIMIT	DISK_USED	NETWORK_IN	SYSTEM_CPU	HANDLE_COUNT	PING_TIME	...
23.05.2017 15:57	12.0	1041591736870.0	4200931102720.0	11365703680.0	18131169119.0	11.0	7114.0	76.0	...
23.05.2017 15:58	15.0	1022166980155.0	4200931102720.0	11365892096.0	19888920634.0	14.0	7131.0	95.0	...
23.05.2017 15:59	26.0	979178647009.0	4200931102720.0	11365974016.0	23081759354.0	25.0	7143.0	116.0	...
23.05.2017 16:00	12.0	961013653278.0	4200931102720.0	11365801984.0	7639945384.0	11.0	7130.0	84.0	...
23.05.2017 16:01	19.0	1008597711161.0	4200931102720.0	11365851136.0	25384724923.0	18.0	7110.0	67.0	...
23.05.2017 16:02	23.0	1065081991437.0	4200931102720.0	11365933056.0	33162817726.0	22.0	7115.0	113.0	...
23.05.2017 16:03	15.0	1031939165229.0	4200931102720.0	11366113280.0	15726460135.0	14.0	7124.0	64.0	...
23.05.2017 16:04	16.0	1055256348403.0	4200931102720.0	11366162432.0	12353895876.0	16.0	7126.0	67.0	...
23.05.2017 16:05	11.0	1009312709737.0	4200931102720.0	11366227968.0	14062384243.0	10.0	7138.0	53.0	...
23.05.2017 16:06	13.0	1000729420012.0	4200931102720.0	11366350848.0	9193912659.0	12.0	7125.0	67.0	...
23.05.2017 16:07	14.0	960031657279.0	4200931102720.0	11366490112.0	8803467849.0	13.0	7132.0	54.0	...
23.05.2017 16:08	15.0	969728064102.0	4200931102720.0	11366645760.0	17504762781.0	14.0	7146.0	59.0	...
23.05.2017 16:09	15.0	973703840767.0	4200931102720.0	11366727680.0	9951570704.0	14.0	7136.0	57.0	...
23.05.2017 16:10	12.0	983732942549.0	4200931102720.0	11366768640.0	13213218295.0	11.0	7123.0	56.0	...
23.05.2017 16:11	14.0	979688438240.0	4200931102720.0	11366793216.0	18724712165.0	14.0	7105.0	59.0	...
23.05.2017 16:12	21.0	980523843525.0	4200931102720.0	11366948864.0	23405352858.0	20.0	7135.0	61.0	...
23.05.2017 16:13	25.0	963118099732.0	4200931102720.0	11367145472.0	17145193429.0	24.0	7130.0	87.0	...
23.05.2017 16:14	13.0	967476548905.0	4200931102720.0	11367186432.0	17866724219.0	12.0	7137.0	77.0	...
23.05.2017 16:15	11.0	997932659658.0	4200931102720.0	11367211008.0	13693543668.0	11.0	7149.0	54.0	...
...	...	...	...	...	...	...	...	...	...

Figure 5.1: A fragment of the Data with 8 out of 32 metrics shown.

In essence , we can distinguish different classes by coloring them and split the records accordingly into so-called *time-windows* like in Figure 5.2. The time range which is marked green mean a *healthy* state, whereas, the other colors indicate anomaly classes in the system behavior.



1	TIMESTAMP	CPU	MEMORY_USED	MEMORY_ALLOCATION_LIMIT	DISK_USED	NETWORK_IN	SYSTEM_CPU	HANDLE_COUNT	PING_TIME	...
2	23.05.2017 15:57	12.0	1041591736870.0	4200931102720.0	11365703680.0	18131169119.0	11.0	7114.0	76.0	...
3	23.05.2017 15:58	15.0	1022166980155.0	4200931102720.0	11365892096.0	19888920634.0	14.0	7131.0	95.0	...
4	23.05.2017 15:59	26.0	979178647009.0	4200931102720.0	11365974016.0	23081759354.0	25.0	7143.0	116.0	...
5	23.05.2017 16:00	12.0	961013653278.0	4200931102720.0	11365801984.0	7639945384.0	11.0	7130.0	84.0	...
6	23.05.2017 16:01	19.0	1008597711161.0	4200931102720.0	11365851136.0	25384724923.0	18.0	7110.0	67.0	...
7	23.05.2017 16:02	23.0	1065081991437.0	4200931102720.0	11365933056.0	33162817726.0	22.0	7115.0	113.0	...
8	23.05.2017 16:03	15.0	1031939165229.0	4200931102720.0	11366113280.0	15726460135.0	14.0	7124.0	64.0	...
9	23.05.2017 16:04	16.0	1055256348403.0	4200931102720.0	11366162432.0	12353895876.0	16.0	7126.0	67.0	...
10	23.05.2017 16:05	11.0	1009312709737.0	4200931102720.0	11366227968.0	14062384243.0	10.0	7138.0	53.0	...
11	23.05.2017 16:06	13.0	1000729420012.0	4200931102720.0	11366350848.0	9193912659.0	12.0	7125.0	67.0	...
12	23.05.2017 16:07	14.0	960031657279.0	4200931102720.0	11366490112.0	8803467849.0	13.0	7132.0	54.0	...
13	23.05.2017 16:08	15.0	969728064102.0	4200931102720.0	11366645760.0	17504762781.0	14.0	7146.0	59.0	...
14	23.05.2017 16:09	15.0	973703840767.0	4200931102720.0	11366727680.0	9951570704.0	14.0	7136.0	57.0	...
15	23.05.2017 16:10	12.0	983732942549.0	4200931102720.0	11366768640.0	13213218295.0	11.0	7123.0	56.0	...
16	23.05.2017 16:11	14.0	979688438240.0	4200931102720.0	11366793216.0	18724712165.0	14.0	7105.0	59.0	...
17	23.05.2017 16:12	21.0	980523843525.0	4200931102720.0	11366948864.0	23405352858.0	20.0	7135.0	61.0	...
18	23.05.2017 16:13	25.0	963118099732.0	4200931102720.0	11367145472.0	17145193429.0	24.0	7130.0	87.0	...
19	23.05.2017 16:14	13.0	967476548905.0	4200931102720.0	11367186432.0	17866724219.0	12.0	7137.0	77.0	...
20	23.05.2017 16:15	11.0	997932659658.0	4200931102720.0	11367211008.0	13693543668.0	11.0	7149.0	54.0	...
21	...	...	...	...	...	...	...	...	...	...
22	24.05.2017 09:25	11.0	1013340044304.0	4200931102720.0	4337149415424.(	11326128128.0	#FIELD!	#FIELD!	#FIELD!	...
23	24.05.2017 09:26	10.0	1090369056811.0	4200931102720.0	4337149415424.(	11326152704.0	1761.0	622.0	0.0	...
24	24.05.2017 09:27	11.0	1174032665206.0	4200931102720.0	4337149415424.(	11326242816.0	1782.0	633.0	0.0	...
25	24.05.2017 09:28	10.0	1062778786234.0	4200931102720.0	4337149415424.(	11326275584.0	#FIELD!	#FIELD!	#FIELD!	...
26	24.05.2017 09:29	15.0	1016046987879.0	4200931102720.0	4337149415424.(	11326472192.0	1813.0	643.0	0.0	...
27	...	...	...	...	...	...	...	...	...	...
28	24.05.2017 17:30	10.0	973891735353.0	4337149415424.0	11371757568.0	16910237696.0	0.0	162.0	1.0	...
29	24.05.2017 17:31	21.0	1001468502944.0	4337149415424.0	11371872256.0	16910237696.0	20.0	7161.0	64.0	...
30	24.05.2017 17:32	19.0	1036619554376.0	4337149415424.0	11371929600.0	16910237696.0	0.0	162.0	1.0	...
31	24.05.2017 17:33	18.0	1028930862595.0	4337149415424.0	11371995136.0	16910237696.0	0.0	162.0	1.0	...
32	24.05.2017 17:34	8.0	964632394540.0	4337149415424.0	11372199936.0	16910237696.0	0.0	162.0	1.0	...
33	...	...	...	...	...	...	...	...	...	...

Figure 5.2: A fragment of a SAP Productive System.

The system state is considered as a *healthy* one if there was no incident and *unhealthy* otherwise. The goal is to create a tool that is capable of distinguishing the states one from another automatically.

5.2 The architecture of the built tool

To design a model for classification task, using LVQ-based ML-algorithms, we need an example of a labeled data set which we can train our model with. See Figure 5.3.

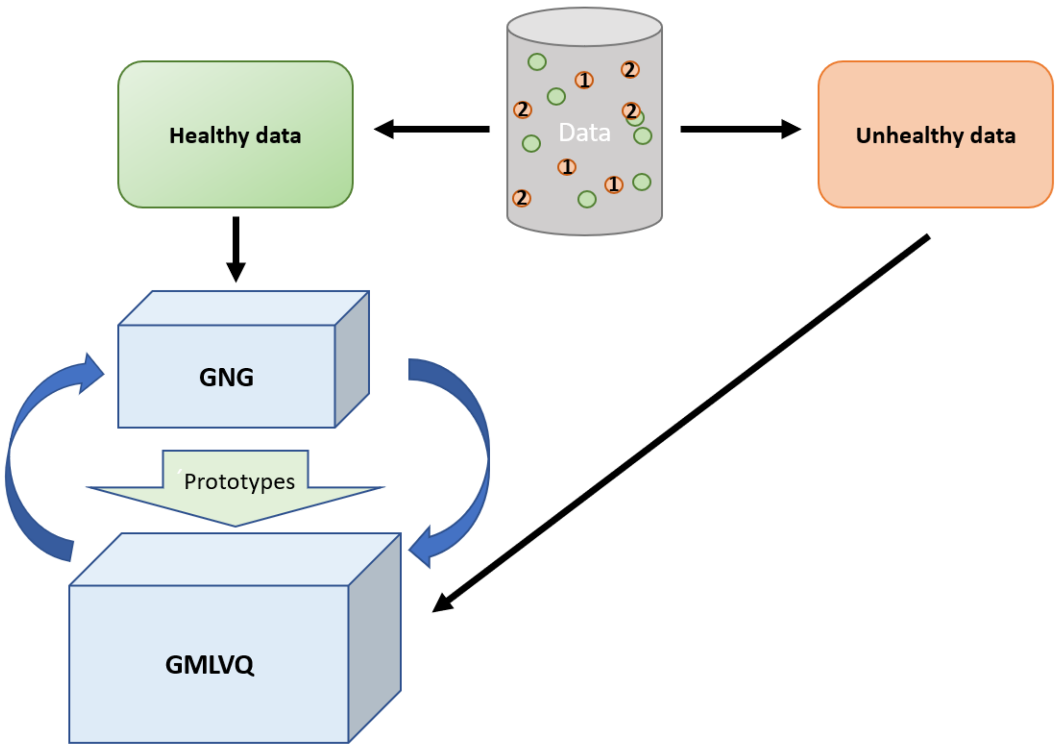


Figure 5.3: The architecture of the tool building.

We perform the following steps:

1. extracting healthy data;
2. learning the topology of the healthy data with GNG;
3a. solving the binary problem with GMLVQ: train the tool to distinguish between abnormal and normal system behavior;
3b. solving the multi-variant problem with GMLVQ: train the model to additionally recognize and cluster anomalies.

The blue arrows in Figure 5.3 mean that we also keep in mind a possible intercommunication between 2 algorithms, for instance, in order to adapt the distance measure that we use in GNG or to see how some parameter tunings might affect the further classification.

In the end, the algorithm takes new unknown input data, and in outputs we get labeled time-windows informing the SAP experts about a class each window belongs to. As it is shown in Figure 5.4.

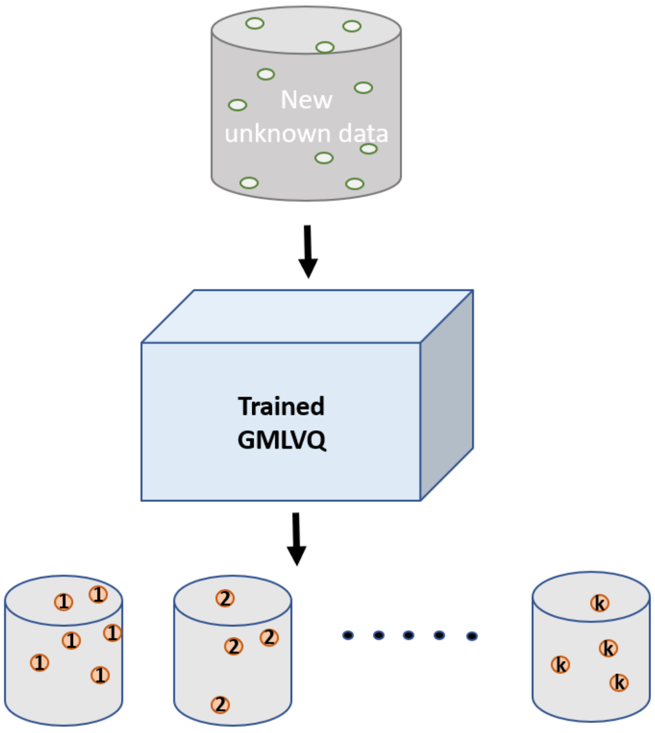


Figure 5.4: Graphical interpretation of the clustering. The output pots represent Y_k.

Here, *'New unknown data'* can be seen as a set

$$\mathrm{X} \in \mathbb{R}^{\mathrm{N} \times 32} : \mathrm{X} = (x_1, x_2, ..., x_n),$$

where $n \in N$ and x_n is an $N \times 32$ matrix. Trained GMLVQ can be seen as a function $F : F(\mathrm{X}) = \mathrm{Y}$, that splits the input set X into clusters:

$$\mathrm{Y} = \mathrm{Y}_1 \bigcup \mathrm{Y}_2 \bigcup ... \bigcup \mathrm{Y}_k$$

(with possible intersections between the subsets) and $k = 1, 2, ..., \mathscr{C}$ - number of clusters,

$$\mathrm{Y}_1 + \mathrm{Y}_2 + ... + \mathrm{Y}_k - \mathrm{Y}_1\mathrm{Y}_2 - \mathrm{Y}_1\mathrm{Y}_3 - ... - \mathrm{Y}_1\mathrm{Y}_k - \mathrm{Y}_2\mathrm{Y}_3 - \mathrm{Y}_2\mathrm{Y}_4 - ... - \mathrm{Y}_{k-1}\mathrm{Y}_k = \mathrm{X}$$

.

5.3 Healthy Data Preprocessing

As it is described in [17], we expect a better performance and more reliable results after normalization of the initial data. Furthermore, in cases when the values for data features belong to different scales, one may end up with inability of such algorithms to learn anything at all out of the data. The term 'normalization' is broad and includes some most common methods, see Figure 5.5.

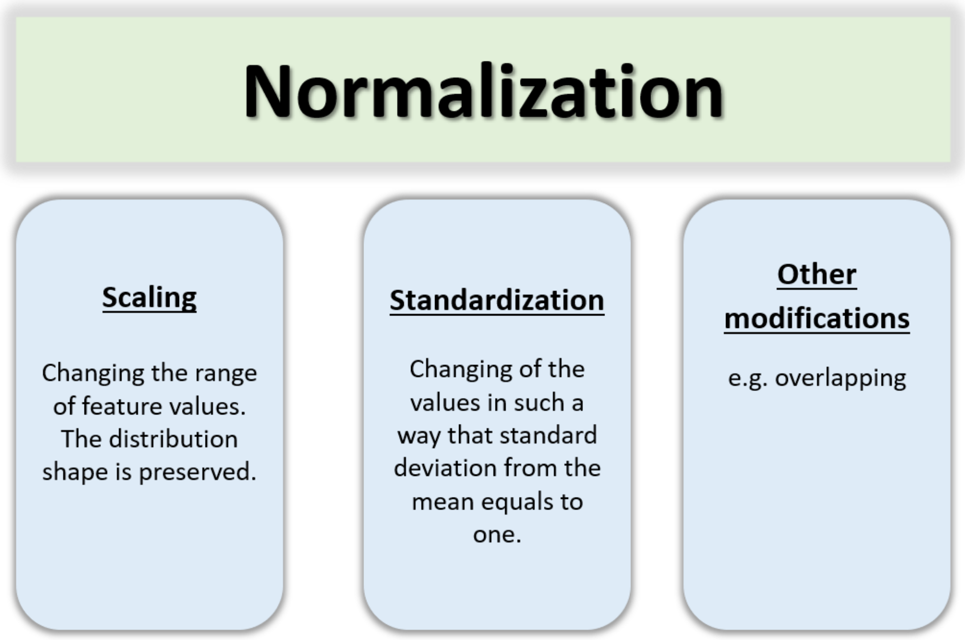

Normalization
Scaling
Changing the range of feature values. The distribution shape is preserved.
Standardization
Changing of the values in such a way that standard deviation from the mean equals to one.
Other modifications
e.g. overlapping

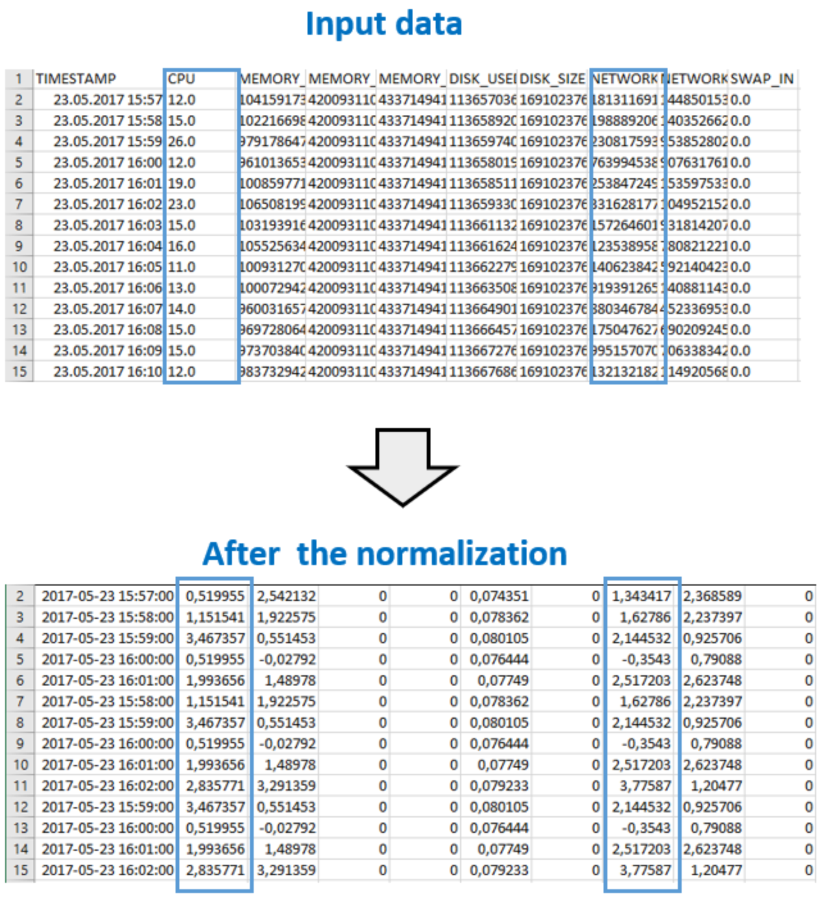
Figure 5.5: The definitions for the most common normalization procedures.

The original data we is represented by values of different units and scales (see Figure 5.1). In order to bring it into a more consistent state and allow by that a better evaluation and data vectors convergence Standard Scaler was chosen with reference to [17] . According to the researches, this scaler tends to lead to the optimization of numerical conditions and has shown reliable results with similar data [16].

Assuming that we have $X = \{x_{ij}\}$, where i – timestamp, j – a feature of the data sample, Standard Scaler is defined as

$$S_j = \frac{x_{ij} - \mu_j}{\sigma_j}, \tag{5.1}$$

where x_{ij} - data vectors, $\mu_j = \frac{1}{n}\sum_{i=1}^{n} x_{ij}$ - mean value and σ_j is the standard deviation of the training samples.


Input data

1	TIMESTAMP	CPU	MEMORY_	MEMORY_	MEMORY_	DISK_USEI	DISK_SIZE	NETWORK	NETWORK	SWAP_IN
2	23.05.2017 15:57	12.0	104159173	420093110	433714941	113657036	169102376	181311691	144850153	0.0
3	23.05.2017 15:58	15.0	102216698	420093110	433714941	113658920	169102376	198889206	140352662	0.0
4	23.05.2017 15:59	26.0	979178647	420093110	433714941	113659740	169102376	230817593	953852802	0.0
5	23.05.2017 16:00	12.0	961013653	420093110	433714941	113658019	169102376	763994538	907631761	0.0
6	23.05.2017 16:01	19.0	100859771	420093110	433714941	113658511	169102376	253847249	153597533	0.0
7	23.05.2017 16:02	23.0	106508199	420093110	433714941	113659330	169102376	331628177	104952152	0.0
8	23.05.2017 16:03	15.0	103193916	420093110	433714941	113661132	169102376	157264601	931814207	0.0
9	23.05.2017 16:04	16.0	105525634	420093110	433714941	113661624	169102376	123538958	780821221	0.0
10	23.05.2017 16:05	11.0	100931270	420093110	433714941	113662279	169102376	140623842	592140423	0.0
11	23.05.2017 16:06	13.0	100072942	420093110	433714941	113663508	169102376	919391265	140881143	0.0
12	23.05.2017 16:07	14.0	960031657	420093110	433714941	113664901	169102376	380346784	452336953	0.0
13	23.05.2017 16:08	15.0	969728064	420093110	433714941	113666457	169102376	175047627	690209245	0.0
14	23.05.2017 16:09	15.0	973703840	420093110	433714941	113667276	169102376	995157070	706338342	0.0
15	23.05.2017 16:10	12.0	983732942	420093110	433714941	113667686	169102376	132132182	114920568	0.0

After the normalization

2	2017-05-23 15:57:00	0,519955	2,542132	0	0	0,074351	0	1,343417	2,368589	0
3	2017-05-23 15:58:00	1,151541	1,922575	0	0	0,078362	0	1,62786	2,237397	0
4	2017-05-23 15:59:00	3,467357	0,551453	0	0	0,080105	0	2,144532	0,925706	0
5	2017-05-23 16:00:00	0,519955	-0,02792	0	0	0,076444	0	-0,3543	0,79088	0
6	2017-05-23 16:01:00	1,993656	1,48978	0	0	0,07749	0	2,517203	2,623748	0
7	2017-05-23 15:58:00	1,151541	1,922575	0	0	0,078362	0	1,62786	2,237397	0
8	2017-05-23 15:59:00	3,467357	0,551453	0	0	0,080105	0	2,144532	0,925706	0
9	2017-05-23 16:00:00	0,519955	-0,02792	0	0	0,076444	0	-0,3543	0,79088	0
10	2017-05-23 16:01:00	1,993656	1,48978	0	0	0,07749	0	2,517203	2,623748	0
11	2017-05-23 16:02:00	2,835771	3,291359	0	0	0,079233	0	3,77587	1,20477	0
12	2017-05-23 15:59:00	3,467357	0,551453	0	0	0,080105	0	2,144532	0,925706	0
13	2017-05-23 16:00:00	0,519955	-0,02792	0	0	0,076444	0	-0,3543	0,79088	0
14	2017-05-23 16:01:00	1,993656	1,48978	0	0	0,07749	0	2,517203	2,623748	0
15	2017-05-23 16:02:00	2,835771	3,291359	0	0	0,079233	0	3,77587	1,20477	0

Figure 5.6: The Standard Scale on the healthy data

The normalization with Standard Scaler results in the distribution with $\mu = 0$ and $\sigma = 1$, thus, the values are on a relatively similar scale (Figure 5.7) and ML algorithms can perform by all means now.

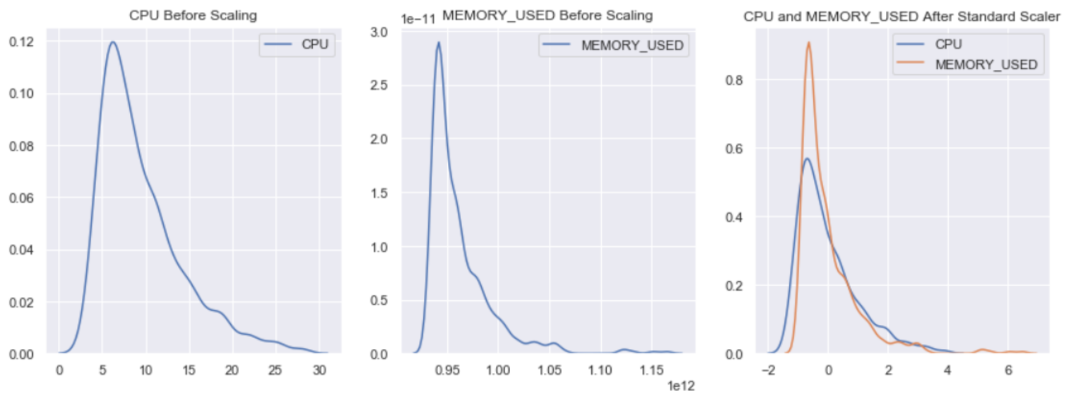


Figure 5.7: The Standard Scale realization in Python

In order to avoid any misunderstanding in further chapters, it is important to keep in mind that the data fragment represented in Figure 5.6 shows an example of *healthy* records. Those were used in GNG to learn the topology and create prototypes of normal data samples. As the records in real life may consist of time gaps and other noise, overlapping was performed as additional data normalization step [18]. This modification allows ML algorithms, GNG for the research, to optimally learn on the data.

5.4 GNG learns Healthy Data

This section describes how the GNG learns the topology of the healthy data which was scaled and normalized as described in the previous section. The main challenge with the given data set is to enable NG to be realized on higher-dimensional input samples. All the previous examples were applied on $1 \times N$ arrays, $N \in \mathbb{R}$ whereas our inputs have a form of $N \times M$, with $M, N \in \mathbb{R}$. To begin with, 2000 normalized records of SAP Productive System were taken and split into 5x32 matrices (time-windows). They became the input samples for GNG.

Let $N = 2000$ be a number of data records, $M = 32$ number of data features, then X is a matrix:

$$X = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ x_{31} & x_{32} & \cdots & x_{3m} \\ x_{41} & x_{42} & \cdots & x_{4m} \\ x_{51} & x_{52} & \cdots & x_{5m} \\ x_{61} & x_{62} & \cdots & x_{6m} \\ \vdots & \ddots & \ddots & \cdots \\ x_{n1} & x_{n2} & \cdots & x_{nm}, \end{pmatrix}$$

where $n \in N$, $m \in M$. Now we split X into input $5 \times m$ matrices $y = (y_1, y_2, ..., y_i), i = \frac{N}{5}, y_i \in R^{5 \times M}$.

$$y_1 = \begin{pmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ x_{31} & x_{32} & \cdots & x_{3m} \\ x_{41} & x_{42} & \cdots & x_{4m} \\ x_{51} & x_{52} & \cdots & x_{5m} \end{pmatrix}, \; y_2 = \begin{pmatrix} x_{21} & x_{22} & \cdots & x_{2m} \\ x_{31} & x_{32} & \cdots & x_{3m} \\ x_{41} & x_{42} & \cdots & x_{4m} \\ x_{51} & x_{52} & \cdots & x_{5m} \\ x_{61} & x_{62} & \cdots & x_{6m} \end{pmatrix},$$

$$, \ldots, \; y_i = \begin{pmatrix} x_{(n-4)1} & x_{(n-4)2} & \cdots & x_{(n-4)m} \\ \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots \\ x_{(n-1)1} & x_{(n-1)2} & \cdots & x_{(n-1)m} \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{pmatrix}$$

and $y \in Y$, $Y \subset \mathbb{R}^n$ is the input space.

Further, we randomly initialize 2 weights (neurons):

$$w_1 = \begin{pmatrix} w^1_{11} & w^1_{12} & \cdots & w^1_{1m} \\ w^1_{21} & w^1_{22} & \cdots & w^1_{2m} \\ w^1_{31} & w^1_{32} & \cdots & w^1_{3m} \\ w^1_{41} & w^1_{42} & \cdots & w^1_{4m} \\ w^1_{51} & w^1_{52} & \cdots & w^1_{5m} \end{pmatrix}, w_2 = \begin{pmatrix} w^2_{11} & w^2_{12} & \cdots & w^2_{1m} \\ w^2_{21} & w^2_{22} & \cdots & w^2_{2m} \\ w^2_{31} & w^2_{32} & \cdots & w^2_{3m} \\ w^2_{41} & w^2_{42} & \cdots & w^2_{4m} \\ w^2_{51} & w^2_{52} & \cdots & w^2_{5m} \end{pmatrix}$$

and after training, obtain a set of N' prototypes of the healthy data Y:

$$\widetilde{w}_1 = \begin{pmatrix} \widetilde{w}^1_{11} & \widetilde{w}^1_{12} & \cdots & \widetilde{w}^1_{1m} \\ \widetilde{w}^1_{21} & \widetilde{w}^1_{22} & \cdots & \widetilde{w}^1_{2m} \\ \widetilde{w}^1_{31} & \widetilde{w}^1_{32} & \cdots & \widetilde{w}^1_{3m} \\ \widetilde{w}^1_{41} & \widetilde{w}^1_{42} & \cdots & \widetilde{w}^1_{4m} \\ \widetilde{w}^1_{51} & \widetilde{w}^1_{52} & \cdots & \widetilde{w}^1_{5m} \end{pmatrix}, \widetilde{w}_2 = \begin{pmatrix} \widetilde{w}^2_{11} & \widetilde{w}^2_{12} & \cdots & \widetilde{w}^2_{1m} \\ \widetilde{w}^2_{21} & \widetilde{w}^2_{22} & \cdots & \widetilde{w}^2_{2m} \\ \widetilde{w}^2_{31} & \widetilde{w}^2_{32} & \cdots & \widetilde{w}^2_{3m} \\ \widetilde{w}^2_{41} & \widetilde{w}^2_{42} & \cdots & \widetilde{w}^2_{4m} \\ \widetilde{w}^2_{51} & \widetilde{w}^2_{52} & \cdots & \widetilde{w}^2_{5m} \end{pmatrix},$$

$$,\ldots, \widetilde{w}_{N'} = \begin{pmatrix} \widetilde{w}^{N'}_{11} & \widetilde{w}^{N'}_{12} & \cdots & \widetilde{w}^{N'}_{1m} \\ \widetilde{w}^{N'}_{21} & \widetilde{w}^{N'}_{22} & \cdots & \widetilde{w}^{N'}_{2m} \\ \widetilde{w}^{N'}_{31} & \widetilde{w}^{N'}_{32} & \cdots & \widetilde{w}^{N'}_{3m} \\ \widetilde{w}^{N'}_{41} & \widetilde{w}^{N'}_{42} & \cdots & \widetilde{w}^{N'}_{4m} \\ \widetilde{w}^{N'}_{51} & \widetilde{w}^{N'}_{52} & \cdots & \widetilde{w}^{N'}_{5m} \end{pmatrix} \tag{5.2}$$

where $w \in \mathbb{R}^{5\times m}$.

The original GNG algorithm requires initialization of 2 learning rates: $\varepsilon_b, \varepsilon_n$ for updating the BMU and its topological neighbors accordingly. In the code which was realized during this thesis, the learning approach was managed by means of Gaussian neighborhood function $h(i,\sigma)$ and the updating concerned all the neurons in the net [10]. The function $h(i,\sigma)$ is defined as

$$h(i,\sigma) = e^{-\frac{winrank(w_i)}{2\sigma^2}}, \tag{5.3}$$

with

$$\sigma = const * \frac{1}{N} \tag{5.4}$$

where N - the number of neurons in the net, $i \in N$, $winrank(w_i)$ returns an array of indices for the neurons according to the dissimilarity measure. For instance, let us assume that we have 5 neurons and one data sample $y \in Y$, where Y is the input space, see Figure 5.8.

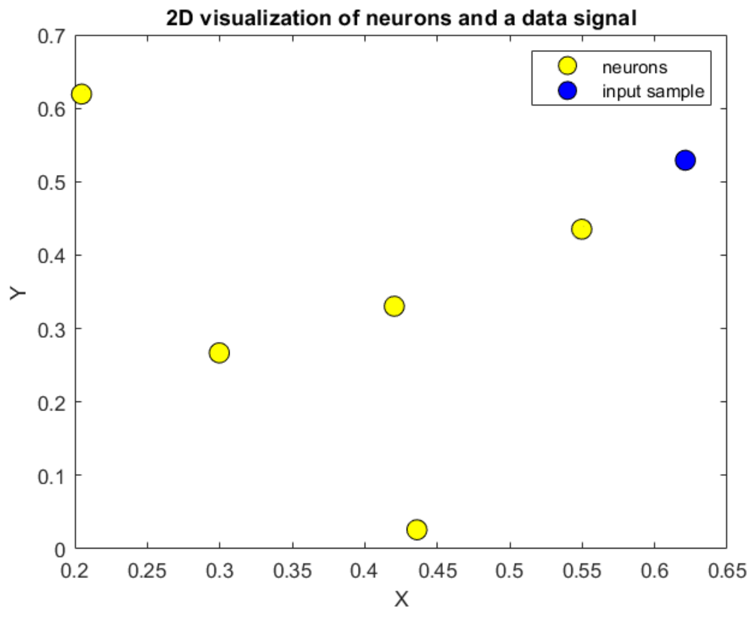


Figure 5.8: A fragment of the neural net with 5 neurons - the yellow dots, and one data sample - the blue dot.

Then we calculate the distances between the neurons and a given data signal, so we can get the following table:

neuron	w_1	w_2	w_3	w_4	w_5
dissimilarity measure	$d(y,w_1)$	$d(y,w_2)$	$d(y,w_3)$	$d(y,w_4)$	$d(y,w_5)$
$winrank$	5	1	2	4	3

Table 5.1: The sorting function result for 5 neurons. Here, neuron w_2 is the BMU, as its winning rank is 1.

In the Figure 5.9 one can observe graphical visualization of $h(i,\sigma)$ versus the result of the sorting function.

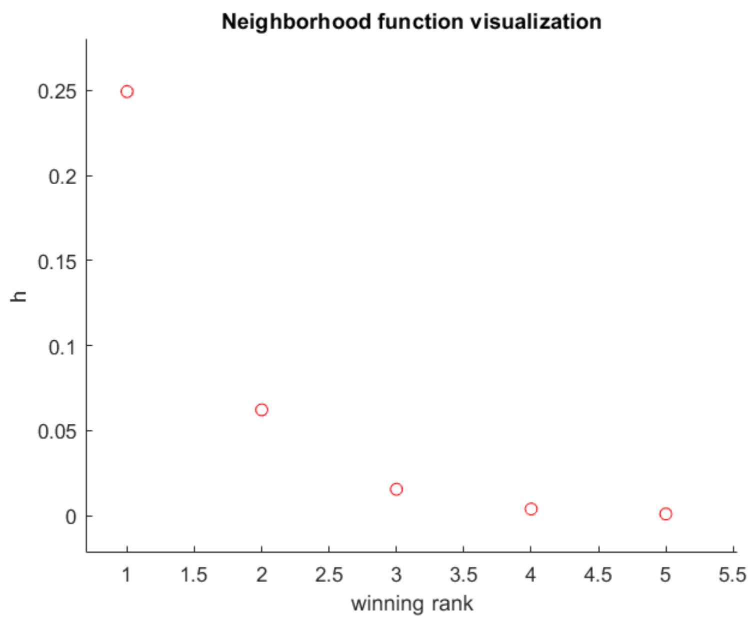
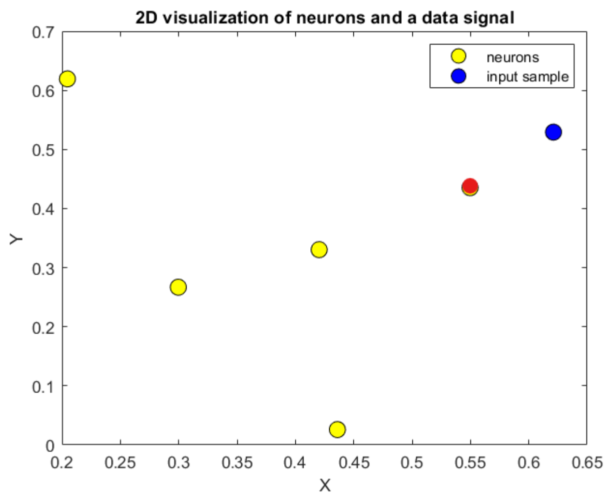
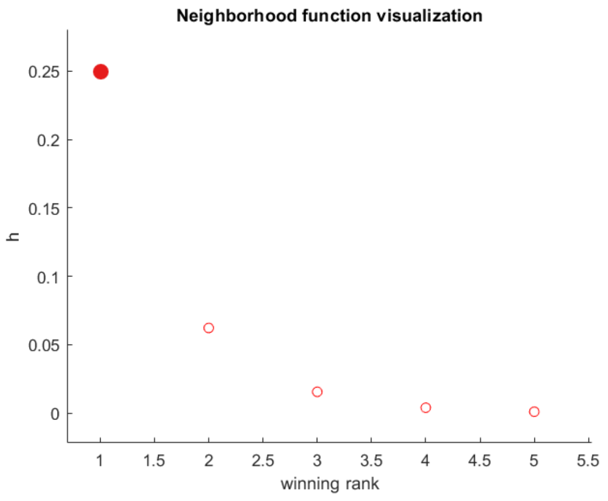


Figure 5.9: Visualization of the adaptation strength for the neurons versus their winning rates



(a) The BMU is marked with red



(b) The value of the BMU multiplier during the updating in (5.4).

Figure 5.10: (a)The neurons position with respect to the input signal. (b)The example of the updating strength for the neurons (marked with red for BMU)

After $h(i,\sigma)$ apply, one can update the neurons with regards to their winning rates.

$$\Delta w_i = \varepsilon_b * h(i,\sigma) * (y - w_i), \tag{5.5}$$

The neuron with rank 1 (BMU) get heavily update, while the shifting of weight with rank 5 is close to zero.

Obviously, the adaptation affects all the neurons that have been created on the net. That not only has led to faster learning but also eliminated the need to tune the additional hyper-parameter ε_n responsible for the adaptation strength of topological neighbors of the BMU.

The programming implementation is performed in Matlab and based on the code 'Realization of Neural Gas network and Growing Neural Gas (GNG) network in MATLAB' taken from *"The Yarpiz Project"* [27].

5.4.1 Evaluation of GNG

In order to achieve the reliable results of the GNG performance one should carefully allocate the hyper-parameters and monitor the error behavior on the flow. For this reason we used a validation technique which allows to calculate the optimal values for the parameters as well as to moderate the magnitude of the error function. The calculation process of the optimal hyper-parameters is explained below and was implemented in MATLAB.

We proceed by splitting the input data set $Y \in \mathbb{R}^n$ into a training and a test subsets Y_{train}, Y_{test} the way that $Y_{train} \cup Y_{test} = Y$, $Y_{train} \cap Y_{test} = \emptyset$.

Running GNG on Y_{train} one gets a set of prototypes $W_{train} = w_1, ...w_n$, $n \leq N$, N is maximal number of neurons in the net. Each $w_i \in W_{train}$ stores its accumulated error $acc_{train}(w_i)$:

$$acc_{train}(w_i) = \Delta E(w_i), \tag{5.6}$$

where ΔE is calculated as it is explained in 3.5. For further validating procedure we also calculate the mean error for the whole set W_{train} as it follows:

$$MeanE_{train} = avg(acc_{train}(w_i)) = \frac{1}{n}\sum_{i}^{n} acc_{train}(w_i). \tag{5.7}$$

Then we consider test data samples $y_s \in Y_{test}$, $s \in S$, where S is a number of elements in the test set. For calculating the mean error for Y_{test} we use the nodes $w_i \in W_{train}$ and compute mean test error according to the following sequence of equations:

$$acc_{test}(w_i) = \sum_{i=1}^{n} \operatorname*{argmin}_{i}\{d(y_s, w_i) | s \in 1, ..., S\}, \tag{5.8}$$

$$MeanE_{test} = avg(acc_{test}(w_i)) = \frac{1}{n}\sum_{i}^{n} acc_{test}(w_i). \tag{5.9}$$

Now, we split the given data Y into train and test subsets using k-cross-fold validation. According to the approach, the data is first partitioned into k equally (or nearly equally) sized segments or folds. Subsequently k iterations of training and validation are performed such that within each iteration a different fold of the data is held-out for validation while the remaining $k-1$ folds are used for learning [30]. Let us split the input data Y into f-folds containing a unique pair of train and test subsets in each f as following

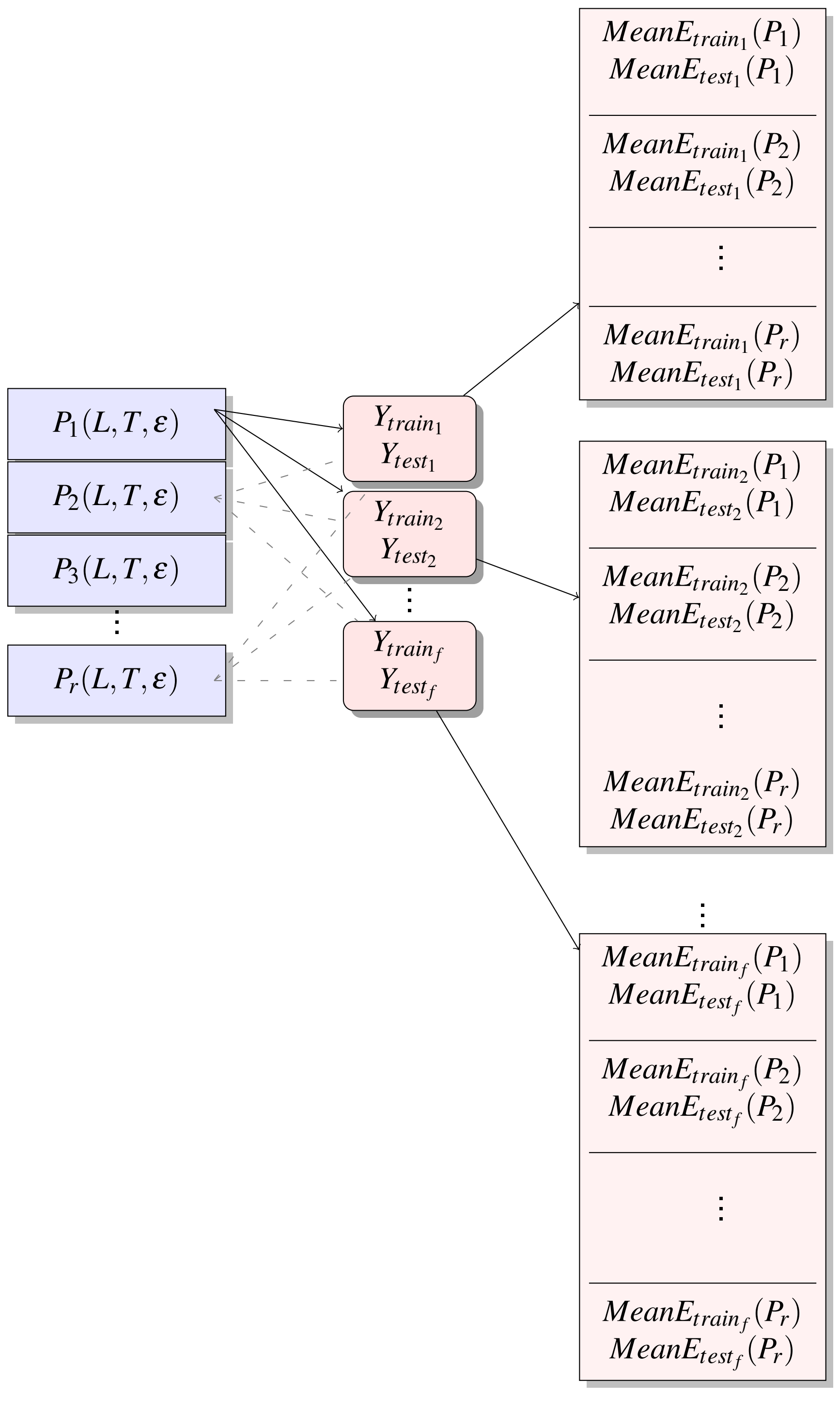
$$Y = (Y_1, Y_2, \dots, Y_f),$$

where f is the number of folds and

$$Y_i = Y_{train_i} \cup Y_{test_i},$$

$$Y = \cup Y_i \text{ and } \cap Y_i = \emptyset.$$

The next step consists of choosing reasonable values for the hyper-parameters $P(N, MaxIt, L, T, \varepsilon, \alpha, \delta)$. Creating representatives of the input signals means a compression. Under compression we understand to take not more than 10% of the given data; having that in mind, 400 input data vectors are taken along with a prototype set, where $N = 40$. Maximum number of iteration is first deliberately set to a higher value and then experimentally can be narrowed down or expanded so that the stable behavior of the mean error can be achieved after a particular iteration. In this work we allocate $MaxIt = 100$, but we can observe that after the 20th circle of running the error value does not change or even slightly goes up. Thus, we set the final value for the hyper-parameter equal to 25, so that the error drop to its minimum can be seen. The moderators of local and global errors do not affect the performance in core, rather ease the computational time, so the values for them are normally chosen in the interval between 0 and 1 (in this thesis: $\alpha = 0.5, \delta = 0.005$) following the example in [36]. In order to optimize L, T and ε we used the combination of k-cross validation[30] and *Random Search* [6] approaches, see Figure 5.11.



$P_1(L,T,\varepsilon)$
$P_2(L,T,\varepsilon)$
$P_3(L,T,\varepsilon)$
$P_r(L,T,\varepsilon)$
Y_{train_1} Y_{test_1}
Y_{train_2} Y_{test_2}
Y_{train_f} Y_{test_f}
$MeanE_{train_1}(P_1)$ $MeanE_{test_1}(P_1)$
$MeanE_{train_1}(P_2)$ $MeanE_{test_1}(P_2)$
$MeanE_{train_1}(P_r)$ $MeanE_{test_1}(P_r)$
$MeanE_{train_2}(P_1)$ $MeanE_{test_2}(P_1)$
$MeanE_{train_2}(P_2)$ $MeanE_{test_2}(P_2)$
$MeanE_{train_2}(P_r)$ $MeanE_{test_2}(P_r)$
$MeanE_{train_f}(P_1)$ $MeanE_{test_f}(P_1)$
$MeanE_{train_f}(P_2)$ $MeanE_{test_f}(P_2)$
$MeanE_{train_f}(P_r)$ $MeanE_{test_f}(P_r)$

Figure 5.11: In the purple boxes there are $P_r(L,T,\varepsilon)$ ***r*** different sets of hyper-parameters depicted; Each set is tried out in each ***f*** fold of different Y_{train}, Y_{test} subsets; for each fold one gets ***r*** pairs of mean errors for Y_{train} and Y_{test}.

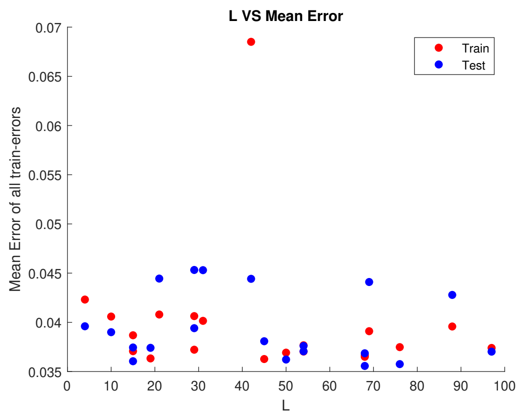
With Random Search we generate r different 3-tuples of hyper-parameters $P_k = \{P_k(L,T,\varepsilon)\}$ and run the GNG with each P_k for $k = 1,...,r$ and every f fold, as it is shown in Figure 5.11. As a result we will obtain r mean train- and test- errors for each train-/test- subset. The errors are computed according to 5.6 and 5.8. For P_i, where $i = 1, ..., r$ the total evaluation error is computed as following:

$$\begin{aligned}
TotalError_{train}(P_1) &= \frac{1}{f}\sum_{i=1}^{f} MeanE_{train_i}(P_1) \\
TotalError_{train}(P_2) &= \frac{1}{f}\sum_{i=1}^{f} MeanE_{train_i}(P_2) \\
TotalError_{train}(P_3) &= \frac{1}{f}\sum_{i=1}^{f} MeanE_{train_i}(P_3) \\
&\vdots \\
TotalError_{train}(P_r) &= \frac{1}{f}\sum_{i=1}^{f} MeanE_{train_i}(P_r)
\end{aligned} \tag{5.10}$$

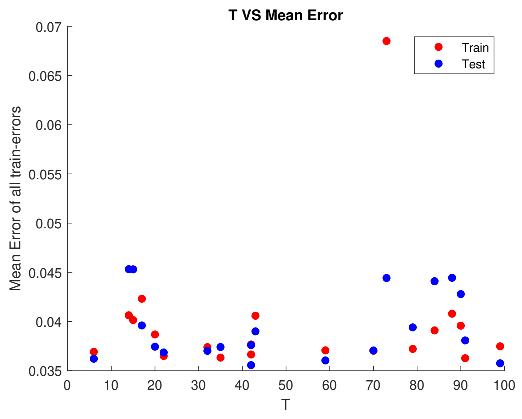
In the same way we compute the total error for test sets summing up all results for the test error in each fold, obtaining :

$$\begin{aligned}
TotalError_{test}(P_1) &= \frac{1}{f}\sum_{i=1}^{f} MeanE_{test_i}(P_1) \\
TotalError_{test}(P_2) &= \frac{1}{f}\sum_{i=1}^{f} MeanE_{test_i}(P_2) \\
TotalError_{test}(P_3) &= \frac{1}{f}\sum_{i=1}^{f} MeanE_{test_i}(P_3) \\
&\vdots \\
TotalError_{test}(P_r) &= \frac{1}{f}\sum_{i=1}^{f} MeanE_{test_i}(P_r)
\end{aligned} \tag{5.11}$$

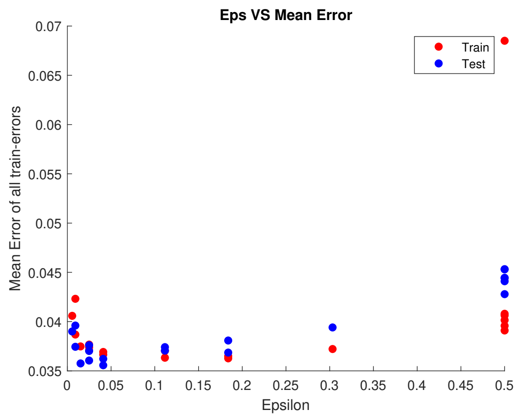
The visualization of the values of L, T, ε against the total train and total test errors is given in Figure 5.15. And there we can capture a dominating impact on the error behavior of the learning rate ε, that is why first, the interval for this hyper-parameter is tuned further, narrowing down the interval from which we choose the value for ε : Figure 5.13. Then fixing ε: $\varepsilon = const$, where $const \in (0.015; 0.035)$ we tune the values for the parameters L and T. Thus,experimentally shown that the values of the parameters do not influence the total error more than on 0.002 and can be chosen randomly without loss of the performance results (for further learning we choose T = 50, L = 50), see Figure 5.14.



(a) L against $TotalErroe_{train}$ and $TotalError_{test}$



(b) T against $TotalErroe_{train}$ and $TotalError_{test}$



(c) ε against $TotalError_{train}$ and $TotalErroe_{test}$

Figure 5.12: Hyperparameters values against $TotalError_{train}(P_i)$ and $TotalError_{test}(P_i)$, $i = 1, ..., r$.

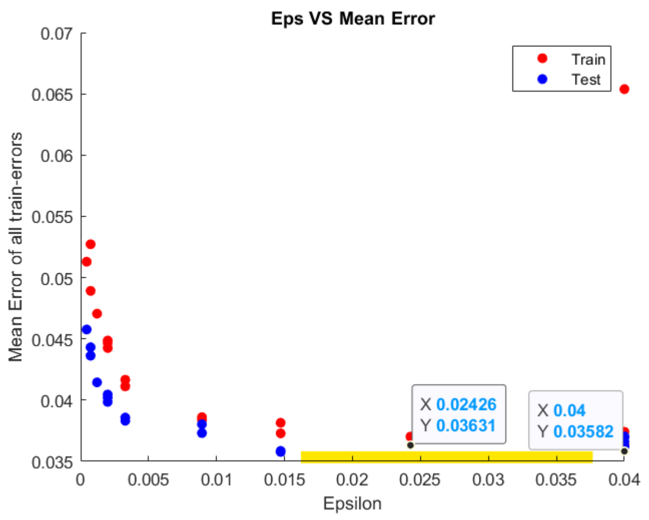
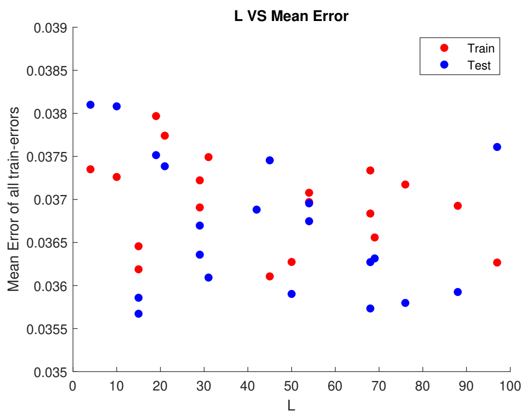
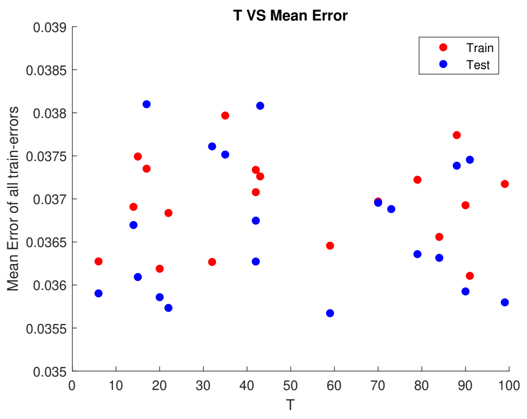


Figure 5.13: ε tuning in a smaller interval

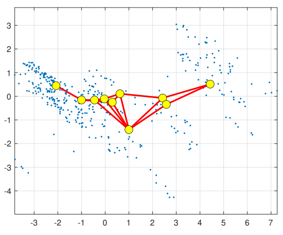


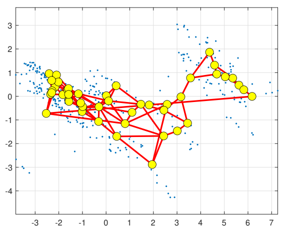
(a) L

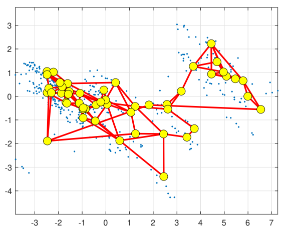


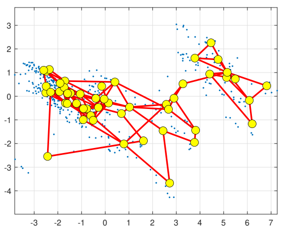
(b) T

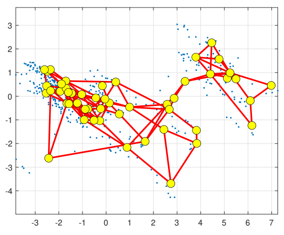
Figure 5.14: Error behavior against different L and T with fixed $\varepsilon = 0.035$


(a) $MaxIt = 25$


(b) $MaxIt = 50$


(c) $MaxIt = 75$


(d) $MaxIt = 100$


(e) $MaxIt = 125$

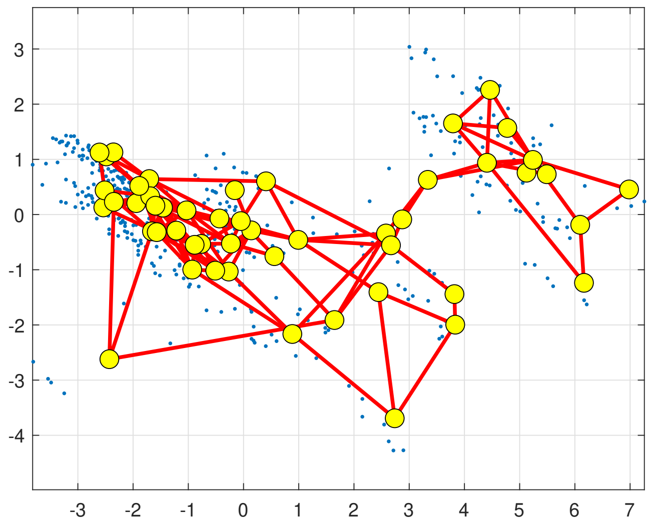

(f) $MaxIt = 150$

Figure 5.15: Visualization of the GNG after different number of iterations. $P(N = 40, L = 50, T = 50, \varepsilon = 0.035, \alpha = 0.5, \delta = 0.05, MaxIt = \{25; 50; 75; 100; 125; 150\})$

After all the hyper parameters are defined one can run the GNG and tune N and $MaxIt$ achieving the minimum mean error. For this master thesis the parameters are allocated as in Figure 5.15. The yellow dots illustrate the neurons (prototypes) of the healthy data signals (the blue dots). As the both input vectors and weights are of forms of 5x32 matrices, the visualization on 2D space was handled with a hand of the Principal Component Analysis and performed in MATLAB. In the end, we got the representation of healthy data, that can we use for GMLVQ classification in the following chapter.

6 GMLVQ Classification of SAP IT Data

After retrieving the prototypes of healthy data with GNG, we apply GMLVQ classifier with the goal to find out classification accuracy of the whole anomaly-detection-root-cause-identification model (see Figure 5.4).
The structure of the chapter: in section 6.1 the data preparation for solving two- and four- class problems is described; section 6.2 illustrates the achieved results of the GMLVQ execution. All the experiments and visualizations were accomplished in MATLAB R2019a environment and all the necessary references to additional programming tools and functions were meticulously made.

6.1 The GMLVQ experimental workflow

This section describes the preprocessing steps of the input data. According to the data provided by SAP experts, at least 3 different types of anomalies can be detected during system workflow:

1. Anomalies that drive to the high CPU load (A_1).
2. Anomalies associated with the amount of HANA thread handles (A_2), see [14].
3. Anomalies causing CPU downtime (A_3).

Some manually categorized and labeled outlying data records were collected and preprocessed in the same manner as the healthy input set described in Section 5.3. Forming the data space for the GMLVQ we consider a $Z \in \mathbb{R}^{N \times P}$ matrix with P - the number of features, N - the number of input samples such that $N = H \cup A_1 \cup A_2 \cup A_3$ with H, A_1, A_2, A_3 representing the numbers of prototypes of healthy and anomaly classes of type one, two and three accordingly. Each data sample is represented by a vector $z = (z_1, ..., z_P)$, an array that is formed from a time-window input sample that had a form of a 5-by-32 matrix:

$$z_k = \begin{pmatrix} z_{11}^k & z_{12}^k & \cdots & z_{1m}^k \\ z_{21}^k & z_{22}^k & \cdots & z_{2m}^k \\ z_{31}^k & z_{32}^k & \cdots & z_{3m}^k \\ z_{41}^k & z_{42}^k & \cdots & z_{4m}^k \\ z_{51}^k & z_{52}^k & \cdots & z_{5m}^k \end{pmatrix}, z_k = \begin{pmatrix} z_{11}^k & z_{12}^k & \cdots & z_{1m}^k & z_{21}^k & \cdots & z_{5m}^k \end{pmatrix}$$

In other words, if we take as an input a healthy prototype from 5.2 and line it up, we will get a healthy input sample $\widetilde{w}_{N'}$ in a form of vector:

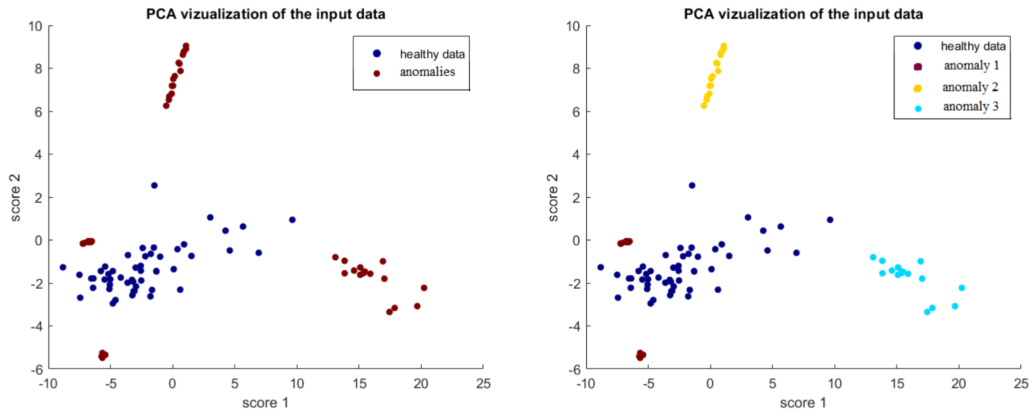
$$z_i = \widetilde{w}_{N'} = \begin{pmatrix} \widetilde{w}_{11}^{N'} & \widetilde{w}_{12}^{N'} & \dots & \widetilde{w}_{1m}^{N'} & \widetilde{w}_{21}^{N'} & \dots & \widetilde{w}_{5m}^{N'} \end{pmatrix} \tag{6.1}$$

and this vector represents a *healthy* input data sample for the GMLVQ. We also define the set of labels $\mathscr{L} := \{1,...,l\}$ whereby each label $l \in \mathscr{L}$ corresponds to a particular behavior of the system (outlying or not) as described above. For instance, here we have $l = 4$ for a healthy type of data samples and 3 anomalous ones.
Based on the conventions the classification task is to determine a classifier function

$$c : Z \to \mathscr{L} : c(z) \to l$$

such that a sample z is assigned in this way to a certain anomaly or to a normal state.

The generated set of whole data consists of 100 vectors with a label set $\mathscr{L} = 1,2,3,4$. For GMLVQ classifier, we run binary and four-class classification. The binary classification involves comparing the distinct anomalous groups (A_1, A_2, A_3) with the healthy group (the prototypes retrieved with GNG). The multi-class classification concerns the comparison of all the groups H versus A_1 versus A_2 versus A_3. The goal is to determine the class membership (healthy or unhealthy) of an unknown input subject and also determine the type of anomaly. To this extend we take 50 healthy and 50 unhealthy samples into consideration. The 50 samples of unhealthy data vectors consist of 15 samples from A_1, 15 - from A_2 and 20 -from A_3. A PCA-projection of the original data is shown in Figure 6.1.

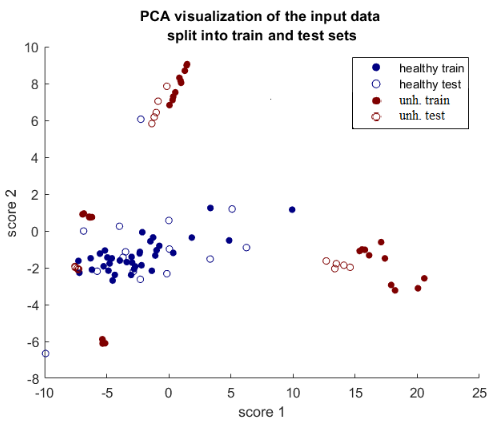


(a) The visualization of the input data for binary classification (b) The visualization of the input data for multivariate classification

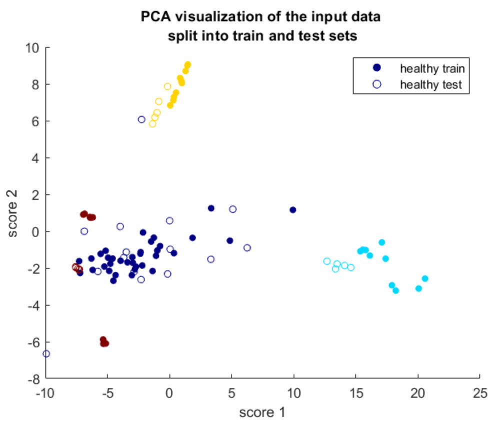
Figure 6.1: The input data for clustering

In order to outline a classification system based on the GMLVQ approach one has first to generate a training data set and subsequently to train the model. In this phase, a set of appropriately chosen prototypes is computed from a given set of labeled example data. Relevance learning provides insight into the data in terms of weighting features and combination of features in the adaptive distance measure [28].

In all the further presented experiments the data was split into train and test subsets with 70% to 30% proportions. The evaluation of the classification algorithm was based on 10-folds-cross-validation results. An example of the original data before classification in 2-dimensions is depicted in Figure 6.2. Training and test sets consist of 50 data points per class. In order to avoid the 'luckily-happened' outcomes, the experiments are executed on 10 statistically independent subsets. One of these data subsets is depicted in Figure 6.2. In the following demonstrations the mean results are being illustrated for a binary and multi classification problems, see in Figures 6.3-6.6. The visualization was based on M.Biehl's MATLAB script [8].



(a) For binary classification problem



(b) For multivariate classification problem

Figure 6.2: An example split of input data into train- and test-subsets.

For each run a part of data (test set) is left out and GMLVQ is performed on the rest of the data (training set). Then the test data samples are used to evaluate the trained classifier. The *sensitivity* (TP rate), *specificity* (TN rate) and classifier accuracy are computed. In addition, ROC curve and Nearest Prototype Classifier confusion matrix are calculated for all the test sets [28].

The GMLVQ optimization tool requires to specify a number of prototypes per class. This hyper-parameter depends on the number of modes of the underlying class distribution. For our experiments, one prototype per class turned out to be sufficient to achieve reliable outcomes, but, obviously, increasing the number of prototypes in each class tends to improve the classification accuracy. It is manually tuned according to the allocation of input data. For instance, based on this thesis research and the data visualization (Figure 6.3), vectors $z \in A3$ (class 4 marked in wine-red color) tend to be sparse and easily confused with healthy state, that is why more than one prototype for the mentioned class is recommended especial in the case of the amount of data to be classified

in real production environment.
A global quadratic distance measure of the form $d(w_k, z) = (z - w_k)^T \Lambda (z - w_k)$ is used to quantify the dissimilarity of an input vector z and the prototypes. The measure is parameterized in terms of the positive semi-definite Λ [28]. Relevance matrix and prototypes are optimized in the training process which is guided by a suitable cost function [28]. The GMLVQ implementation that we use realizes a batch gradient descent minimization with automated step size control, see [5] for details. After 60 steps of gradient descent, the training errors and cost function appeared to have converged in all considered classification problems.

6.2 Results

In this section we present the results after 10-folds CV for distinct anomalous groups against the healthy group in the two- and four-class classification. Furthermore, we present the percentage of correctly classified data samples (sensitivity) and the percentage of correctly classified healthy controls (specificity), as well as AUC and ROC curve and correctness of overall labeling (accuracy). Additionally, the corresponding results are visualized in terms of projections on the leading two eigenvectors of the relevance matrix. This exploits the fact that GMLVQ shows a tendency to yield low-rank matrices which correspond to low-dimensional representation of the feature space [28]. We also provide the plots of diagonal and off-diagonal matrix elements as an example illustration.

6.2.1 Binary Classification

The objective is to observe how precise the separability is between the healthy data samples and the anomalous groups. The results after GMLVQ are presented in Table 6.1.

Data Set (size)	Accuracy (%)	Sensitivity (%)	Specificity (%)	AUC
A1-H (30)	98	100	96	0.98
A2-H (30)	96	100	94	0.99
A3-H (40)	83	76	96	0.84

Table 6.1: GMLVQ Classifier performance after 10-folds CV for the different data set combinations (healthy data samples against anomalies of three different types, number of samples in brackets)

In the Table above *Accuracy* indicates the percentage of correctly classified samples in each group. All three measures here (*Accuracy, Sensitivity, Specificity*) correspond to the Nearest Prototype Classifier (NPC). As it is observed, AUC measures tend to be relatively high, meaning that that GMLVQ weighted features are very suitable for separability of the groups.

Further, we run the algorithms giving as input all the healthy and unhealthy data samples together with a goal to achieve a separability of the healthy data signals from all the others. The computed results are presented in the form of the confusion matrix below, off-diagonal relevance matrix elements (Figure 6.3) and visualization of projected labeled data signals after GMLVQ and ROC-AUC (Figure 6.4).

$$ConfusionMatrix = \begin{pmatrix} 100 & 0 \\ 18.2846 & 81.7154 \end{pmatrix} \tag{6.2}$$

According to the matrix above, all healthy data samples were classified correctly, unfortunately, we still see the tendency to assign some anomalous samples to the healthy group as well. This means that the training procedure should continue, probably knowing the importance of the features (see Figure 6.5), we could tune the training the way to low miss-classification rate.

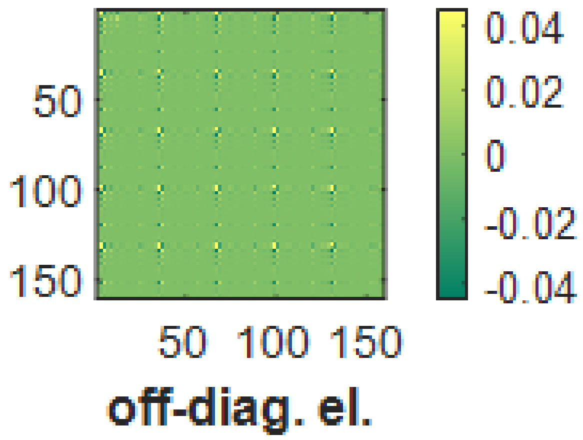


Figure 6.3: Off-diagonal relevance matrix elements

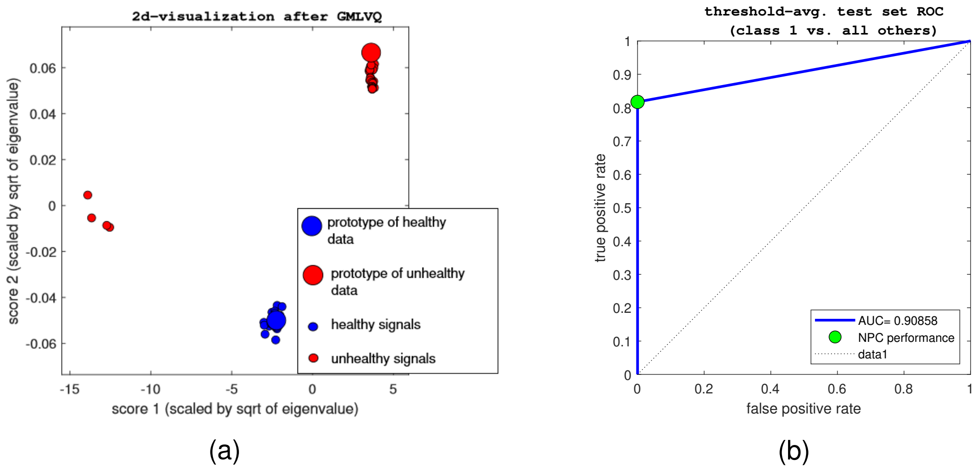


Figure 6.4: (a)The visualization of the training data in terms of their projection on the two leading eigenvectors of the relevance matrix. (training over 10 validation runs with random 30% of samples left out for testing and 60 gradient descent steps). (b) ROC-AUC

The projecting relevance vectors in Figure 6.5 indicate that mostly three data features significantly influence the classification: current volume of disk usage, memory consumption by a service and number of open handles (see in Appendix A). Furthermore, a relevance vectors are included in the distance measurement to scale the input dimensions according to their importance with respect to the classification task. This not only might boost the classification performance, but may also be used for feature selection and dimensional reduction [28].

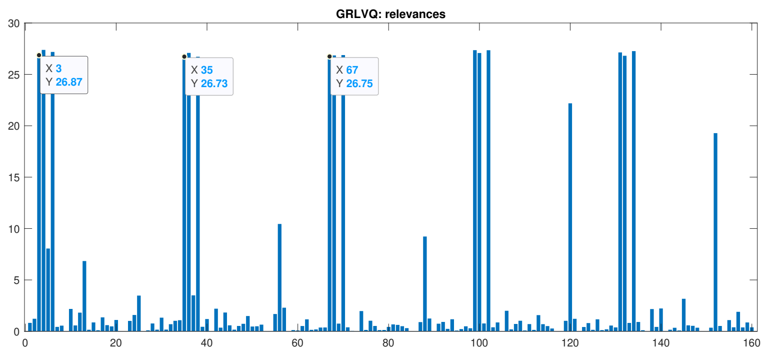


Figure 6.5: The importance of the data features according to GRLVQ. The features peak repeatedly since the time-windows were flattened into arrays. E.g. Features 3, 35, 67,... represent disk usage

As observed in Table 6.2, the comparison of healthy signals against different anomalies shows a clear separation between healthy and unhealthy groups. We can clearly see the separability in Figure 6.4(a). Apart from a few outliers, most of the data signals cluster around the specific prototypes. As well as, the histogram of the relevance matrix in Figure 6.5 illustrates the features and their diagonal weights as used in the classification.

Further we could analyze this information, for example, what features are weighted the highest, meaning they carry relevant information that is important for the separability. In fact, we should treat such features with more attention, as an idea, critically analyze the principal component image corresponding to this feature to gain insights from the system perspective [28].

6.2.2 Multi-class Classification

The objective here is to present the results after 10-folds CV of the GMLVQ classifier on the four classes: labeling healthy class and all the three types of anomalies. We run the GMLVQ with all the anomalous data sets together because we want to be able to distinguish the three types from each other with help of the classifier. In addition, we include healthy part of data samples because we still need to distinguish a healthy sample from any of the other groups. The results are summarized in Table 6.2, confusion matrix. As well as, in Figure 6.6 we depict the scatter plots that shows the training data signals distribution in the two-dimensional projection of the feature space in a single run of the training procedure.

GMLVQ classification	H	A_1	A_2	A_3
H (50)	**50**	0	0	0
A_1(15)	0	**14**	0	1
A_2(15)	0	0	**15**	0
A_3(20)	6	1	0	**13**
Class accuracy (%)	100	93.3	100	65
Overall performance (%)	89.5			

Table 6.2: The table illustrates the number of subjects correctly classified for each class in bold and the overall performance in percentage as obtained in the CV

The confusion matrix below carries illustrative purpose to show the mean classification results after 10-folds CV runs.

$$ConfusionMatrix = \begin{pmatrix} 100 & 0 & 0 & 0 \\ 0 & 98.5714 & 0 & 1.4286 \\ 0 & 0 & 100 & 0 \\ 34.6667 & 2 & 0 & 63.3333 \end{pmatrix} \tag{6.3}$$

We can see that healthy group and unhealthy group of type 1 have been correctly classified with accuracy 100%. None of the anomalous group A_1, A_2 were classified as healthy ones. However, we observe about 34% of miss-classification of A_3-data samples.

The scatter plot in Figure 6.6 (a) show the training data points with respect to their projections on the two leading eigenvectors of the relevance matrix. It can be observed that all four groups a clearly separable from each other. There is no overlapping shown, $AUC = 0.92$, all healthy samples are classified correctly. This is encouraging since we are generally interested in distinguishing between healthy and any types of unhealthy data samples.

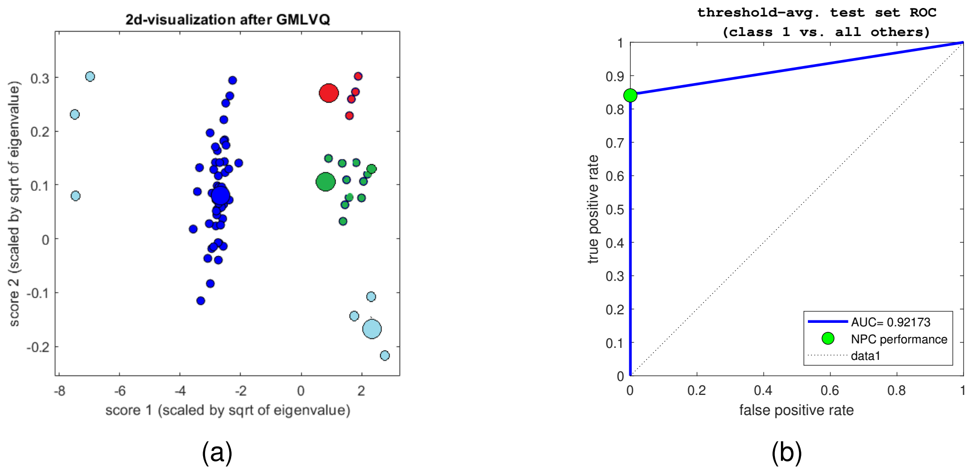


Figure 6.6: 2D performance of training over 10 validation runs with random 30% of samples left out for testing and 60 gradient descent steps. blue-healthy, red-A_1, green-A_2, light-blue-A_3

To sum up, we also depict the evolution in terms of mean errors in Figure 6.7.

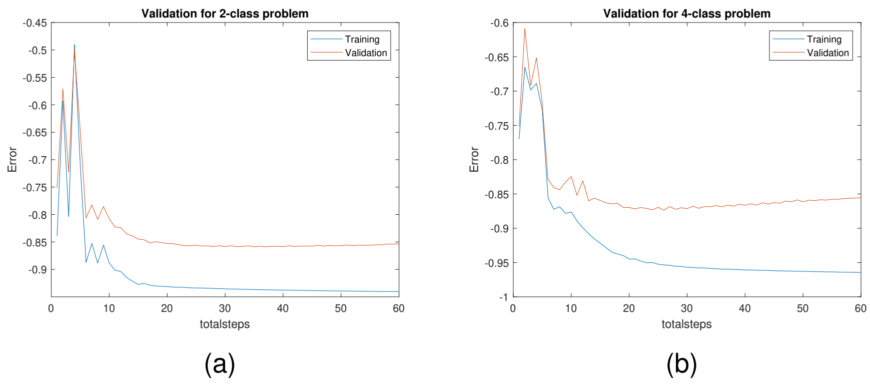


Figure 6.7: Evolution of the mean training and test error in the course of GMLVQ-Training

In preliminary experiments performed in this thesis, the GMLVQ model managed to distinguish all healthy samples correctly and also demonstrated a good separability among outliers. The whole system can be further tuned to achieve higher results in order to minimize anomaly miss-classification.

7 Conclusion and future research

In this Master thesis we set up a joint model of the Growing Neural Gas and the Generalized Matrix Learning Vector Quantization. Then, we investigated its application for anomaly detection process in an SAP productive system. As the data that was given for investigation consists of many healthy data records and comparably fewer records of anomalies, we applied GNG to obtain a generalized representation of the healthy data that further is used as signals in the healthy group in the GMLVQ classifier. There was also created a special form of the Growing Neural Gas Algorithm that allows one not only to perform learning straight on matrix-structure inputs, but also to automatically calculate the suitable hyper-parameters. This pioneer enforcement helps to avoid some data information loss and save time that one normally spends on tuning the model. Furthermore, to distinguish the healthy prototypes from different unhealthy data signals we performed the GMLVQ. The preliminary experiments on a single SAP system with labeled data have shown promising results of the above mentioned algorithms combination. There, we have found out that we can detect and classify different types of anomalous states.

As the data records and anomaly types are much huger than the ones we used for this work, further researches of the model should be undertaken before introducing the tool to a global enterprise. Owing to the fact that for each SAP system the determination of the GNG and the GMLVQ prototypes as well as mapping matrix Ω are better to be calculated just ones as uniform prototypes of the model, we need to implement the algorithm in some kind of productive environment, so that the results could be transferable to other systems, where we do not have labeled data. The open question is if we can use our trained model and hyper-parameters and just run them on another system, or another system still requires relearning. Moreover, it is essential to proceed with other studies focusing on the art of outlier selection. It is important to create a classifier with high classification ability and distinguish between other possible types of anomalies that might occur in a system.

In this work there were three anomalous groups taken into consideration, whereas in fact the number of occurring anomaly-kinds in the system is unknown. Therefore, there are two possible ways to continue this work: to set up the most frequently happened classes and adjust the model to the fixed number of them. The second and more sophisticated solution is to introduce an additional function based on the learned distance measure to detect outliers we do not know. So that, there will be an option not to assign a data vector to a cluster if it is drastically diverse from all the already existing prototypes, but rather signalize about that and automatically turn the newcomer-vector into a new prototype and create throughout a new group. The further runs executed by the model will include the new number of clusters. In order to prevent the model from extra prototypes or a cluster duplication, a kind of radius-function might be used as a condition to merge the clusters if the prototypes happen to find themselves close to each other.

During this study, we run the experiments with all 32 features of the given data. However, the GMLVQ provides us with additional information. The scaling matrix

$$\Lambda = \Omega^T \Omega$$

is the classification correlation matrix. This matrix stores the insights, which data dimensions and permutations endow an upcoming classification performance. For the considered anomaly detecting system such the histogram shown in Figure 6.5. The evaluation of it can be used for dimensionality reduction and, thereby, lower computation time in overall performance of the created tool. Further research could also include the optimization of other statistical measure, but classification errors, as well as, the receiver operating characteristic can be optimized in case of the binary classification.

The superiority of LVQ approaches and the GMLVQ itself is in their ability to conduct the prototype-adaptation (learning) off-line. Thus, the time processing costs do not play a big role in the application mode. The error computations for the GNG topology learning and the GMLVQ classification are easily maintained and are proven to converge in this work. So the results tend to be promising in a single system. Hence, there is still a question how to introduce the built system into a global production.

Appendix A: List of Features

Metric name	Abbr	Description
CPU	CPU	used CPU load
DISK_SIZE	DS	The total disk size
DISK_USED	DU	The current volume of disk used
HANDLE_COUNT	HC	Number of open handles
MEMORY_SIZE	MS	Physical memory size
MEMORY_USED	MU	Used memory by service
MVCC_VERSION _COUNT	MVC	Number of active MVCC versions. The multi vrsion concurrency control ensures consistent read operations
NETWORK_IN	NI	Bytes read from network by all processes
NETWORK_OUT	NO	Bytes written to network by all processs
PENDING_SESSION _COUNT	PSC	Number of pending requests
PING_TIME	PT	Duration of service ping request in ms
RECORD_LOCK _COUNT	RLC	Number of acquired record locks
STATEMENT_COUNT	SC	Number of finished SQL statements
SWAP_IN	SI	Bytes read from swap by Service
SWAP_OUT	SO	Bytes written to swap by all processes
SYSTEM_CPU	SCPU	OS kernel/system CPU used by service
TOTAL_SQL_ EXECUTOR_COUNT	SQL	Total number of SQLExecutors
TOTAL_THREAD _COUNT	TTC	Total number of threads
TRANSACTION _COUNT	TC	Number of open SQL transactions
TRANSACTION_ID _RANGE	TID	Range between newest and oldest active transaction ID
WAITING_SQL_ EXECUTOR_COUNT	WSQL	Number of waiting SqlExecutors
AITING_THREAD _COUNT	WTC	Number of waiting threads
ACTIVE_SQL_ EXECUTOR_COUNT	ASQL	The number of active SQL executors. An SQL executor organizes the execution of operations by invoking the corresponding SAP HANA component
ACTIVE_THREAD _COUNT	ATC	The number of active threads from the thread pool which are executing a runnable task pool

Table A.1: a) Dataset metrics of the system

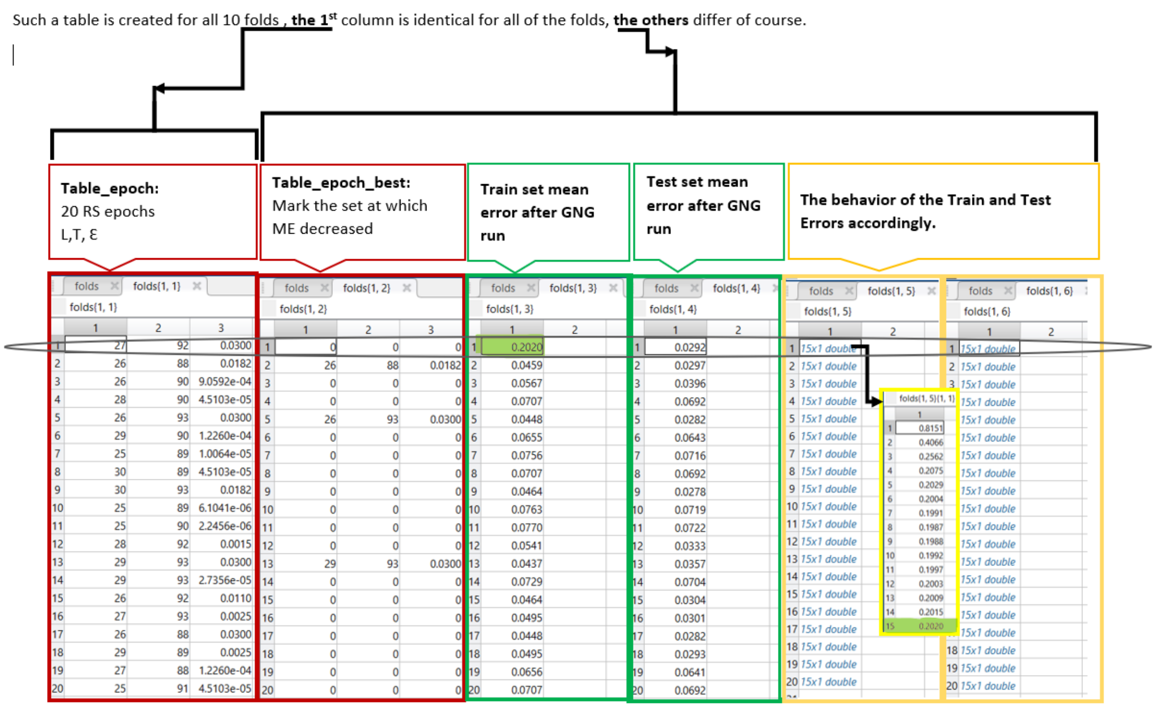
Metric name	**Abbr**	**Description**
BLOCKED_TRANSACTION_COUNT	BTC	Number of blocked SQL transactions
COMMIT_ID_RANGE	CID	Range between newest and oldest active commit ID. Commit IDs are simple incremented integers
CONNECTION_COUNT	CC	Number of open SQL connections to other database systems
MEMORY_ALLOCATION_LIMIT	MAL	MAL for service. Max-size of allowed MA
CS_MERGE_COUNT	CSM	Number of merge requests
CS_READ_COUNT	CSR	Number of read requests
CS_UNLOAD_COUNT	CSU	Number of columns unloads
CS_WRITE_COUNT	CSW	Number of written requests

Table A.2: b) Dataset metrics of the system

Appendix B: Reference Implementation in MATLAB for GNG

The combination of GNG and Random Search for hyper-parameters tuning. The code following is organized in a 'tree'-structure, consisting of several subfunctions called one out of another:

```
 SplitDataIntoInputsJ.m
  └─ObjFunJ.m
      └─GNG_RSJ.m
└─Evaluation.m
└─scatterplots.m
```



Such a table is created for all 10 folds , **the 1st** column is identical for all of the folds, **the others** differ of course.

Table_epoch:
20 RS epochs
L,T, Ɛ

Table_epoch_best:
Mark the set at which ME decreased

Train set mean error after GNG run

Test set mean error after GNG run

The behavior of the Train and Test Errors accordingly.

folds{1, 1}

	1	2	3
1	27	92	0.0300
2	26	88	0.0182
3	26	90	9.0592e-04
4	28	90	4.5103e-05
5	26	93	0.0300
6	29	90	1.2260e-04
7	25	89	1.0064e-05
8	30	89	4.5103e-05
9	30	93	0.0182
10	25	89	6.1041e-06
11	25	90	2.2456e-06
12	28	92	0.0015
13	29	93	0.0300
14	29	93	2.7356e-05
15	26	92	0.0110
16	27	93	0.0025
17	26	88	0.0300
18	29	89	0.0025
19	27	88	1.2260e-04
20	25	91	4.5103e-05

folds{1, 2}

	1	2	3
1	0	0	0
2	26	88	0.0182
3	0	0	0
4	0	0	0
5	26	93	0.0300
6	0	0	0
7	0	0	0
8	0	0	0
9	0	0	0
10	0	0	0
11	0	0	0
12	0	0	0
13	29	93	0.0300
14	0	0	0
15	0	0	0
16	0	0	0
17	0	0	0
18	0	0	0
19	0	0	0
20	0	0	0

folds{1, 3}

	1	2
1	0.2020	
2	0.0459	
3	0.0567	
4	0.0707	
5	0.0448	
6	0.0655	
7	0.0756	
8	0.0707	
9	0.0464	
10	0.0763	
11	0.0770	
12	0.0541	
13	0.0437	
14	0.0729	
15	0.0464	
16	0.0495	
17	0.0448	
18	0.0495	
19	0.0656	
20	0.0707	

folds{1, 4}

	1	2
1	0.0292	
2	0.0297	
3	0.0396	
4	0.0692	
5	0.0282	
6	0.0643	
7	0.0716	
8	0.0692	
9	0.0278	
10	0.0719	
11	0.0722	
12	0.0333	
13	0.0357	
14	0.0704	
15	0.0304	
16	0.0301	
17	0.0282	
18	0.0293	
19	0.0641	
20	0.0692	

folds{1, 5}

	1	2
1	15x1 double	
2	15x1 double	
3	15x1 double	
4	15x1 double	
5	15x1 double	
6	15x1 double	
7	15x1 double	
8	15x1 double	
9	15x1 double	
10	15x1 double	
11	15x1 double	
12	15x1 double	
13	15x1 double	
14	15x1 double	
15	15x1 double	
16	15x1 double	
17	15x1 double	
18	15x1 double	
19	15x1 double	
20	15x1 double	

folds{1, 5}{1, 1}

	1
1	0.8151
2	0.4066
3	0.2562
4	0.2075
5	0.2029
6	0.2004
7	0.1991
8	0.1987
9	0.1988
10	0.1992
11	0.1997
12	0.2003
13	0.2009
14	0.2015
15	0.2020

folds{1, 6}

	1	2
1	15x1 double	
2	15x1 double	
3	15x1 double	
4	15x1 double	
5	15x1 double	
6	15x1 double	
7	15x1 double	
8	15x1 double	
9	15x1 double	
10	15x1 double	
11	15x1 double	
12	15x1 double	
13	15x1 double	
14	15x1 double	
15	15x1 double	
16	15x1 double	
17	15x1 double	
18	15x1 double	
19	15x1 double	
20	15x1 double	

Figure B.1: Graphical

```
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%                        %%%%%%%%%%%%%%%%%%%
%%
 %%%%%%%%%%%%%%%%%%%%GNG with matrix-shaped inputs%%%%%%%%%%%%%%%%%%%
%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%
        %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%
         % Load Data

        % I call GNG on the whole data from here
        data = load('ScaledHealthyData');
            X = data.ScaledHealthyData;
            Xmin = min(X);
            Xmax = max(X);

            % Parameters from APP

            params.N = 50;% maximum # of nodes
            params.MaxIt = 300;
            % # the creterion for creating new nods (<= MaxIt), use it
 with 'ny'
            params.L = 50;
            params.epsilon_b = 0.03;% moderate BMU1 adaptation
            params.alpha = 0.5;% moderate local error
            params.delta = 0.005; % moderate global error
            params.T = 50;% maximum possible  age of an edge

            net = GNGJ(X, params, true); % GNG on the whole data
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%
             function net = GNGJ(X, params, PlotFlag)

                % Load Data
               % in MyAppLast.....

                % Prepare inputs
            %Y is a cell array
            % 5x32 matricies are stored in each cell
            windows = size(X,1)/5;
            %%%%%% Time Window%%%%%%%%
            Y = cell(windows, 1);
            i = 1;
            l = 1;


            while l ~= windows + 1
               Y(l,:) = {X(i:i+4,:)}; % cells of time windows
```

```
                    i = i+5;
                    l = l+1;
                end

                %Y = cell(496, 1);
                %i = 1;
                %l = 1;

                %while l ~= 497
                  % Y(l,:) = {X(i:i+4,:)};
                   %i = i+5;
                   % l = l+1;
                %end


                nDim = size(X,2);        % # of columns (32 in the
     example)
                nDataInputs = size(Y,1); % # of cells (496 in the example)


                Y = Y(randperm(nDataInputs), :);
                % Distribute data (at every new iteration we will
                %take a data sample, the samples shouldn't be
                %taken one by one as they are given, but randomly,
                %that is why we shuffle them now).

                       % Parameters
                    N = params.N; % maximum # of nodes
                    MaxIt = params.MaxIt;
                    L = params.L;
                    % # the creterion for creating new nods (<= MaxIt),
     use it with 'ny'
                    epsilon_b = params.epsilon_b; % moderate BMU1
     adaptation
                    alpha = params.alpha; % moderate local error
                    delta = params.delta; % moderate global error
                    T = params.T; % maximum possible  age of an edge

                     % Initialization
                % Create 2 randomly located nodes, they look exactly like
     some input
                % samples
                % so, there will be 2 cells, each cell stores a 5x32
     matrix

                    Ni = 2;
                    W = cell(Ni,1);
                rng('default'); %%%Seeding, so the initial prototypes will
     be alawys the same
                rng(2);         %%%and we can observe the error changing
     during the evaluation
                     for i = 1:Ni
                        W{i,:} = rand(5, nDim);
                     end
```

```
            % error for each node, if we have e.g. 5 nodes, E will be
one column with 5 rows,
            % 1st row stores an error for the 1st node, 2nd - for
node 2 and so on.
               E = zeros(Ni,1);
            % edges between nodes, in the very beginning it is a zero
2x2 matrix,
            % as we have 2 nodes and they are not connected , so '0'
here means -
            % that the edge doesn't exist
               C = zeros(Ni, Ni);
           %matrix which stores age between nodes, it is of the same
form as matrix 'C'
           % age of edges, so also 0 at first, but actually the edges
don't exist so far,
           % so, it doesn't realy matter, the matrice can be empty as
well
               t = zeros(Ni, Ni);

                % Loop

               ny = 0; %number of input samples

               % one of the stopping criterion, we run the Algorithm
until the maxnumb
```

of itterations is reached

```
               for it = 1:MaxIt
                   fprintf('Iteration step: %i / %i\n',it,MaxIt);
                       % we run through all inputs (496 cells), take
one at a time

                           for l = 1:nDataInputs

           % Select Input
            %within 1 iteration 'it' we can create several new nodes:
            ny = ny + 1;
            % this just to count the # of input we are at and if it
is mod of
            %L, then we insert a new node on this iteration
            y = Y{l,:}; % a 5x32 matrix of an input sample



                       %THIS PART WAS CREATED TO PICTURE THE MAIN
IDEA! SKIP all in
                       %stars


 %***********************************************
                       %*****d1 = norm(pdist2(y, W{1,1},
'cosine'));**
```

```
                       %*****d2 = norm(pdist2(y, W{2,1},
'cosine'));**
                       %*****d = [d1 d2];
**
                       %*****[~, SortOrder] = sort(d);
**
                       %*****s1 = SortOrder(1);
**
                       %*****s2 = SortOrder(2);
**

 %***********************************************



             % Choose metric:

             %'euclidean'
             %'minkowski'
             %'chebychev'
             %'cosine'
             %Here we calculate distances between an input and all
nodes
             %%in d(1) will be stored distance between the
             %current input sample and the 1st node, d(2)..., etc

               d = zeros(size(W,1));
                   for h = 1 : size(W,1) % from 1st to last node
                   d(h) = norm(pdist2(y, W{h,1}, 'euclidean'));
                   %W{h,1} - 5x32 matrix, d(h) 1xN array, where N is
number of weights
                   end

            [~, SortOrder] = sort(d); % in d we have distance values
in each cell,

              s1 = SortOrder(1); % # of the closest node
              s2 = SortOrder(2); % # of the second - closest node

              % Explanation. Skip all in stars if everything is clear
so far


 %*******************************************************************

 %*******************************************************************
                  %*SortOrder stores indexes of the cells in d in
ascending order.****
                  %*For
example:********************************************************
                       %*d = [4 , -2, 0], so
                       %*distance(w1,y) = 4;
                       %*distance(w2,y) =-2;
                       %*distance(w3,y) = 0
```

```
                 %*SortOrder = [2 , 3, 1] so the second node has the
smallest dist **
                  %*between y.
              **

 %*******************************************************************

 %*******************************************************************
                      %%%%%%%%%%%%%%%%%%%%%%%%%%%%

                      % Aging
                      t(s1, :) = t(s1, :) + 1;
                      t(:, s1) = t(:, s1) + 1;

                      % Add Error
                      E(s1) = E(s1) + d(s1)^2;

                      % Adaptation

                 W{s1,:} = W{s1,:} + epsilon_b*(y-W{s1,:}); %  W{s1,:}
- BMU, 5x32 matrix
                      Ns1 = find(C(s1,:)==1); % Ns1 - neighbors of
BMU;

                    %  C_Graph = graph(C);


                         for j = 2:size(W,1)
                             if j == Ns1
                            % TopNb = neighbors(C_Graph, j);
                            % Sig = 3/size(TopNb,1);
                             Sig = 3/size(W,1);
                 W{j,:} = W{j,:} + epsilon_b*exp(-SortOrder(j)/
(2*Sig^2))*(y-W{j,:});
                             else
                             Sig = 3/size(W,1);
                 W{j,:} = W{j,:} + epsilon_b*exp(-SortOrder(j)/
(2*Sig^2))*(y-W{j,:});
                             end
                         end

             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%
             %C(s1,:)==1 boolean, checks if  there are 1's in the s1-
th row,
             % find(C(s1,:)==1) returns number of columns (node) in
which we have 1
             %        for j=Ns1
             % W{j,:} = W{j,:} + epsilon_n*(y-W{j,:});
             % E.g. Ns1=[2,3,7], then node2, node3, node7 are adaped
             %        end
             %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%
```

```
                       % Create Link
                       C(s1,s2) = 1;
                       C(s2,s1) = 1;
                       t(s1,s2) = 0;
                       t(s2,s1) = 0;

           % Remove Old Links
              C(t>T) = 0;
              nNeighbor = sum(C);
              %returns a row vector and show the # of neighbors for
each node
              AloneNodes = (nNeighbor==0);% index of a node ithout
neighbors
              C(AloneNodes, :) = []; % delete the row # AloneNodes
from matrix C
              C(:, AloneNodes) = []; % delete the column # AloneNodes
from matrix C
              t(AloneNodes, :) = [];
              t(:, AloneNodes) = [];
              W(AloneNodes, :) = [];%delete the cell with node #
AloneNodes
              E(AloneNodes) = [];

                       % Add New Nodes
                       if mod(ny, L) == 0 && size(W,1) < N
              %ny counts the # of iteration and
              %if it is mod of L, then we insert a new node on this
iteration
                           [~, q] = max(E);
                           [~, f] = max(C(:,q).*E);
                           r = size(W,1) + 1;
                           W{r,:} = (W{q,:} + W{f,:})/2;
                           C(q,f) = 0;
                           C(f,q) = 0;
                           C(q,r) = 1;
                           C(r,q) = 1;
                           C(r,f) = 1;
                           C(f,r) = 1;
                           t(r,:) = 0;
                           t(:,r) = 0;
                           E(q) = alpha*E(q);
                           E(f) = alpha*E(f);
                           E(r) = E(q);
                       end

                       % Decrease Errors

                       E = delta*E;%accumulated error
                       B(it,:) = norm(E);
                           end

                   fprintf('Total error: %f\n', norm(E));
```

```
                    % For Plot
                    if PlotFlag
                        figure(1);
                        PlotResultsMyLast(Y, W, C)
                        pause(0.01);
                    end

                end

                % Export Results
                net.W = W;
                net.E = E;
                net.C = C;
                net.t = t;
                net.B = B;

            end
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%
            %%%%%%%%%%%%%%%%%%Tuning HPs with RS%%%%%%%%%%%%%%%%%%%%%
%%%%%%
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%

            data = load('ScaledHealthyData');
            X = data.ScaledHealthyData;

            % Create names folds: name->TrainSet->TestSet

            CfoldV = cell(10,3); % pre-allocation of folds

            for i = 1:10

            CfoldV{i,1} = sprintf('fold%i', i); % name of folds (just
 for convinient
            %reading)

            end

            % Fill in the folds splitting into 200 and 1800 set pairs
 (Test and Train)
            % Beginning%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%
            k = 200;
            CfoldV{1,2} = X(k+1:2000, :);% Train set 1
            CfoldV{1,3} = X(1: k,:); % Test set 1

            for i = 2:10
```

```
                CfoldV{i,2} = cat( 1, X(1:(i-1)*k, :), X(i*k
+1:2000,:));%TrainSets 2-10
                CfoldV{i,3} = X((i-1)*k+1: i*k,:); %T est Sets 2-10
            end
            %End%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%
            folds = cell(10,6); % preallocation, here I store sets of
            %HPs for each epoch and Mean Errors for Train and Test
 sets
            epoch_number = 20; % random search epochs
            OptimResult = 0.05; % randomly chosen number, I want Err
 be less or so
            table_epoch = zeros(epoch_number,3);% where we record L,T,
 epsilon for
            %each epoch
            table_epoch_best = zeros(epoch_number,3); %the epochs
 where Err went down

            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%

            for fold = 1:10
                Xtrain = CfoldV{fold, 2};
                Xtest = CfoldV{fold, 3};


                FindMeanError = zeros(epoch_number, 1);
                FindMeanTestError = zeros(epoch_number, 1);
                MeanTrainIt = cell(epoch_number, 1);
                MeanTestIt = cell(epoch_number, 1);



                       rng('default'); %%%Seed
                       rng(1);
                       for epn = 1:epoch_number
                       L_candidate = randi([1, 100],1);
                       T_candidate = randi([1, 100],1);
                       rnd_n = randi(10,1,1);
            %epsilon_b_candidate = 0.5*exp(-0.5*(rnd_n-1));
            % Value of the learning rate
             epsilon_b_candidate = 0.03;
            % Record the 3-tuple [L, T, epsilon]:
            table_epoch(epn,:)=[L_candidate T_candidate
 epsilon_b_candidate];
            % L, T, epsilon_b
                       end


             for epn = 1:epoch_number
```

```
                    % Parameters from APP

                params.N = 50;% maximum # of nodes
                params.MaxIt = 15;%#the creterion forcreatingnewnods(<=
MaxIt,useitwith'ny'
                params.L = table_epoch(epn,1);
                params.epsilon_b = table_epoch(epn,3);% moderate BMU1
adaptation
                params.alpha = 0.5;% moderate local error
                params.delta = 0.005; % moderate global error
                params.T = table_epoch(epn,2);% maximum possible  age of
an edge

                [FindMeanError(epn,1),FindMeanTestError(epn,1),
MeanTrainIt{epn,1}, ...
                    ...MeanTestIt{epn,1}] = ...
                ...ObjFunJ(Xtrain, Xtest, params.L, params.T,
params.epsilon_b);

                     if FindMeanError(epn,1) <= OptimResult
                         Lbest = params.L;
                         Tbest = params.T;
                         epsilon_best = params.epsilon_b;
                         OptimResult = FindMeanError(epn,1);
                         table_epoch_best(epn,:)=[Lbest Tbest
epsilon_best];
                      end
                 end

                folds{fold, 1} = table_epoch;
                folds{fold, 2} = table_epoch_best;

                folds{fold, 3} = FindMeanError;
                folds{fold, 4} = FindMeanTestError;

                folds{fold, 5} = MeanTrainIt;
                folds{fold, 6} = MeanTestIt;
                end
                 %%%%%%%%%%%%%%%%%%%%%%%%%%%
                 % *______ObjFunJ______* %%
                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
                 %the function calls GNG on 10 different sets   %
                 %and outputs Mean Error for evluation           %
                 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

                function  [FindMeanError,FindMeanTestError, MeanTrainIt,
MeanTestIt] = ...
                ObjFunJ(Xtrain, Xtest,L, T, epsilon_b)


                % Parameters from APP %%

                params.N = 50;% maximum # of nodes
params.MaxIt = 15;
% # the creterion for creating new nods
%(<= MaxIt), use it with 'ny'
params.L = L;
params.epsilon_b = epsilon_b;% moderate BMU1 adaptation
params.alpha = 0.5;% moderate local error
params.delta = 0.005; % moderate global error
params.T = T;% maximum possible  age of an edge

net = GNG_RSJ(Xtrain, Xtest, params, false);
% false means there is no plot executed


FindMeanError = mean(net.E_train);%mean(net.E);
FindMeanTestError = mean(net.E_test);
MeanTrainIt = net.MeanTrainError;%net.MeanError; % array
MeanTestIt = net.MeanTestError; %array

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%
 %%%%%%%%%%%%%%%%%%%%%%%%%%%
 % *______GNG_RSJ______* %%
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 %the function performs GNG on 10 different sets%
 %and outputs Mean Error for evluation          %
 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function net = GNG_RSJ(Xtrain, Xtest, params, PlotFlag)

    % Prepare inputs
% # of Time windows in Train set, time window is an input
windowsTrain = size(Xtrain,1)/5;
windowsTest = size(Xtest,1)/5;


%%%%%%Train Time Window%%%%%%%%%
Ytrain = cell(windowsTrain, 1);
i = 1;
l = 1;

while l ~= windowsTrain + 1
   Ytrain(l,:) = {Xtrain(i:i+4,:)}; % cells of time
windows
    i = i+5;
    l = l+1;
end

%%%%%%Test Time Windows%%%%%%%%%
Ytest = cell(windowsTest, 1);
i = 1;
l = 1;

while l ~= windowsTest + 1
   Ytest(l,:) = {Xtest(i:i+4,:)};
```

```
                i = i+5;
                l = l+1;
            end
            %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
Error using evalin
Undefined function 'SplitDataIntoInputsJ' for input arguments of type
 'double'.
```

Published with MATLAB® R2019b

Bibliography

[1] Understanding confusion matrix. URL `https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62`.

[2] Hyperparameter tuning the random forest in python. URL `https://towardsdatascience.com/hyperparameter-tuning-the-random-forest-in-python-usi`

[3] Understanding auc-roc curve. URL `https://towardsdatascience.com/understanding-auc-roc-curve-68b2303cc9c5`.

[4] Kohonen self-organizing maps. URL `https://towardsdatascience.com/kohonen-self-organizing-maps-a29040d688da`.

[5] A no-nonsense matlab (tm) toolbox for gmlvq (2015). URL `https://www.cs.rug.nl/biehl/gmlvq.html`.

[6] J. Bergstra and Y. Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, 2012. URL `http://dblp.uni-trier.de/db/journals/jmlr/jmlr13.html#BergstraB12`.

[7] D. P. Bertsekas and J. N. Tsitsiklis. Comments on “coordination of groups of mobile autonomous agents using nearest neighbor rules”. *IEEE Transactions on Automatic Control*, 52(5):968–969, May 2007. ISSN 2334-3303. doi: 10.1109/TAC.2007.895885.

[8] M. Biehl. Relevance and matrix adaptation in learning vector quantization (grlvq, gmlvq and liram lvq), 2017. URL `http://matlabserver.cs.rug.nl/gmlvqweb/web/`.

[9] K. Bunte, P. Schneider, B. Hammer, F. Schleif, T. Villmann, and M. Biehl. Limited rank matrix learning, discriminative dimension reduction and visualization. *Neural Networks*, 26:159–173, Feb. 2012.

[10] F. Canales and M. Chacón. Modification of the growing neural gas algorithm for cluster analysis. In L. Rueda, D. Mery, and J. Kittler, editors, *Progress in Pattern Recognition, Image Analysis and Applications*, pages 684–693, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg. ISBN 978-3-540-76725-1.

[11] M. E. Celebi and K. Aydin. *Unsupervised Learning Algorithms*. Springer Publishing Company, Incorporated, 1st edition, 2016. ISBN 3319242091.

[12] O. Chapelle, B. Schlkopf, and A. Zien. *Semi-Supervised Learning*. The MIT Press, 1st edition, 2010. ISBN 0262514125.

[13] Y. Fedorenko. Data classification based on neural gas and markov's algorithms. *Molodezniy Nauchno-Technicheskiy Vestnik*, 08 2014. doi: 004.67.

[14] F. Färber, N. May, W. Lehner, P. Große, I. Müller, H. Rauhe, and J. Dees. The sap hana database – an architecture overview. *IEEE Data Eng. Bull.*, 35(1):28–33, 2012. URL `http://dblp.uni-trier.de/db/journals/debu/debu35.html#FarberMLGMRD12`.

[15] B. Fritzke. A growing neural gas network learns topologies. In *Proceedings of the 7th International Conference on Neural Information Processing Systems*, NIPS'94, pages 625–632, Cambridge, MA, USA, 1994. MIT Press. URL `http://dl.acm.org/citation.cfm?id=2998687.2998765`.

[16] J. Hale. Scale, standardize, or normalize with scikit-learn, 2018. URL `https://www.kaggle.com/discdiver/guide-to-scaling-and-standardizing`.

[17] J. Hale. Scale, standardize, or normalize with scikit-learn, 2019. URL `https://towardsdatascience.com/scale-standardize-or-normalize-with-scikit-learn-6ccc7d176a02`.

[18] A. Harri and B. W. Brorsen. The overlapping data problem. 1998.

[19] S. Hert and M. Seel. dD convex hulls and delaunay triangulations. In *CGAL User and Reference Manual*. CGAL Editorial Board, 2019. URL `https://doc.cgal.org/5.0/Manual/packages.html#PkgConvexHullD`.

[20] C. Hofer, R. Kwitt, M. Niethammer, and A. Uhl. Deep learning with topological signatures. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 1633–1643, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

[21] I. Jolliffe. *Principal Component Analysis*, pages 1094–1096. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-04898-2. doi: 10.1007/978-3-642-04898-2_455. URL `https://doi.org/10.1007/978-3-642-04898-2_455`.

[22] T. Kohonen. *Self-Organizing Maps*. Springer-Verlag, Berlin, Heidelberg, 1997. ISBN 3540620176.

[23] T. Kohonen, M. R. Schroeder, and T. S. Huang. *Self-Organizing Maps*. Springer-Verlag, Berlin, Heidelberg, 3rd edition, 2001. ISBN 3540679219.

[24] D. Kumar, J. C. Bezdek, S. Rajasegarar, M. Palaniswami, C. Leckie, J. Chan, and J. Gubbi. Adaptive cluster tendency visualization and anomaly detection for streaming data. *ACM Trans. Knowl. Discov. Data*, 11(2), Dec. 2016. ISSN 1556-4681. doi: 10.1145/2997656. URL https://doi.org/10.1145/2997656.

[25] T. Martinetz. Competitive hebbian learning rule forms perfectly topology preserving maps. 01 1993. doi: 10.1007/978-1-4471-2063-6_104.

[26] T. Martinetz and K. Schulten. A neural network for robot control: Cooperation between neural units as a requirement for learning. *Comput. Electr. Eng.*, 19(4): 315–332, July 1993. ISSN 0045-7906. doi: 10.1016/0045-7906(93)90053-T. URL http://dx.doi.org/10.1016/0045-7906(93)90053-T.

[27] S. Mostapha Kalami Heris. Neural gas network in matlab, 2015. URL http://yarpiz.com/77/ypml111-neural-gas-network.

[28] D. Mudali, M. Biehl, K. Leenders, and J. Roerdink. *LVQ and SVM Classification of FDG-PET Brain Data*, volume 428, pages pp 205–215. 01 2016. doi: 10.1007/978-3-319-28518-4_18.

[29] D. Nebel, M. Kaden, A. Villmann, and T. Villmann. Types of (dis-) similarities and adaptive mixtures thereof for improved classification learning. *Neurocomputing*, 268:42–54, Dec. 2017.

[30] P. Refaeilzadeh, L. Tang, and H. Liu. *Cross-Validation*, pages 532–538. Springer US, Boston, MA, 2009. ISBN 978-0-387-39940-9. doi: 10.1007/978-0-387-39940-9_565. URL https://doi.org/10.1007/978-0-387-39940-9_565.

[31] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall Press, USA, 3rd edition, 2009. ISBN 0136042597.

[32] A. Sato and K. Yamada. Generalized learning vector quantization. In *Proceedings of the 8th International Conference on Neural Information Processing Systems*, NIPS'95, page 423–429, Cambridge, MA, USA, 1995. MIT Press.

[33] P. Schneider, M. Biehl, and B. Hammer. Relevance matrices in lvq. In *ESANN*, 2007.

[34] P. Schneider, M. Biehl, and B. Hammer. Adaptive relevance matrices in learning vector quantization. *Neural Comput.*, 21(12):3532–3561, Dec. 2009. ISSN 0899-7667. doi: 10.1162/neco.2009.11-08-908. URL https://doi.org/10.1162/neco.2009.11-08-908.

[35] P. Schneider, M. Biehl, and B. Hammer. Adaptive relevance matrices in learning vector quantization. *Neural Computation*, 21(12):3532–3561, 2009. doi: 10.1162/neco.2009.11-08-908. URL `https://doi.org/10.1162/neco.2009.11-08-908`. PMID: 19764875.

[36] T. Villmann and J. C. Claussen. Magnification control in self-organizing maps and neural gas. *Neural Comput.*, 18(2):446–469, Feb. 2006. ISSN 0899-7667. doi: 10.1162/089976606775093918. URL `http://dx.doi.org/10.1162/089976606775093918`.

[37] R. Xu and D. Wunsch. *Clustering*. Wiley-IEEE Press, 2009. ISBN 9780470276808.

[38] Y.-L. Zhang, L. Li, J. Zhou, X. Li, and Z.-H. Zhou. Anomaly detection with partially observed anomalies. In *Companion Proceedings of the The Web Conference 2018*, WWW '18, page 639–646, Republic and Canton of Geneva, CHE, 2018. International World Wide Web Conferences Steering Committee. ISBN 9781450356404. doi: 10.1145/3184558.3186580. URL `https://doi.org/10.1145/3184558.3186580`.

Eidesstattliche Versicherung

Hiermit versichere ich an Eides statt, dass ich meine Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die Arbeit noch nicht anderweitig für Prüfungszwecke vorgelegt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

Mittweida, 30. March 2020