Using Self-Organizing Maps to Identify Operational Risk

Jonatan Berggren, Lukas Ljungblom

June 12, 2018

Abstract

In recent years, the awareness and concern for operational risk in financial institutions have increased, and several disastrous events in the last two decades been caused by human error. With this, the regulatory demands have increased on the financial institutions to control operational risk.

One operational risk that Svenska Handelsbanken AB (SHB) has detected are the audit changes of trades which occur when a trade need some form of altering from its original state, which can lead to losses for the bank. The bank has looked into identifying and forecasting these losses with the use of a neural network clustering method called Self-Organizing Map.

This thesis expands on a previous project initialized by SHB on the potential of using this method to identify operational risk, and research the robustness and effectiveness of the Self-Organizing Map and trying to obtain an optimal solution by using quantifiable measurements like Matthew's Correlation Coefficient.

By evaluating the algorithm through visualizations of the generated maps and evaluating its prediction ability through Cross-Validation, the results obtained from this thesis indicate that the Self-Organizing Map has great potential in this area and is able to identify these risks with a relatively high accuracy.

Key Words: Self-organizing Map, Machine Learning, Neural Network, Clustering, Operational Risk, Audit Change

Acknowledgements

This thesis was written during the spring of 2018 at the Department of Mathematical Statistics at Lund University, in cooperation with the Risk Control Department at Svenska Handelsbanken AB, Stockholm.

We wish to thank various people for their contribution to the project; Erik Lindström, our supervisor at LTH, for his guidance and support; Richard Henricsson and Cecilia Pettersson, our supervisors at SHB, for their valuable expertise and support; The rest of the staff at the Risk Control Department at SHB, for letting us be part of the team; Markus Holmgren and Hampus Pettersson, our fellow master's students at SHB, for great companionship and sharing of ideas.

This project has been of great enjoyment and has broaden our knowledge in machine learning, programming, statistical analysis and risk management.

Contents

0.1 Abbreviations

SHB	Svenska Handelsbanken AB
SOM	Self-organizing map
U-matrix	Unified distance matrix
BMU	Best matching unit
ROC	Receiver operating characteristic
AUC	Area under the ROC-curve
P/N	Positive/Negative
PP/PN	Predicted Positive/Negative
TP/TN	True Positive/Negative
FP/FN	False Positive/Negative
MCC	Matthews Correlation Coefficient
SMOTE	Synthetic Minority Over-sampling Technique

Chapter 1

Introduction

1.1 Background

The risk management in financial institutions has in recent years, since the global financial crisis, become more substantial and important for every actor in the financial market. The requirements on these institutions to handle their risks has also increased with the expansion of the Basel Committee Regulation Framework. The risk management of a company includes managing their credit risk, market risk, liquidity risk, and increasingly important, their operational risk. The Basel Committee defines operational risk as *"the risk of loss resulting from inadequate or failed internal process, people and systems or from external events"* [9]. Over the years, many of the most severe and disastrous losses related to financial institutions can be traced back to single events caused by individuals. Some examples includes the Allied Irish Bank in 2002, Soceite Generale in 2008 and perhaps most famously the collapse of Barings Bank caused by the speculative trading of Nick Leeson in 1995 [6]. In 1997, the British Bankers Association and Coopers Lybrand performed a research that showed the importance of surveiling a company's operational risk. They found that in 70% of the monitored banks, operational risk was more or equally important as market or credit risk [10].

One component that is included in operational risk is the audit changes of trades, which occurs when the information of a transaction is changed from its original state by updating or correcting of the initial input, or by cancellation of a transaction, i.e. an effect of human error.

As a result of the technological progress and the ability to store large amount of data, the possibilities to model and draw conclusions from big data have developed and opened new ways to analyze trades and to model and manage operational risk [19]. This thesis revolves around the categorization, measuring and forecasting of operational risk in the form of audit changes by using big data-simulation.

1.2 Aim of the thesis

The aim of this thesis is to investigate the possibility of using a so-called Self-Organizing Maps (SOM), which is an unsupervised machine learning method that uses an artificial neural network, to capture operational risk. The idea is to find clusters within data of multiple dimensions in a low-dimensional output space, most commonly a two-dimensional grid, in order to categorize, measure and forecast losses resulting from audit changes. The data in this thesis is based on historical audit changed trades provided by SHB. The thesis expands on previous basic work done on the subject conducted by SHB [20], that showed potential for using SOMs to identify operational risk. The aim is to explore the method further and execute various performance tests to get quantitative results.

This thesis is an initiative of the Risk Control Department at SHB. The bank has a low tolerance when it comes to risks and SHB is a world leader in credit rating and no bank has a higher credit rating from Moody's, Standard and Poor's or Fitch [3] as of 2017. The risks identified by the bank are credit risk, market risk, financial and liquidity risk, operational risk, compliance risk, renumeration system risk and insurance department risk [4]. Furthermore, even tough SHB operates in a such cautious manner, the bank has been able to obtain their company goal of having a higher profitability than the average of their competitors in their home markets for 46 years in a row.

1.3 Operational risk

As mentioned above, operational risk has become increasingly more important and the handling and control over this area have developed, and so has the models to capture the risk.

The earliest works concerning operational risk were more qualitative than quantitative, and most of the prior quantitative research has revolved around capturing the outcome of the risks [22]. One early method to quantify operational risk was to apply Extreme Value Theory to approximate the losses, and in 2003 the Federal Reserve Bank of Boston introduced the idea of using a Generalized Pareto Distribution on the matter, which showed some promise [18]. The convergence of this model was however proved to be very slow by Degen in 2007, which could lead to insufficient results [16]. An approach introduced by Martinez-Sanchez was to use a Markov Chain Monte Carlo simulation method with a Bayesian Network approach to track the risks where nodes represented different causes for operational losses, which appeared to be a feasible method [12]. In recent years the possibilities offered by Machine Learning have been increasingly explored.

1.4 Outline

In the second chapter, a description concerning the fundamental theory for this thesis is provided. A short general description of machine learning continuing with a thorough description of the theory on which the Self-Organizing Map algorithm is based on, including its purpose, background and different steps are covered. Further on, the chapter handles the theory for different methods to test the robustness and effectiveness of the algorithm's ability to identify clusters with Cross-Validation, ROC and K-means, and also a method to handle imbalanced data set with SMOTE.

In the third chapter, the data set provided by SHB is given an introduction and a way of processing categorical data in order for the algorithm to handle the data called a One-hot encoder is described.

In the fourth chapter, the methods for running, visualizing and testing the algorithm based on the theory is described, and in the fifth chapter, the results from these methods are presented.

In the sixth chapter, a discussion on the methods performance, parameters and efficiency based the result is presented. In the seventh and final chapter, a short summary on the thesis outcome is provided along with some realizations and recommendations for future work within this field.

Chapter 2

Theory

2.1 Unsupervised Machine Learning

A machine learning algorithm can be defined as an algorithm that has the ability to learn without being explicitly programmed, i.e. "hard coded". The algorithm uses input data to generate a specific desired outcome by learning from experience and adapting through training and repetition. Throughout the training, the model configures itself so that it comes closer to the desired outcome. The sought-after result is that when presented with data of the same categorization, the outcome will be the desired one independent of whether the input data is the training data or new data. This way, it becomes possible to categorize data without the need of determining the specific criteria that the data needs to have in order to end up in the desired category, which can be a very difficult task especially when dealing with very large data sets. Instead the algorithm generates the categorization by itself. Unsupervised machine learning is used to draw conclusions and find patterns in the underlying structure of the data when you do not have any corresponding output data. The unsupervised learning approach is used to look for similarities in the input data and organize it, and is a commonly used method to find clusters in data [23].

2.2 SOM

In the 1980s, the Finnish professor Teuvo Kohonen from Helsinki University of Technology, nowadays Aalto University, proposed a new algorithm of unsupervised machine learning using a neural network. The idea from Kohonen was to try to mimic and explain how the neurons in the brain ordered sensory signals in the cerebral cortex. One application example of this is of the neurons in the olfactory cortex, which are activated by smells, and those neurons that becomes activated by similar smells lies nearby each other in the cortex. The idea therefore became a grid, or a map, of neurons which after training would hold input data points that are similar to those which lie nearby, and input data where the underlying attributes differ substantially would lie further away or at least with a clear barrier between them, hence creating clusters in the map. The neurons can for example be organized in a hexagonal or a rectangular grid. Each neuron in the map holds, additionally to the coordinates of their position in the map, a vector of weights which is updated throughout the training where the number of weights is equal to the number of attributes in the input data space[15].

The algorithm that Kohonen proposed to accomplish this consists of three essential steps:

1. The initialization of the map
2. The decision step
3. The update step

The initialization step consists of initializing all of the components that make up the map before the training begins. The number of neurons in the map and the dimensions of the map, the number of epochs or iterations that the algorithm will do before completing the training of the map as well as the initial neighborhood size and the initial learning rate of the neurons are the design parameters set during the initialization of the map. The weight of each neuron is initialized at random. The training of the map consists of an iteration through the two latter steps, i.e. the decision and update step, and will be described in more detail in the following sections.

2.2.1 The Decision Step

A SOM practice something called competitive learning. Let the vector $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ define a randomly selected data point from the input data, where n is the number of attributes of the data, and let the weight vector $W_i = [W_{i1}, W_{i2}, \ldots, W_{in}]$ define the weights of the i^{th} neuron of the map. The distance between the selected data point $\mathbf{x}$ and W_i is defined by $f(\mathbf{x}, W_i)$ [15]. The competitiveness of the algorithm occur when selecting the best matching vector with index b, which is the vector with the minimum distance to $\mathbf{x}$,

$$f(\mathbf{x}, W_b) = \min_i f(\mathbf{x}, W_i). \tag{2.1}$$

The index b is obtained by

$$b = \arg(\min_i f(\mathbf{x}, W_i)). \tag{2.2}$$

The data point W_b that is chosen during the decision step is called the Best Matching Unit (BMU).

2.2.2 The Update Step

When the BMU W_b has been selected, the weights belonging to that neuron and neurons within its neighborhood are updated according to a gradient update shown in Equation 2.3 [15]. The neighborhood is closer defined later on.

$$W_i(t+1) = W_i(t) + \eta_i(t)\Big(\mathbf{x}(t) - W_i(t)\Big) \tag{2.3}$$

Here, η_i(t) is the adaption rate for the i^{th} neuron within the neighborhood. The idea of the algorithm is that the adaption rate and the neighborhood around the BMU will decrease and shrink respectively over the training period.

2.2.3 Distance Measures

Several times in the model the distance between two points is calculated, both in the input and the output space. One example is when finding the BMU in Equation 2.2. There are several different distance measurements, which can be used for different purposes. In this thesis three different measures are used, the Manhattan distance (L_1), the Euclidean distance (L_2) and the Chebyshev distance (L_∞). They are all variants of the general Minkowski distance (L_p). For two points $(x_1, x_2, ..., x_n)$ and $(y_1, y_2, ..., y_n)$

those four measures are defined as

$$\text{Minkowski distance} = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p} \tag{2.4}$$

$$\text{Manhattan distance} = \sum_{i=1}^{n} |x_i - y_i| \tag{2.5}$$

$$\text{Euclidean distance} = \left(\sum_{i=1}^{n} |x_i - y_i|^2\right)^{1/2} \tag{2.6}$$

$$\begin{aligned}\text{Chebyshev distance} &= \lim_{p\to\infty}\left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{1/p}\\ &= \max\left(|x_1 - y_1|, |x_2 - y_2|, \ldots, |x_n - y_n|\right).\end{aligned} \tag{2.7}$$

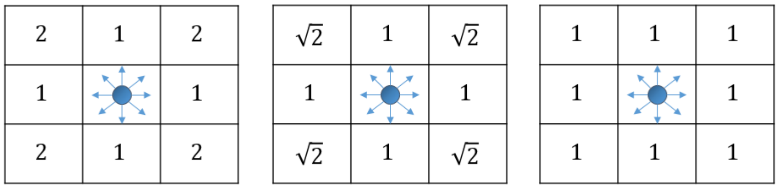
In one dimension these distances are the same, which is simply the absolute value of the difference between the points. In two dimensions, which the output space of the SOM has, the distance between the points $\mathbf{x} = (x_1, x_2)$ and $\mathbf{y} = (y_1, y_2)$ differ between the three measurements.

$$\begin{aligned}\text{Manhattan distance: } \delta(\mathbf{x}, \mathbf{y}) &= ||\mathbf{x} - \mathbf{y}||_1\\ &= |x_1 - y_1| + |x_2 - y_2| \end{aligned} \tag{2.8}$$

$$\begin{aligned}\text{Euclidean distance: } \delta(\mathbf{x}, \mathbf{y}) &= ||\mathbf{x} - \mathbf{y}||_2\\ &= \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2} \end{aligned} \tag{2.9}$$

$$\begin{aligned}\text{Chebyshev distance: } \delta(\mathbf{x}, \mathbf{y}) &= ||\mathbf{x} - \mathbf{y}||_\infty\\ &= \max(|x_1 - y_1|, |x_2 - y_2|) \end{aligned} \tag{2.10}$$

A good example to illustrate the difference between the Euclidean and Manhattan distances is how to get from one place to another in a city with perfectly square blocks. The Euclidean distance describes the shortest way, the way that a bird can fly. The Manhattan distance describes the shortest way a person can walk among the streets and make left and right turns in order to reach the destination. The Chebyshev distance is also called the "Chessboard Distance" since it can be illustrated by how many steps that a king on a chessboard has to go in order to reach a certain place [8]. Figure 2.1 illustrates the three measurements in two dimensions.



(a) Manhattan L_1 (b) Euclidean L_2 (c) Chebyshev L_∞

Figure 2.1: Illustration of the distance from a neuron in a two dimensional grid with the three different distance measurements.

In Figure 2.1, the Manhattan distance is equal or greater than the Euclidean distance, which in turn is equal or greater than the Chebyshev distance. That is always the case, except in one dimension when they are all equal. These methods can be used to calculate the distance of the BMU's neighborhood, as well as to identify clusters in the map. Both of these uses will be described in closer detail later on.

2.2.4 Neighborhood

The idea of a SOM is to be able to identify clusters within the map, which means that neurons that are close to each other in output space, should also be close in input space. That is achieved by not only updating the BMU for every iteration, but also the neurons that are in the neighborhood of the BMU. The design of the neighborhood is determined by how the neurons in the neighborhood are connected, the size of the neighborhood, and how much the update step should influence the neurons in the neighborhood [15].

The connection of the neighborhood is determined by the choice of distance measurement, usually Euclidean or Manhattan. The difference between these two becomes that the neurons that does not lie directly above, below or to one of the BMU's sides will have shorter distances with the Euclidean measurement than with the Manhattan measurement. For example, the closest diagonal neuron will have the distance of 2 with Manhattan and $\sqrt{2}$ with Euclidean.

Since the weights of the neurons are randomly generated initially, data points that are similar could end up far apart on the map at the start of the simulation. It is therefore favorable to start off with a large neighborhood size to make sure that some data points will not get stuck initially, and through the training period reduce the size of the neighborhood in order to have a better precision towards the end. One way of computing the size of the neighborhood in a beneficial way is:

$$\alpha(t) = \alpha_0 \exp\left(\frac{-\text{iteration}(t)}{\lambda}\right). \tag{2.11}$$

Here $\alpha(t)$ is the neighborhood size at iteration t, α_0 is the initial neighborhood size, iteration(t) is the number of iterations at time t and λ is a constant. In order to get a final neighborhood size of 1, λ should be set to:

$$\lambda = \frac{\text{iterations}}{\ln(\alpha_0)}. \tag{2.12}$$

In order to include the majority of the neurons in the initial neighborhood, α_0 is usually set to half of the longest side of the map, i.e

$$\alpha_0 = \frac{\max(length, width)}{2}. \tag{2.13}$$

The learning rate $L(t)$, which is a part of determining how much the update step will influence the weight of each neuron in the neighborhood, also needs to be defined. The idea, similar to the definition of $\alpha(t)$, is that the learning rate will decrease during the training period to make a general placement at first, and then fine-tune the placement later on. L_0 is a constant, usually set to 0.1.

$$L(t) = L_0 \exp\left(\frac{-\text{iteration}(t)}{\text{iterations}}\right) \tag{2.14}$$

To complete the adaption rate $\eta(t)$ from Equation 2.3, a parameter concerning the distance to the BMU is defined. The idea is that the BMU is supposed to be influenced with the highest rate, and then the rate decreases the further the distance between the BMU and each neuron in the neighborhood is. To implement this, a version of the Gaussian distribution is used and the decay factor for updating the weights in the i^{th} neuron in the neighborhood is defined as

$$\theta_i(t) = \exp\left(\frac{-\text{distance}_i^2}{2\alpha^2(t)}\right). \tag{2.15}$$

Here distance is the distance between the i^{th} neuron and the BMU defined in either Equation 2.5 or Equation 2.6, and $\alpha(t)$ is the neighborhood size defined in Equation 2.11. This results in an adaption rate for the i^{th} neuron of:

$$\eta_i(t) = L(t)\theta_i(t). \tag{2.16}$$

2.3 Model Evaluation

2.3.1 U-matrix

The unified distance matrix, or short U-matrix, first introduced in [26] is a widely used tool to inspect the performance of a SOM. The idea is to see if the final map does indeed cluster the data, by measuring the distance between the weights of neighboring neurons. For each pair of neighboring neurons p_1 and p_2 with n attributes the euclidean distance between them is measured as

$$\delta(p_1, p_2) = \sqrt{\sum_{i=1}^{n} (p_1^i - p_2^i)^2}. \tag{2.17}$$

The size of the dimensions of the U-matrix is twice those of the SOM minus one, so that in between all neurons the distances can be displayed. The points displaying the distances are what is interesting, but to get a better visualization all other points are set to the average of the neighboring points. Table 2.1 illustrates how the U-matrix for a $2x2$ SOM is calculated. Note that are there no distances for diagonally related neurons, instead point 5 in Table 2.1 is the average of the neighboring distances.

Table 2.1: Calculation of the U-matrix for a $2x2$ SOM

1. *Neuron* Average of 2 & 4	2. Distance between 1 & 3	3. *Neuron* Average of 2 & 6
4. Distance between 1 & 7	5. Average of 2, 4, 6, 8	6. Distance between 3 & 9
7. *Neuron* Average of 4 & 8	8. Distance between 7 & 9	9. *Neuron* Average of 6 & 8

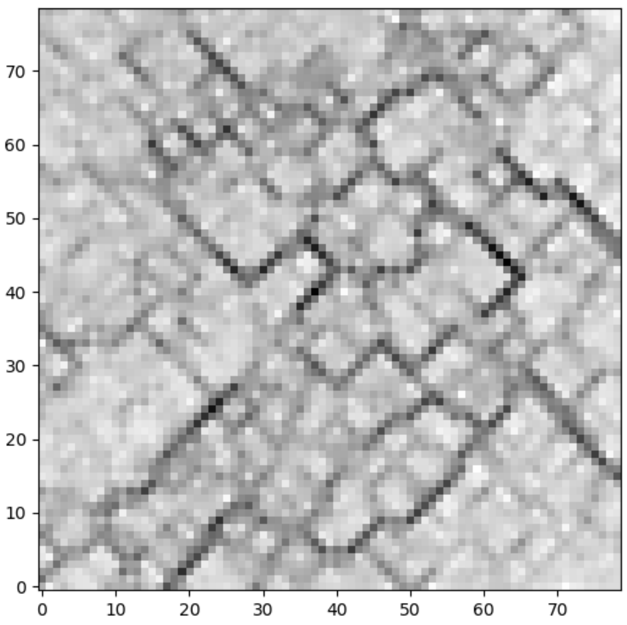


Figure 2.2: Illustration of a typical U-matrix for a $40x40$ SOM

A typical U-matrix can be seen in figure 2.2. Typically the U-matrix is visualized in gray-scale where light colors depict small distances and dark colors large distances between neurons. If the SOM successfully clusters the data, the U-matrix will show clusters of light colors separated by strings of dark colors.

2.3.2 Cross-Validation

Cross-Validation is a traditional method used to estimate the error of a prediction. The method consists of having a set of data which can be divided into k numbers of smaller sets, $\mathbf{s} = [s_1, s_2, \ldots, s_k]$, with roughly the same size, and having the learning algorithm train with the entire set except for one subset at a time, and doing this training k times. After each training, the excluded subset is tested against the trained data in order to examine the performance of the learning algorithm [2].

2.3.3 Binary Classifier

One common basis for evaluating machine learning algorithm which has a binary (two classes) classification application is the confusion matrix illustrated in Table 2.2.

Table 2.2: Confusion Matrix

	Predicting Positive	Predicting Negative
Actual Positive	TP	FN
Actual Negative	FP	TN

The columns represent the classification of the data that is obtained from running the machine learning algorithm, and the rows represent the actual classification of the data. TP stands for *True Positive* which indicates the number of actual positive samples labeled as positive through the algorithm. FP stands for *False Positive* and indicates the number of actual negative samples labeled as positive. FN stands for *False Negative* and indicates the number of actual positive samples labeled as negative. TN stands for *True Negative* and indicates the number of actual negative samples labeled as negative [13].

2.3.4 ROC and Youden's J Statistic

The receiver operating characteristic curve, or the ROC-curve, is a method of visualizing the performance of the prediction ability of a binary classifier. The curve visualizes the rate of true positive samples versus the rate of false positive samples obtained by using different thresholds in the algorithm, see Figure 2.3.

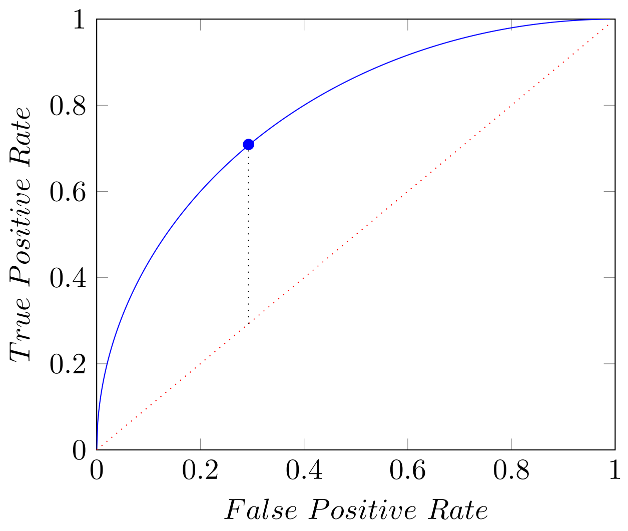


Figure 2.3: Hypothetical ROC-curve with the Youden's J statistic featured.

In the bottom left corner in Figure 2.3, where both the true positive and the false positive rates are equal to zero, is the result of when the threshold is set so that no samples are labeled as positive. In the top right corner in Figure 2.3, where both the true positive and the false positive rates are equal to one, is the result of when the threshold is set so that all samples are labeled as positive. Of course, neither of those two extremes is of much use. The space between the two extreme points represents the results when using other thresholds. The blue line in Figure 2.3 represent a hypothetical classification machine learning algorithm and the red dotted line represent the curve that would be obtained from an algorithm that randomly guesses the label. Thus, the curve of any useful classifier should definitely be above the red line. The further the distance is from the red line to the blue line, with the condition that the blue line is above the red line, the more efficient the classifier is. For an optimal classification method the blue line would pass through the point where the true positive rate is one and the false positive rate is zero [1].

By also selecting the point where the longest vertical distance from the red line to the blue line is in the curve, the optimal threshold for a specific classifier can be obtained. In this point, a higher True Positive rate will be at the cost of even higher False Positive rate, and a lower False Positive rate will be at the cost of even lower True Positive rate. This measurement is known as the Youden's J statistic [24] and is illustrated in Figure 2.3.

Another way of measuring the performance of a classifier is to measure the area under the ROC-curve, AUC. The further the distance is from the red line to the blue line for every threshold point, the more accurate is the classification method as previously stated and the larger the area under the curve becomes, thus making the value of the AUC a measurement of performance [1].

2.3.5 Matthews Correlation Coefficient

While the confusion matrix in Table 2.2 show the most important qualities of a binary classifier, there is often a need to transform that information into a single score in order to more easily compare different classifiers. There exists multiple metrics that try to do that and one of them is Matthews Correlation Coefficient, or MCC, introduced by Brian W. Matthews in 1975 [17]. MCC is calculated as

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}. \tag{2.18}$$

The MCC can be seen as the correlation between predicted and actual values of a binary classifier. It takes a value between -1 and 1. A value of 1 represents perfect prediction, and -1 the opposite, a classifier that gets every prediction wrong. Random prediction would give a MCC of 0, hence any useful classifier needs to produce a positive MCC.

It is generally regarded as one of the best measures [21] and commonly used in machine learning. One advantage of MCC over many other measures is that it takes into account the balance ratios of the four metrics in the confusion matrix, making it prevalence-independent and therefore applicable on asymmetrical data. In *"Ten quick tips for machine learning in computational biology"* Davide Chicco explains how other measures can be misleading and concludes

> "For these reasons, we strongly encourage to evaluate each test performance through the Matthews correlation coefficient (MCC), instead of the accuracy and the F1 score, for any binary classification problem." [7].

2.3.6 K-means

Another commonly used machine learning method to identify clusters is the K-means method. The essential idea of the method is to partition the input data into K clusters such that the squared error between each clusters empirical mean and the data point belonging to that cluster is minimized.

Consider a set of n data points $X = [x_1, x_2, \ldots, x_n]$, each with the dimensional d which are being clustered into a set of K clusters, $C = [c_1, c_2, \ldots, c_K]$. The empirical square error for every cluster c_k is defined as

$$J(c_k) = \sum_{x_i \in c_k} \|x_i - \mu_k\|^2. \tag{2.19}$$

Here μ_k is defined as the mean of the data points in cluster c_k. The objective of the K-means method is to organize the data points into certain clusters so that Equation 2.20 is minimized.

$$J(C) = \sum_{k=1}^{K} \sum_{x_i \in c_k} \|x_i - \mu_k\|^2 \tag{2.20}$$

The algorithm is divided into three steps:

1. Initial partition
2. Assigning each data point to the closest cluster center
3. Computation of new cluster centers

The first step randomly selects K data points of the input data which for the first iteration serve as its own cluster's center. In the second step, each data point is assigned to a cluster by measuring the distance from the data point to each cluster's center, choosing the one with the shortest distance. This distance measurement can be defined in various alternatives, but the most common one is to use the Euclidean distance described in Equation 2.6. The third step consists of calculating new cluster centers for each cluster c_k based on these new updated clusters, described in Equation 2.21.

$$\mu_k = \frac{1}{N} \sum_{i=1}^{N} x_i, \quad x_i \in c_k \tag{2.21}$$

Here N is the number of data points assigned to cluster c_k. The algorithm iterates over the two latter steps until the clusters are stabilized. The most critical choice when initializing the K-means algorithm is the choice of K. One issue with K-means is the initial partition of the cluster when randomly selecting data points which can result in different clusters for the same K. An unfortunate initial partition can lead to a local minimum in the final result. A way to overcome this issue is to run the method several times with the same K and consolidating the total square error of the final cluster. This can be readily done as the computational cost is low [11].

2.3.7 Bayes' Theorem

Bayes' theorem is used when calculating conditional probabilities and is essential in probability theory [14]. It is stated as

$$\mathbf{P}(A|B) = \frac{\mathbf{P}(B|A)\mathbf{P}(A)}{\mathbf{P}(B)}. \tag{2.22}$$

When using a binary classifier it is often of interest to know the conditional probability - if an event is classified as predicted positive (PP), what is the probability that the event is in fact positive (P). Using Bayes' theorem that probability is calculated as

$$\mathbf{P}(P|PP) = \frac{\mathbf{P}(PP|P)\mathbf{P}(P)}{\mathbf{P}(PP|P)\mathbf{P}(P) + \mathbf{P}(PP|N)\mathbf{P}(N)}. \tag{2.23}$$

2.4 Imbalanced Data

In machine learning it is often a problem if the data is highly imbalanced, meaning the number of samples in each class is very different. Consider a data set with a positive/negative ratio of 1:99. A classifier that labels all samples as negative would then get an accuracy of 99%, but would probably be of little use. To deal with this problem there exist different methods to balance the data.

2.4.1 SMOTE

One way of addressing this problem is to use a bootstrapping method and creating a new dataset based on the existing one where the labeled data has been oversampled. Synthetic Minority Over-sampling Technique (SMOTE) is a method used to create a new synthetic dataset where the minority class has been oversampled to compensate for the imbalanced original data set. The method uses, as implied by the name, new synthetic data instead of conventional bootstrapping with replacement. The new data is generated from the minority class by selecting a data point at random, finding its k nearest neighbors in the data set, selecting one of its neighbors, and constructing the new data point by multiplying the difference between these two data point with a number between 0 and 1 and adding it to the first data point. This new data point replaces an old data point of the majority class in the data set, and creates a larger and more general decision region for the minority class with the aim to improve the prediction accuracy when performing the machine learning algorithm [13]. The number of new synthetic data points generated is chosen to desired amount.

2.4.2 Undersampling

Another way of handling an imbalanced data set with the same objective as in the previous section is by undersampling the majority class in the set. One easy way of doing this is to simply not include all data points from the majority class while including all data points from the minority class, before having the algorithm train the data.

Chapter 3

Data

3.1 One-hot Encoder

As previously stated, the basis of machine learning is the ability to transform and organize input data into output data, and the algorithm for the SOM relies on the premise that the input data among itself is comparable in some manner. When dealing with numerical data where the size of the value is of importance, comparing data is quite intuitive since the smaller the distance between two data points is, the more similar they are. However, a predicament appears when the data points include categorical data since the distance between that sort of data can not be measured in the same manner.

Consider that one of the features in the input data is Country and it has different attributes, for example Sweden and Denmark. One way of dealing with this problem is to assign a value to each country, Sweden is assigned 1 and Denmark is assigned 0. Two data points with the same country-attribute according to this premise will have a distance of zero, and different country-attributes will give a distance of one. In the case of two attributes for a specific feature, this approach is sufficient, but a problem arises if a third attribute is introduced. If Norway is introduced and assigned the value 2, this results in a larger distance between Denmark and Norway than between Denmark and Sweden, which there does not have to be a basis for and makes the approach not as sufficient. When introducing more countries, this problem grows larger.

One solution to this problem which produces unbiased differences between categorical data is to use one-hot encoding on the input data, also known as using dummy variables. When applying one-hot encoding on categorical data, it can be seen as giving every unique attribute for a specific feature its own dimension. If a feature consists of N different attributes, that feature gets assigned a vector of length N, with a one placed in the vector to indicate which unique attribute its representing. The rest of the vector is empty. In the example with three different country-attributes, Sweden, Denmark and Norway, Sweden is represented by [1 0 0], Denmark by [0 1 0] and Norway by [0 0 1]. With this method, a feature of three dimensions has been created which is linearly separable, allows all combinations of countries and the Euclidean distances between all countries are equal. This method can be applied to features with an arbitrary number of unique attributes [5].

3.2 Audit Changed Trades

The data used in this thesis is a sample from a large dataset of Audit changed trades retrieved by SHB. Out of all the trades, 63 has been labeled as bad, henceforth referred to as the labeled data. These labeled data points are manually constructed by SHB to mimic high risk trades and the goal for SHB is to be able to predict if a new trade falls into this category. The number of total data points used is 9 258. Each trade has 15 features of which most are categorical with a certain number of unique attributes, see Table 3.1.

Table 3.1: An overview of the features and attributes of the data

Features	Sample Attributes	Unique Attributes
Changes	Pricing Issues, Fost Cancellation	59
Counterparty	Counterparty, Intern Dept	2
Portfolio	Internal names	110
Trader	ID number	123
Update User	ID number	111
Group	AMBA, FX, IR TRADING	18
Hour	Hour during the day	24
Instrument Name	EUR, GBP, SEK	30
Instrument Type	FXforward, FXspot, FXswap	3
Region	SE, DE, EE	16
Weekday	Monday, Tuesday, Wednesday	6
Delta	Days between creation and update	Numerical
Nominal	The nominal value of a trade	Numerical
Price	The price of the asset	Numerical
Seconds	Seconds between creation and update	Numerical

The One-hot encoding method described in the previous section applied to the data set used in this thesis generates a data set with 506 attributes for each of the 9,258 data points, arranged in the manner which can be seen in Table 3.1. This means that the first 59 attributes for every data point is attributed to Changes, consisting of 1 one and 58 zeros. The same principle refers to all of the categorical data, and the numerical data is normalized to get a value between one and zero attributed to it. The normalized value is calculated by taking the difference between the lowest and the highest number in one particular feature, subtracting the lowest number from each trade in that feature and dividing it by this difference.

All of the trades in the data set are not complete, but the amount of missing attributes is small in relation to the whole data set. For the features containing missing values, a new attribute Missing Value has been added.

Chapter 4

Method

This chapter firstly describes the methodology to implement the algorithm of a SOM, what the initial settings were and how it was customized for this application. Then the methods used to cluster the trained SOM and the methods used to balance the data set are described. It ends with the different methods used to evaluate the performance of the algorithm, which includes visualization of the SOMs and methods to get hard numbers on the prediction ability of the algorithm, as well as comparison with well-known methods.

4.1 SOM Algorithm

The initial settings for the algorithm used in every simulation were:

- The map was organized in a rectangular grid.
- The selection of the best matching unit was based on the Euclidean distance measurement.
- The neighborhood distance used the Manhattan distance measurement.
- The neighborhood settings described in detail in section 2.2.4.
- L_0 from Equation 2.14 was set to 0.1.

In the previous work conducted by SHB [20] a rectangular grid was used. For that reason the same choice was made here. This choice was not obvious and it would be interesting to explore other choices, e.g. a hexagonal grid which is used in many applications of SOMs. However, that was outside the scope of this work. When using a rectangular grid it makes sense that each neuron that is not at an edge has four neighbors, i.e. there are no diagonal relations between neurons. That was achieved by using the Manhattan distance as the neighborhood distance. For finding the BMU the Euclidean distance was the more logical measure and what Kohonen used in his original paper [15].

The Gaussian neighboring function in Equation 2.15 is widely used and proven to be the best option by P. Stefanovic and O. Kurasova in [25]. In the same paper they showed that the choice of learning rate can improve the results. Again, investing that was outside the scope of this work, hence the choice of learning rate was based on the previous work done by SHB which showed promising results.

The implementation of the SOM training algorithm is described in pseudo code in Algorithm 1.

Algorithm 1 SOM training

```
1:  Create Map
2:  Initial Neurons
3:  Initial Learning rate
4:  Initial Neighborhood
5:  for <Number of Iterations> do
6:      Update Learning rate                                ▷ Equation 2.14
7:      Update Neighborhood                                 ▷ Equation 2.11
8:      Choose Data point at random
9:      for all Neurons do                                  ▷ The Decision Step
10:         Calculate distance between Neuron and Data point
11:     BMU ← Neuron that gives smallest distance
12:     for all Neurons in Neighborhood of BMU do           ▷ The Update Step
13:         Update Neuron                                   ▷ Equation 2.3
```

After the training of the neurons' weights depending on the input data, for every data point the BMU in the map was determined and a final placement for each data point was assigned. The amount of labeled and unlabeled data points placed at each neuron was stored in two separate matrices.

4.2 Clustering

The idea was that the SOM algorithm would cluster the data and hopefully in such a way that the labeled data would be contained in the same cluster, referred to as the labeled cluster. Previous work by SHB supported this hypothesis [20]. This put the question of how the labeled cluster should be identified in the trained SOM. Below the methods used to do that are described.

First a center point of the labeled cluster had to be determined. This was done by calculating the weighted placement of the labeled data in accordance with Equation 4.1,

$$\bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i \pi_i, \quad \bar{y} = \frac{1}{N}\sum_{i=1}^{N} y_i \pi_i. \tag{4.1}$$

N stands for the number of neurons that have at least one labeled data point assigned to it, π_i stands for the number of data points assigned to the i^{th} neuron in the set, and x_i and y_i are the coordinates for i^{th} neuron.

When the center point had been determined, the labeled cluster was chosen as all neurons within a certain distance of the center point that had at least one labeled data point assigned to it. Three different distance measurements were used for this, the Manhattan distance, the Euclidean distance and the Chebyshev distance, all described in Section 2.2.3. To determine the optimal distance for each cluster, the ROC-curve was used together with Youden's J Statistic, described in Section 2.3.4.

4.3 SMOTE Algorithm

The input data consisted of 63 labeled data points out of 9 258, which could be too little to draw robust and trustworthy conclusions from. Therefore the effect of applying the SMOTE-technique on the input data was explored. Before applying the training algorithm on the input data, the data set was run through the implemented SMOTE-algorithm described in Algorithm 2.

After the SMOTE-algorithm had been applied to the input data, the data set consisted of numerous more synthetic labeled data point which was used in the training of the SOM. The number of synthetic generated data points for the simulations was provided with each result when the algorithm was applied.

Algorithm 2 SMOTE

Input: Number of minority class samples T; Amount of oversampling N%; Number of nearest neighbors k; Total data set Data.
Output: A data set with T*N/100 synthetic data points.

```
 1: n = T*N/100
 2: Sample[ ][ ]: Matrix of $n$ random selected original minority class samples
 3: narray[ ][ ]: Matrix to store $k$ nearest neighbors
 4: for i← 1:n do
 5:     Dist[ ]: Distance between row $i$ in Sample and every row in Data.
 6:     narray[$i$]: Store the indices of the $k$ smallest values of Dist
 7: Synthetic[ ][ ]: Matrix to store n synthetic samples
 8: for i← 1:n do
 9:     Choose a random number between 1 and k, name it $nn$.
10:     for attr←1:Number of attributes do
11:         Compute: $dif$ = Data[$nn$][attr]-Sample[$i$][$attr$]
12:         Compute: $gap$ = random generated number between 0 and 1
13:         Synthetic[$i$][attr]=Sample[$i$][$attr$] + $gap + dif$
14: tempData: Copy of Data
15: while i less than n do
16:     Choose a random number $r$ between 1 and number of rows in Data
17:     if Index $r$ in Data is not of minority class then
18:         tempData[$r$] = Synthetic[$i$]
19:         $i = i + 1$
20: return tempData
```

4.4 Visualization

An attractive feature of SOMs is the ability to visualize high-dimensional data in a low-dimensional space, which enables interpretation by the human eye. There are two different ways of visualizing SOMs, either displaying the output in input space or displaying the input in output space. In this application the input space had 506 dimensions, which of course was too high for any useful visualization. Instead the SOMs were visualized in the output space consisting of a two-dimensional quadratic map of neurons.

To investigate the performance of the SOM, several plots were made. Figure 4.1 shows a plot of the final placement of all trades after the training. For each neuron in the map a circle was plotted if that neuron was the BMU for a data point, and the more data points for which that neuron was the BMU, the bigger the circle was. This was done separately for labeled and unlabeled trades and to distinguish them they were mapped in different colors. They were also scaled differently because of the asymmetrical data. By plotting the trades in this way it was possible to get an indication of how well the SOM clustered the labeled data.

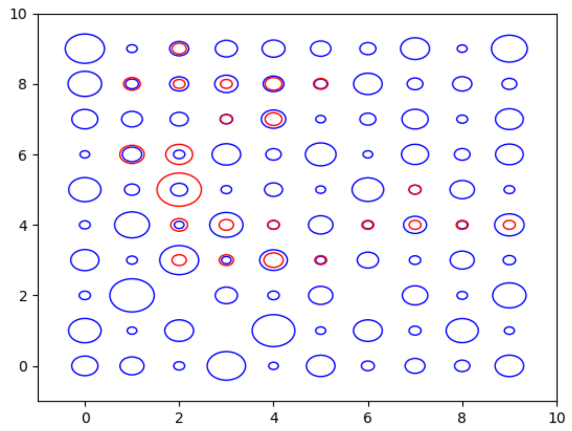


Figure 4.1: Visualization of a 10x10 SOM

In Figure 4.1 trades that had been placed near each other were likely to be in the same cluster, but it was not necessarily the case. To determine if that was the case an U-matrix was used, which shows clusters more clearly.

4.5 Convergence

The method used to see if the SOM converges was to visually inspect how the map changed. With constant intervals during the training phase the map was plotted as in Figure 4.1. If big transformations of the map was observed towards the end of the training phase, that would indicate that the map was not stable.

4.6 Model Evaluation

The main aim of the algorithm was to predict new trades, i.e. giving a label to trades that have not been used in the training of the SOM. To evaluate the prediction ability, the Cross-Validation approach described in Chapter 2 was used. The input data was initially divided into ten roughly equally large samples. Each of these subsets were left out in the training of the SOM algorithm at one occasion to be used as test data while the remaining nine subsets were used as training data. In the simulations where the SMOTE-algorithm was applied, only the training data was altered by the oversampling on the account of not having any offspring of the test data in the training data.

Two measurements generated by the Cross-Validation in order to capture the performance of the algorithm were the probability that a trade inside the labeled cluster actually was a labeled trade, P$(P|PP)$, and the probability that a trade outside the identified cluster actually was an unlabeled trade, P$(N|PN)$, both according to Bayes' Theorem described in Section 2.3.7. The components to calculate these probabilities were the following:

- P$(PP|P)$ is the True Positive data points divided by the number of labeled data points in the test set.
- P(P) is the number of labeled data points divided by the total number of data points in the test set.
- P$(PP|N)$ is the number of False Positive data points divided by the number of unlabeled data points in the test set.
- P(N) is the number of unlabeled data points divided by the total number of data points in the test set.
- P$(PN|N)$ is the number of True Negative data points divided by the number of unlabeled data points in the test set.
- P$(PN|P)$ is the number of False Negative data points divided by the number of labeled data points in the test set.

The MCC described in Section 2.3.5 was the measurement used to compare different models, because of the reasons mentioned in that section.

4.7 Simulations

Several simulations were run in order to test different sets of parameters and to see which performed better. The parameters that were changed were the number of iterations, the size of the map, the cluster method, and whether or not SMOTE was used. First a simulation to get the visual results was done for each set of parameters. Then the prediction ability of each set was tested with Cross-Validation. Finally, some parameters, e.g. the map size, were kept unchanged while some parameters as well as the robustness and variance of the models were further investigated.

4.8 K-means

To test the effectiveness of the SOM algorithm and if there are any incentives for using the method, the outcome of the algorithm was compared to another clustering method which is commonly used in machine learning, the K-means method which was discussed earlier in Chapter 2.

The only parameter which was of importance and could be initialized in this method was the number of K:s, i.e. the number of clusters to identify. The measurements derived from this method to determine its performance in relation to the SOM was from the Cross-Validation approach and by measuring the MCC-value and using Bayes' Theorem the same way as in the previous section. To identify a cluster in the best way, the ROC-curve was applied once more. With the K-means method, the thresholds were represented by the number of clusters with the highest amount of labeled data points which were included in a larger cluster. This means that the identified cluster in which data points was considered as True Positives and False Positives might consist of several sub-clusters. The same sort of input data set with categorical data altered by One-hot encoding, and both data applied with and without the SMOTE-algorithm was tested. When the method had been tested with several K:s, a K with a high-performing results was selected to further test the method with that parameter and some further alterations, just like with the SOM.

Chapter 5

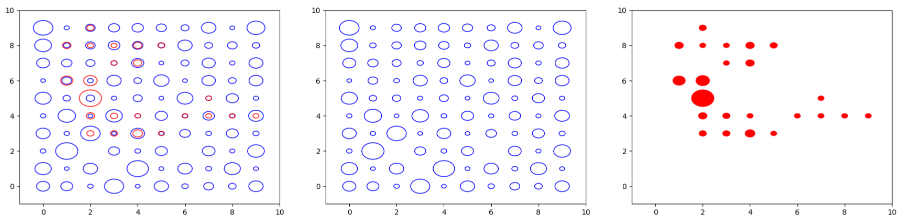
Results

Before the hard numbers from the cross-validation are presented, the visual results of the SOM algorithm is provided.

5.1 Visual Results of the SOM Algorithm

The SOMs presented are of sizes 10x10, 20x20, 30x30 and 40x40, and each size is done with and without SMOTE. The SMOTE is done with an over-sampling of 1000%. Thus there are eight different SOMs. All SOMs are trained with 1 000 000 iterations. The number of iterations have shown to have little impact on the visual results. For two different simulations with the same parameters, the resulting figures can look very different. However, the performance of the algorithm is stable when it comes to how it clusters the data.

10x10 SOM



(a) All trades (b) Unlabeled (c) Labeled

Figure 5.1: **10x10 SOM**: Final map after training. The map is visualized as described in 4.4. The labeled data has been somewhat ordered, but is spread out and does not seem to form a cluster.

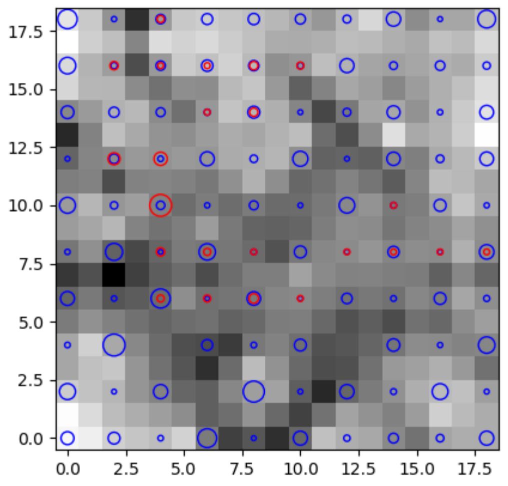


Figure 5.2: **10x10 SOM**: U-matrix together with the data. The U-matrix is the black-and-white background calculated as described in 2.3.1. On top of the U-matrix is the same plot as in Figure 5.1a. A cluster of light colors separated by dark colors can be seen in the lower left corner. There is no cluster of labeled trades.

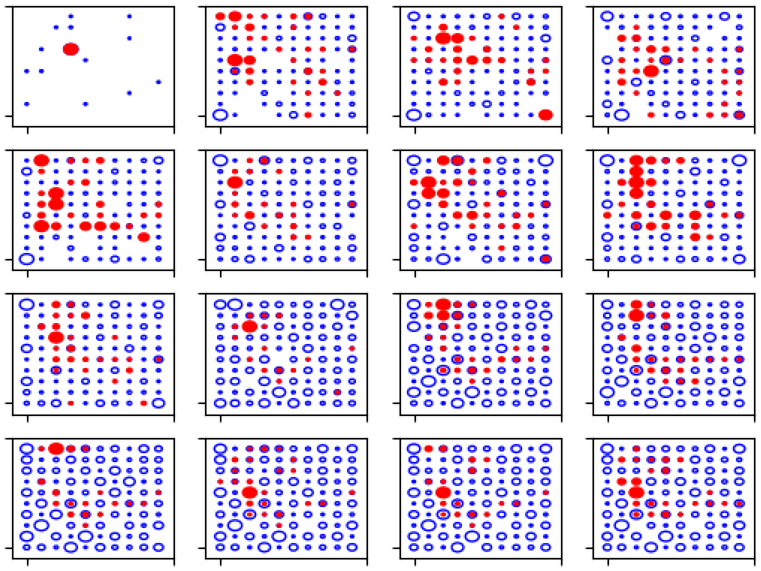
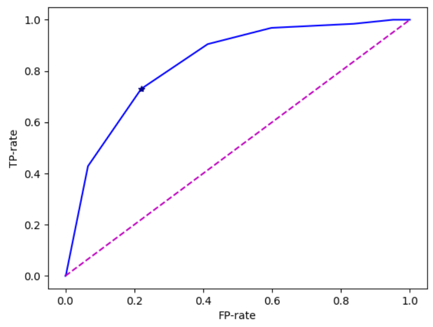
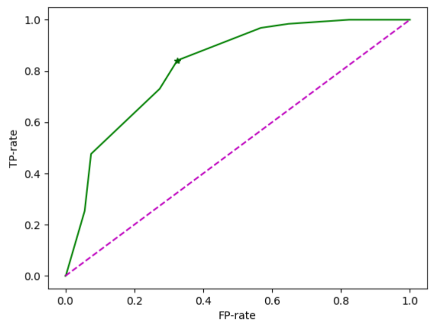


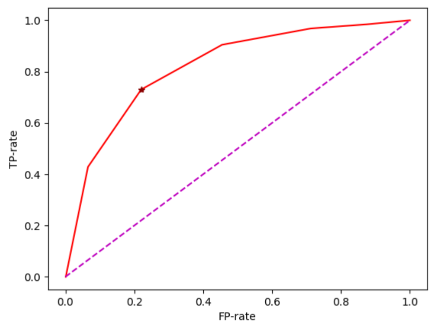
Figure 5.3: **10x10 SOM**: Plot of how the SOM changes during the training phase. The upper left plot shows the SOM after the initialization, before the training phase has begun. Going from left to right, up to down, the plots show the SOM with constant intervals. The plot at the top, second from the left shows the SOM after 1 000 000/15=66 667 iterations, the plot to the right of that one shows the SOM after 133 333 iterations, and so on. The lower right plot shows the final SOM after the training is completed, the same as in 5.1a. As the training progresses the SOM changes less.



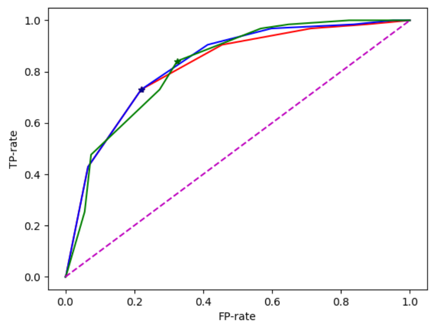
(a) Euclidean



(b) Manhattan

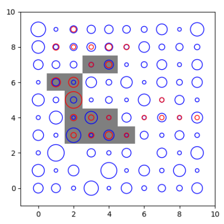


(c) Chebyshev

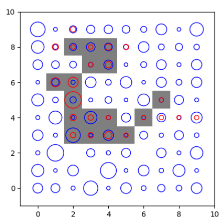


(d) All

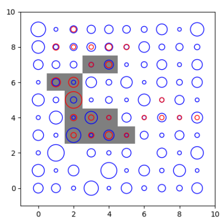
Figure 5.4: **10x10 SOM**: ROC-curves of the three different cluster methods. The asterisks show the points with the highest J-statistic.



(a) Euclidean



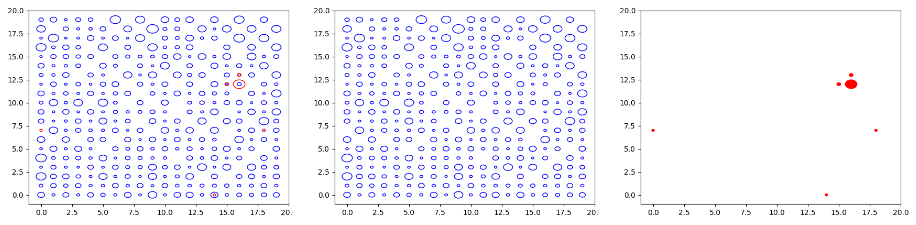
(b) Manhattan



(c) Chebyshev

Figure 5.5: **10x10 SOM**: The optimal clusters according to the J-statistic. The dark area shows which neurons each cluster includes. In this case the Euclidean cluster and the Chebyshev cluster are the same.

20x20 SOM



(a) All trades (b) Unlabeled (c) Labeled

Figure 5.6: **20x20 SOM**: Final map after training. The labeled data has been clustered in three neurons.

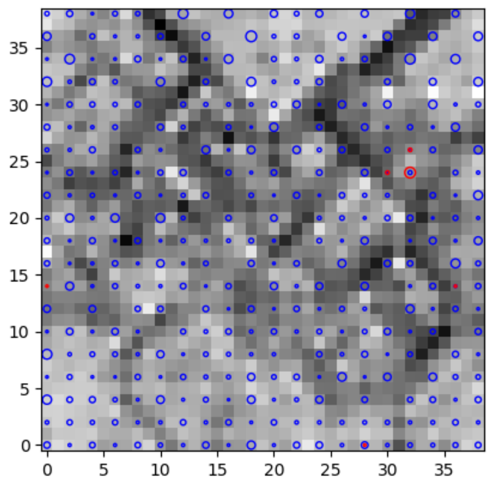


Figure 5.7: **20x20 SOM**: U-matrix together with the data. The dark strings around the three neurons with labeled data confirms the cluster.

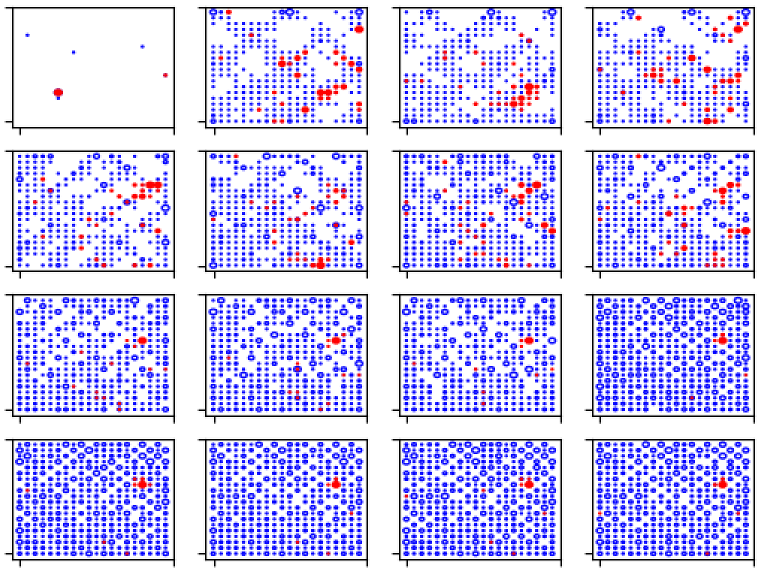
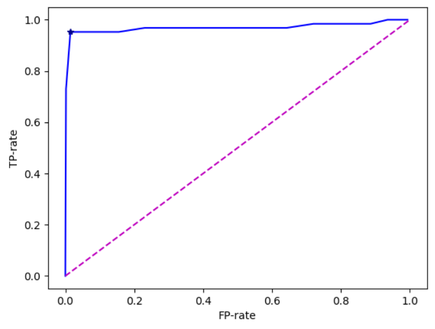
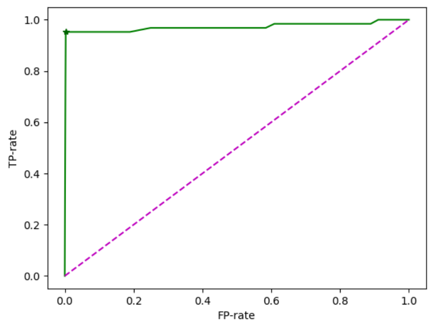


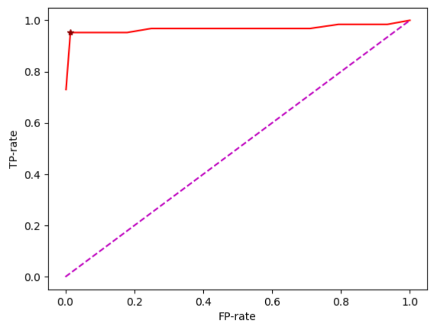
Figure 5.8: **20x20 SOM**: Plot of how the SOM changes during the training phase.



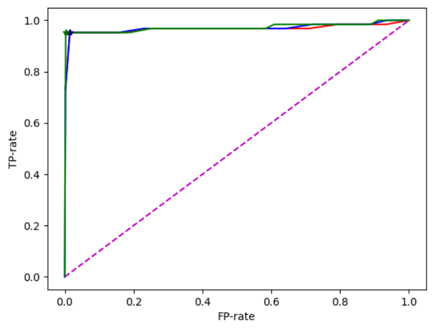
(a) Euclidean



(b) Manhattan

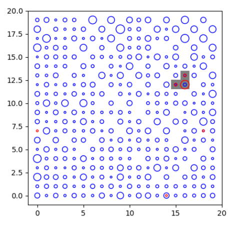


(c) Chebyshev

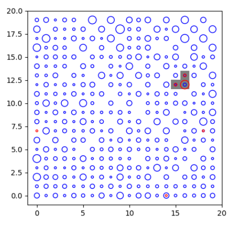


(d) All

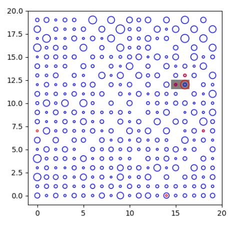
Figure 5.9: **20x20 SOM**: ROC-curves of the three different cluster methods. The asterisks show the points with the highest J-statistic. All three ROC-curves are almost perfect.



(a) Euclidean



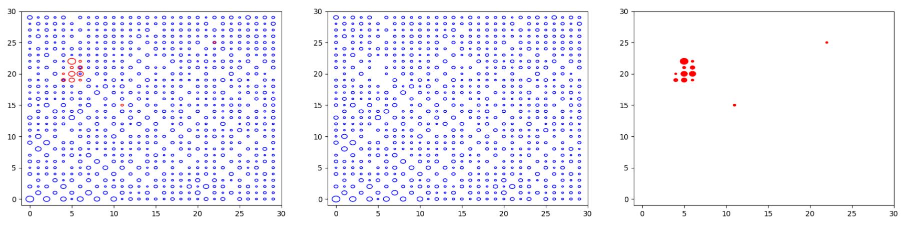
(b) Manhattan



(c) Chebyshev

Figure 5.10: **20x20 SOM**: The optimal clusters according to the J-statistic. The Chebyshev cluster differs from the other two.

30x30 SOM



(a) All trades (b) Unlabeled (c) Labeled

Figure 5.11: **30x30 SOM**: Final map after training. The map is visualized as described in 4.4. The labeled data has been clustered to the left.

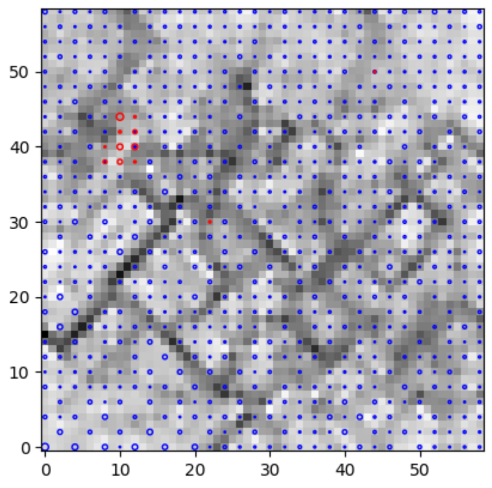


Figure 5.12: **30x30 SOM**: U-matrix together with the data.

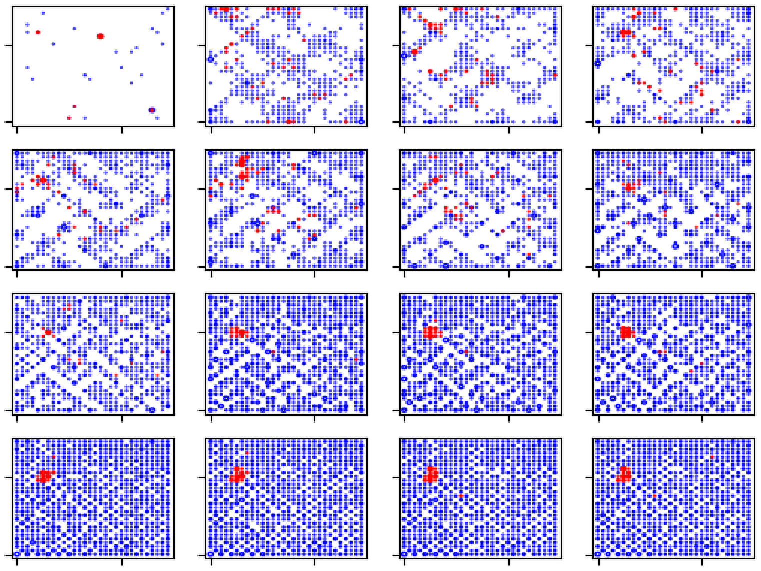
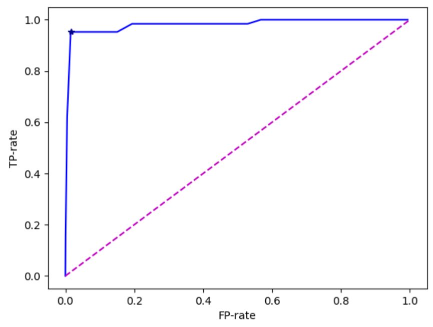
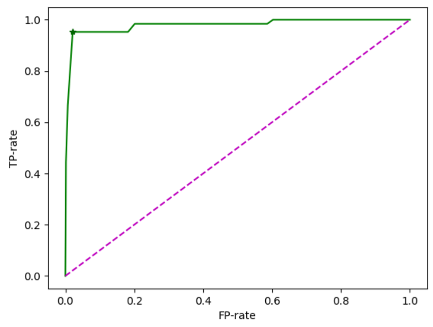


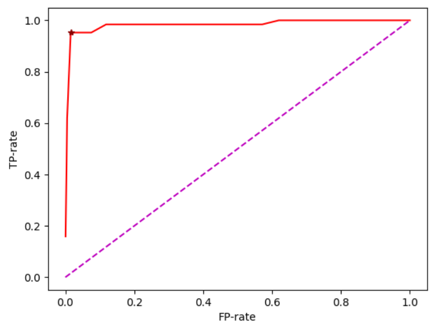
Figure 5.13: **30x30 SOM**: Plot of how the SOM changes during the training phase.



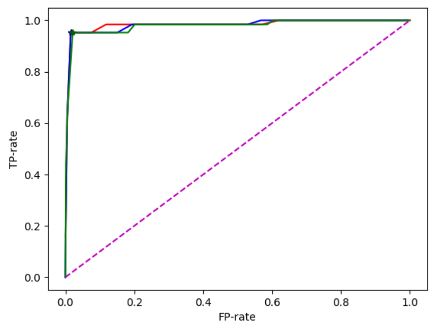
(a) Euclidean



(b) Manhattan

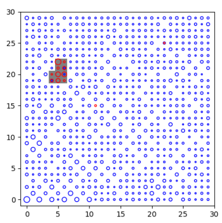


(c) Chebyshev

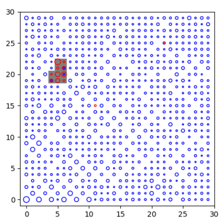


(d) All

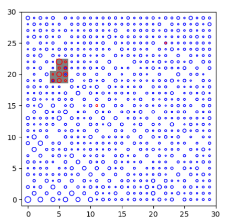
Figure 5.14: **30x30 SOM**: ROC-curves of the three different cluster methods. The asterisks show the points with the highest J-statistic.



(a) Euclidean



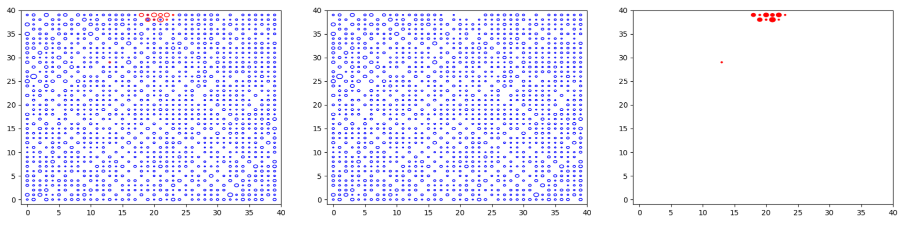
(b) Manhattan



(c) Chebyshev

Figure 5.15: **30x30 SOM**: The optimal clusters according to the J-statistic. All three clusters are the same.

40x40 SOM



(a) All trades (b) Unlabeled (c) Labeled

Figure 5.16: **40x40 SOM**: Final map after training. Outside the cluster at the top, only one neuron contain labeled trades.

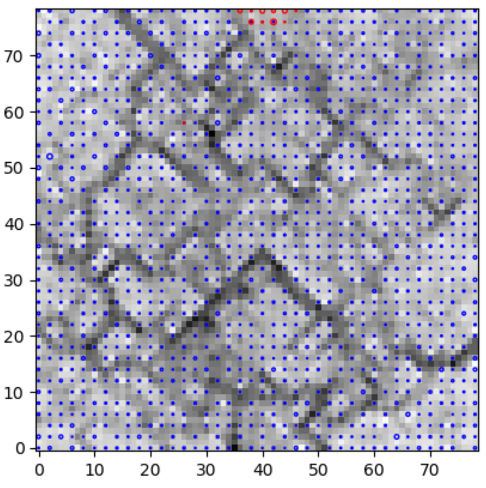


Figure 5.17: **40x40 SOM**: U-matrix together with the data. The cluster of labeled trades at the top is similar to its surrounding and not clearly defined.

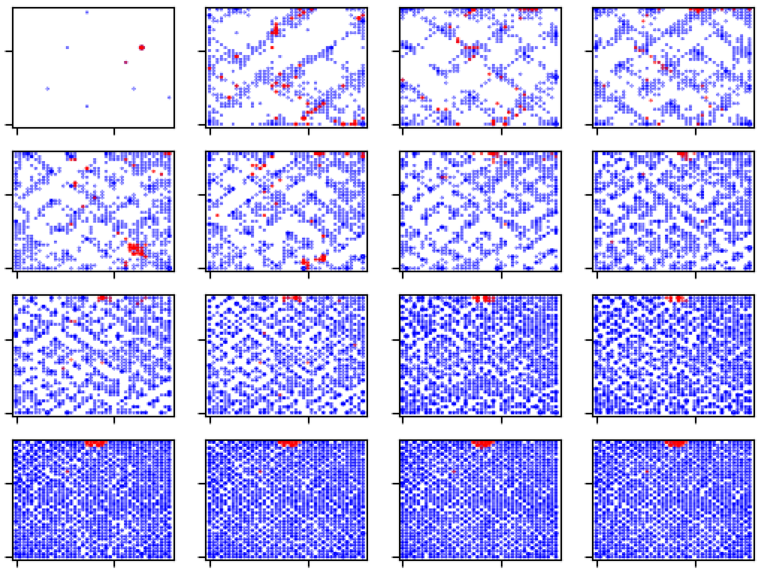
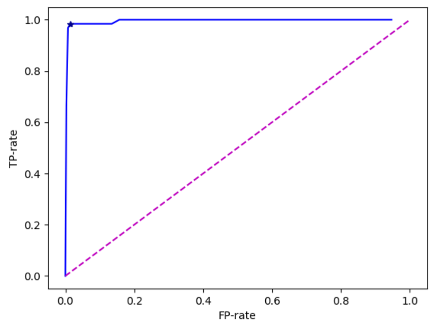
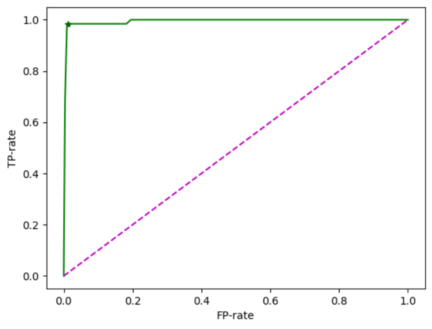


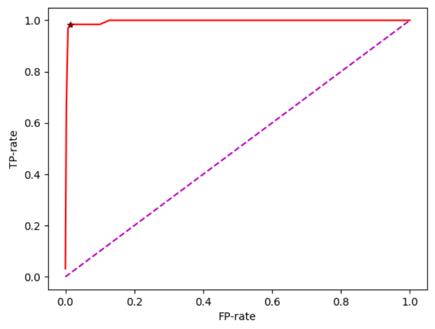
Figure 5.18: **40x40 SOM**: Plot of how the SOM changes during the training phase. Interesting to note is that the labeled data first forms a cluster in lower right of the map before the cluster a the top is formed. That could indicate a convergence issue.



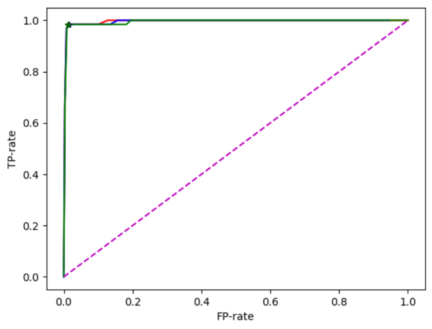
(a) Euclidean



(b) Manhattan

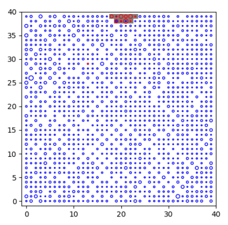


(c) Chebyshev

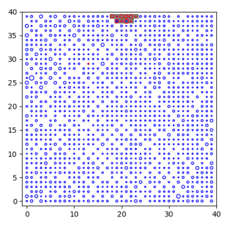


(d) All

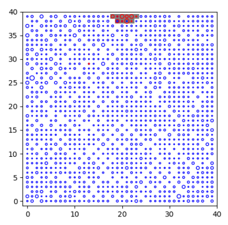
Figure 5.19: **40x40 SOM**: ROC-curves of the three different cluster methods. The asterisks show the points with the highest J-statistic.



(a) Euclidean



(b) Manhattan



(c) Chebyshev

Figure 5.20: **40x40 SOM**: The optimal clusters according to the J-statistic. All three clusters are the same.

10x10 SOM with SMOTE

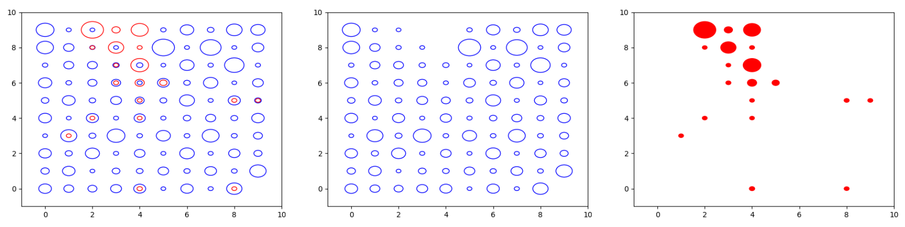


Figure 5.21: **10x10 SOM with SMOTE**: Final map after training. Compared to the 10x10 SOM without SMOTE in Figure 5.1, the use of SMOTE seems to lead to better exclusion of unlabeled trades in the area where the labeled trades have been clustered.

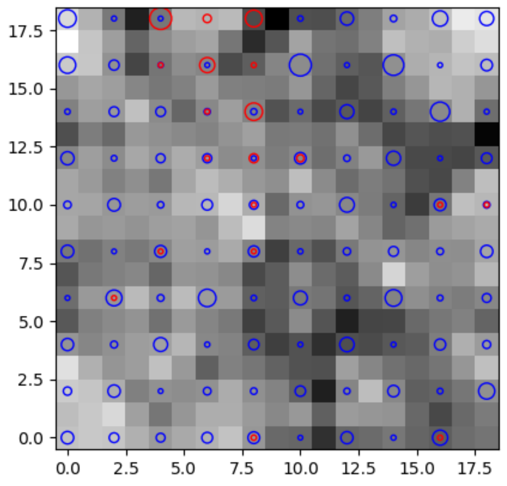


Figure 5.22: **10x10 SOM with SMOTE**: U-matrix together with the data. The labeled trades does not form a clear cluster.

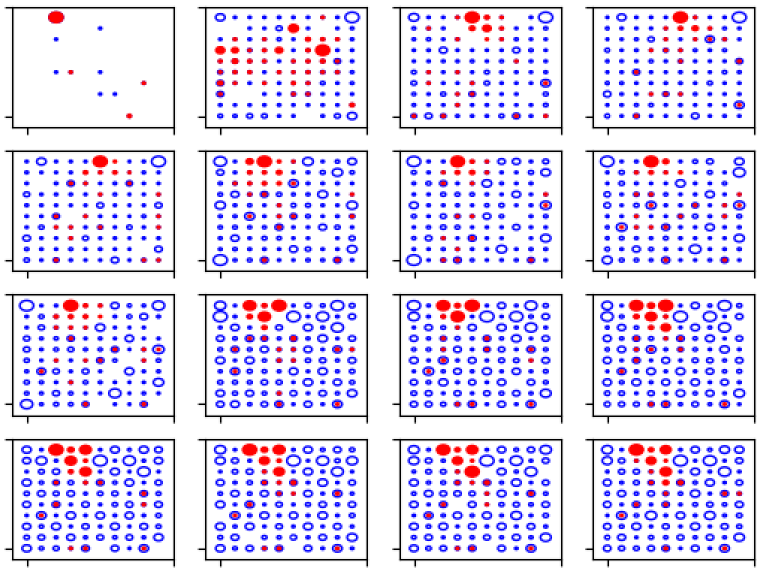
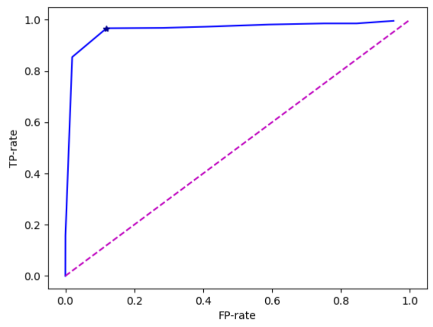
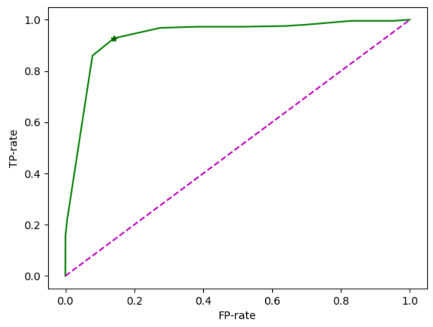


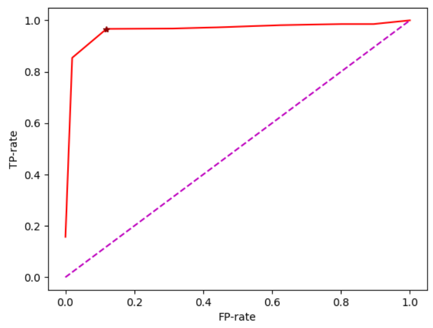
Figure 5.23: **10x10 SOM with SMOTE**: Plot of how the SOM changes during the training phase.



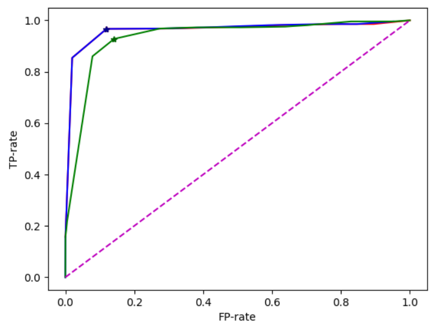
(a) Euclidean



(b) Manhattan

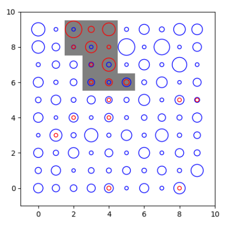


(c) Chebyshev

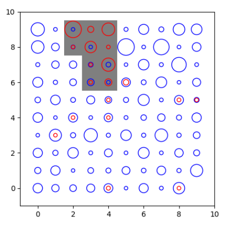


(d) All

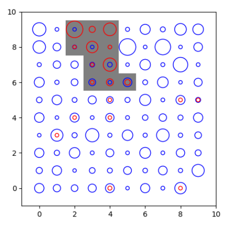
Figure 5.24: **10x10 SOM with SMOTE**: ROC-curves of the three different cluster methods. The asterisks show the points with the highest J-statistic.



(a) Euclidean



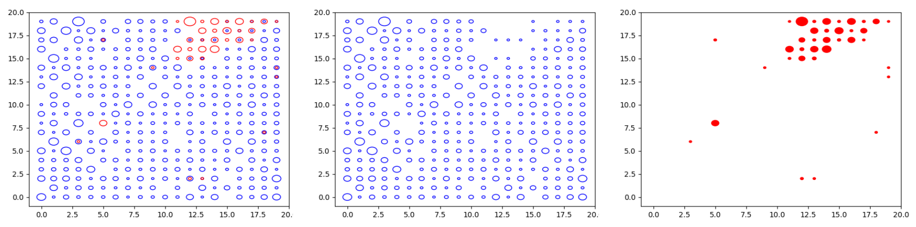
(b) Manhattan



(c) Chebyshev

Figure 5.25: **10x10 SOM with SMOTE**: The optimal clusters according to the J-statistic. The Manhattan cluster differs from the other two.

20x20 SOM with SMOTE



(a) All trades (b) Unlabeled (c) Labeled

Figure 5.26: **20x20 SOM with SMOTE**: Final map after training. The exclusion of unlabeled trades in the area with labeled trades is clear.

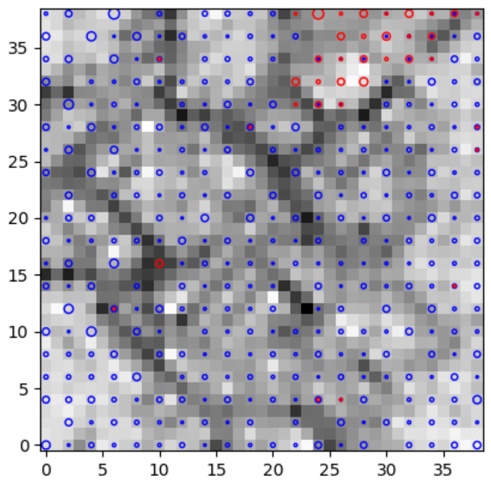


Figure 5.27: **20x20 SOM with SMOTE**: U-matrix together with the data. This U-matrix is very promising, the labeled trades at the top are all in a clear cluster. It also reveals another interesting thing. In Figure 5.26c the neuron at position $(5, 8)$ with labeled trades seems to be an outlier. The U-matrix shows that the neuron in fact should be considered to be a cluster, since it is different from its surrounding. Also, that neuron only contains labeled trades. Hence, there are two clusters of labeled trades. Unfortunately, the method used in this report can not catch that.

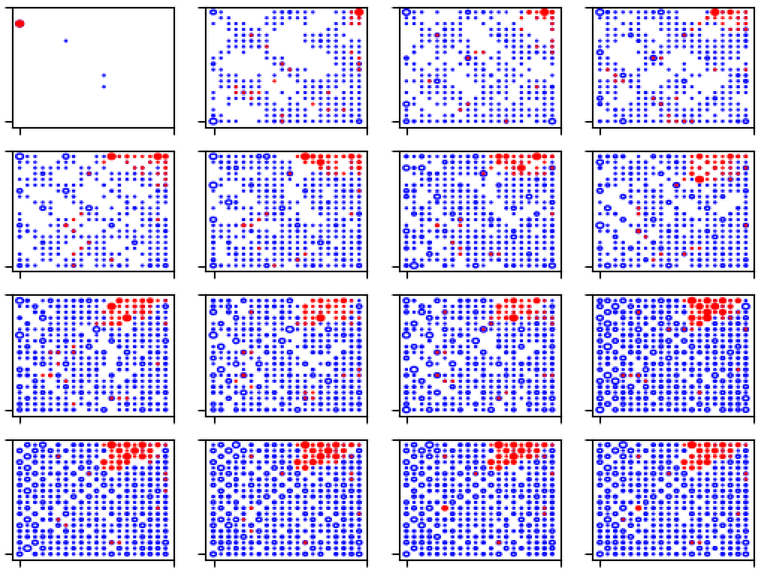
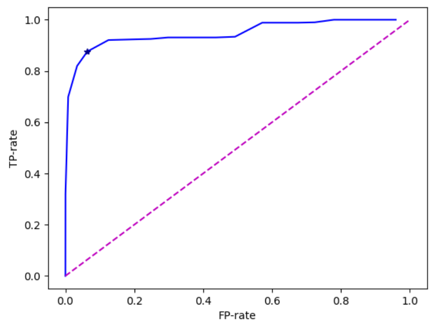
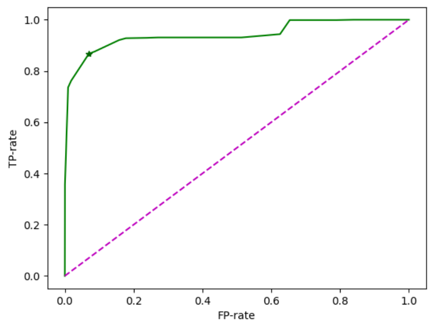


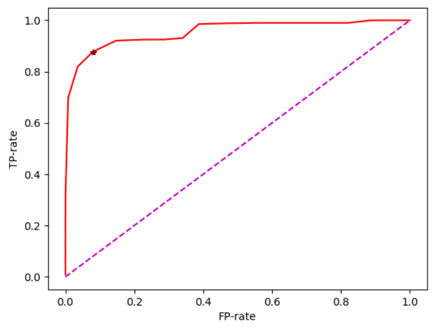
Figure 5.28: **20x20 SOM with SMOTE**: Plot of how the SOM changes during the training phase.



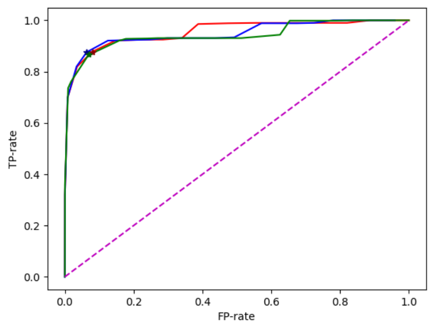
(a) Euclidean



(b) Manhattan

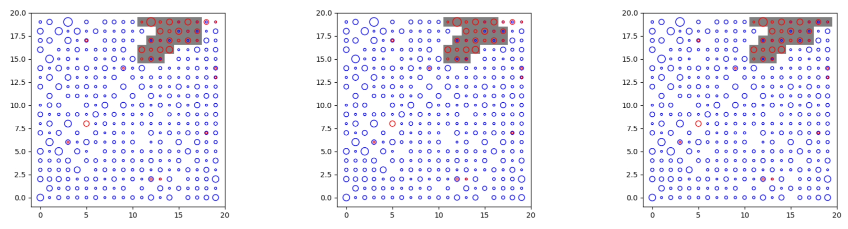


(c) Chebyshev



(d) All

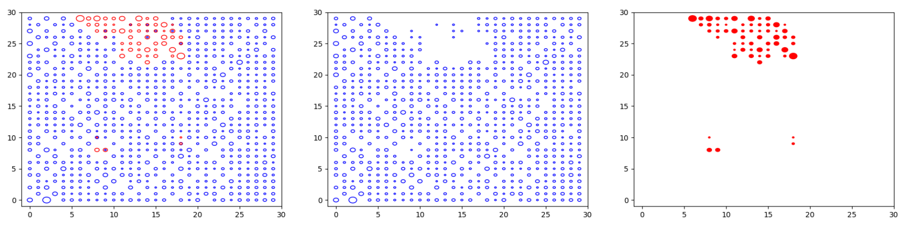
Figure 5.29: **20x20 SOM with SMOTE**: ROC-curves of the three different cluster methods. The asterisks show the points with the highest J-statistic.



(a) Euclidean (b) Manhattan (c) Chebyshev

Figure 5.30: **20x20 SOM with SMOTE**: The optimal clusters according to the J-statistic. The U-matrix in Figure 5.27 shows that the neurons in the top right corner are part of the cluster, but only the Chebyshev cluster includes all of them.

30x30 SOM with SMOTE



(a) All trades (b) Unlabeled (c) Labeled

Figure 5.31: **30x30 SOM with SMOTE**: Final map after training. The exclusion of unlabeled trades in the area with labeled trades is clear.

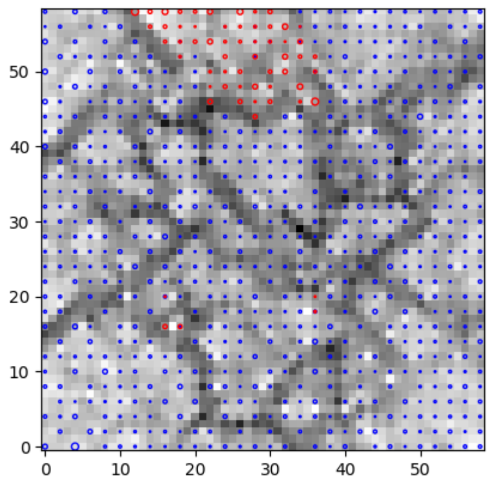


Figure 5.32: **30x30 SOM with SMOTE**: U-matrix together with the data. The U-matrix confirms the cluster at the top. The pattern revealed in Figure 5.27, that the 20x20 SOM with SMOTE contains a second cluster of labeled trades, can be seen here as well. The two neurons in the lower left with labeled trades clearly form a cluster.

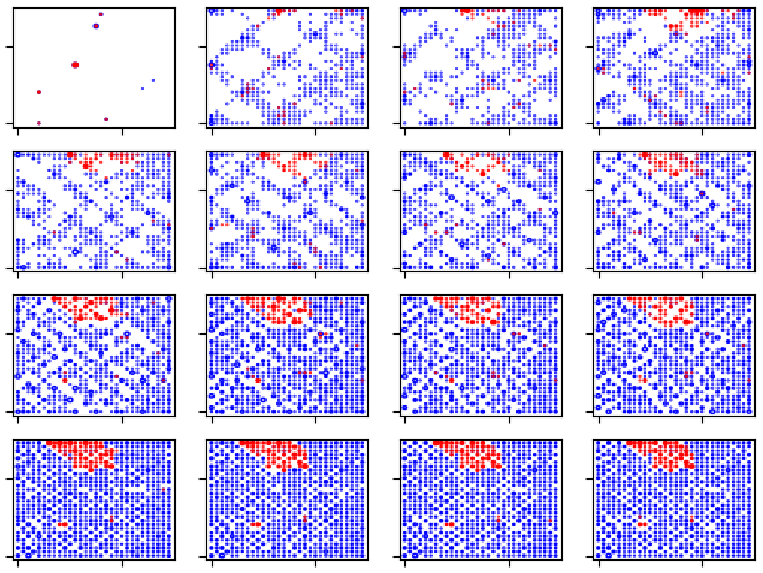
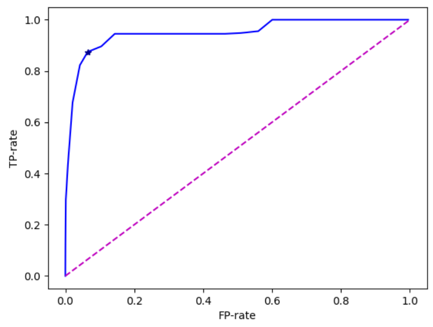
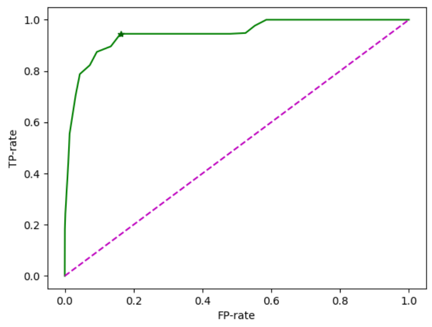


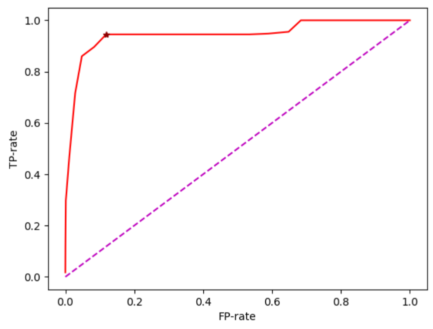
Figure 5.33: **30x30 SOM with SMOTE**: Plot of how the SOM changes during the training phase.



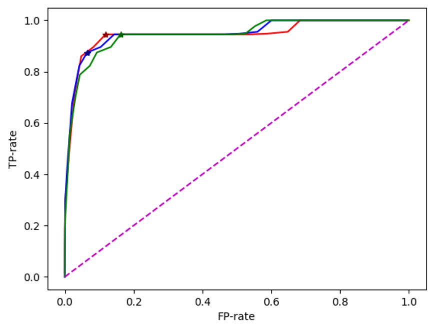
(a) Euclidean



(b) Manhattan

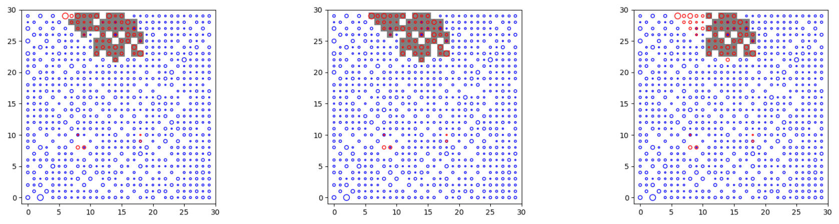


(c) Chebyshev



(d) All

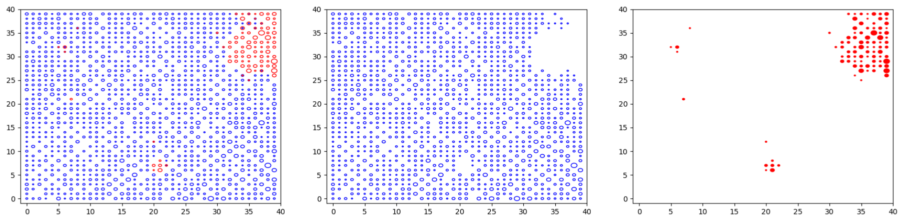
Figure 5.34: **30x30 SOM with SMOTE**: ROC-curves of the three different cluster methods. The asterisks show the points with the highest J-statistic.



(a) Euclidean (b) Manhattan (c) Chebyshev

Figure 5.35: **30x30 SOM with SMOTE**: The optimal clusters according to the J-statistic. The U-matrix in Figure 5.32 clearly shows that the right tail of the area of labeled trades is part of the cluster. Only the Manhattan cluster includes the whole tail.

40x40 SOM with SMOTE



(a) All trades (b) Unlabeled (c) Labeled

Figure 5.36: **40x40 SOM with SMOTE**: Final map after training. It is apparent that there is a second cluster towards the bottom.

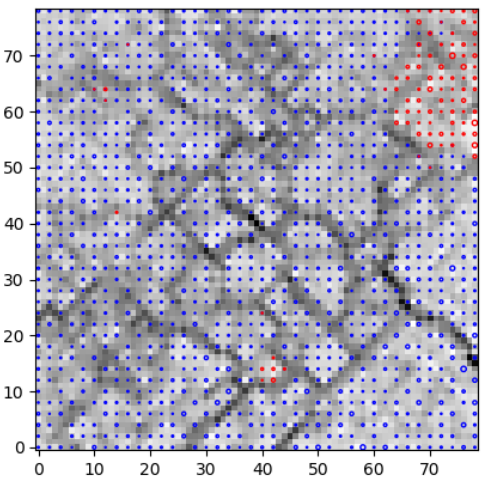


Figure 5.37: **40x40 SOM with SMOTE**: U-matrix together with the data. The U-matrix confirms that the second cluster towards the bottom is clearly defined. There is even an argument for a third cluster in the upper left.

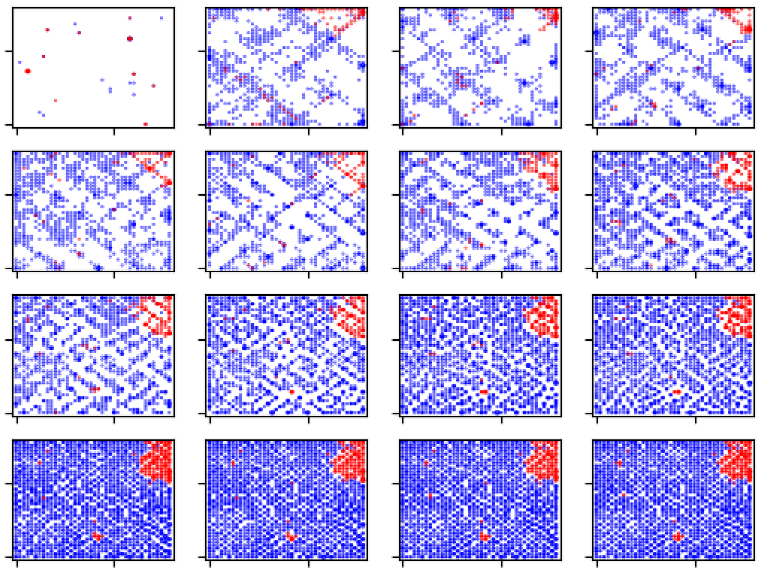
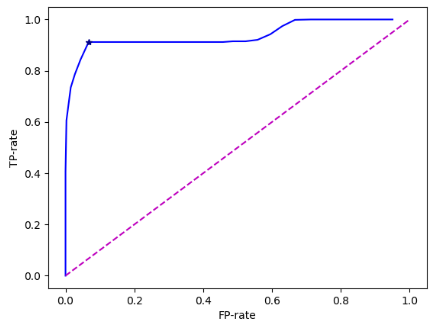
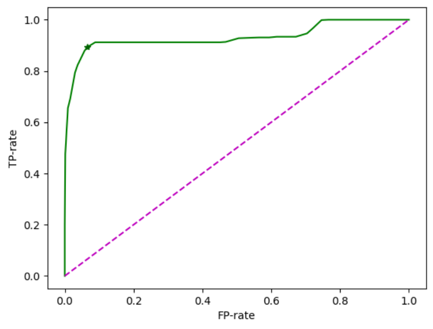


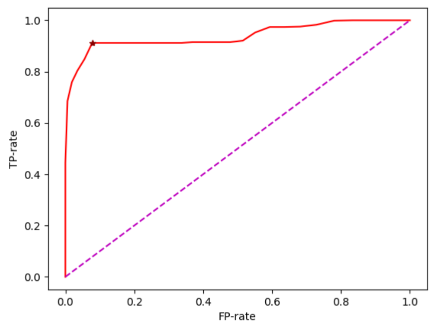
Figure 5.38: **40x40 SOM with SMOTE**: Plot of how the SOM changes during the training phase.



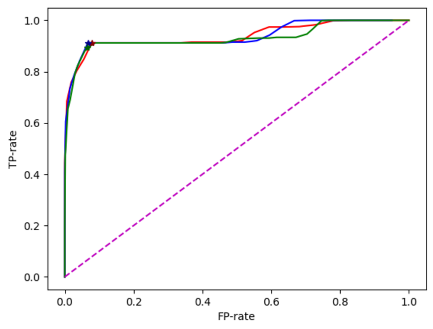
(a) Euclidean



(b) Manhattan

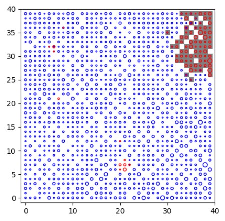


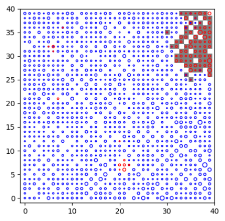
(c) Chebyshev



(d) All

Figure 5.39: **40x40 SOM with SMOTE**: ROC-curves of the three different cluster methods. The asterisks show the points with the highest J-statistic.


(a) Euclidean


(b) Manhattan

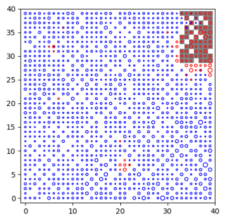

(c) Chebyshev

Figure 5.40: **40x40 SOM with SMOTE**: The optimal clusters according to the J-statistic. The Euclidean method catches all neuron that are part of the cluster according to the U-matrix in Figure 5.37. The Manhattan cluster misses the top right neuron, and the Chebyshev cluster misses many neurons.

5.2 Cross-Validation/Prediction

5.2.1 SOM

The Cross-Validation approach was applied to the map with the map size of 10x10, 20x20, 30x30 and 40x40, and with 1 000 000, 2 000 000, 3 000 000, 4 000 000 and 5 000 000 number of iterations. The results of the MCC-value from these simulations with all combinations of these parameters are shown in Figure 5.41 and Figure 5.42. The simulations were also performed with and without the SMOTE-algorithm with an oversampling of 1000 %.

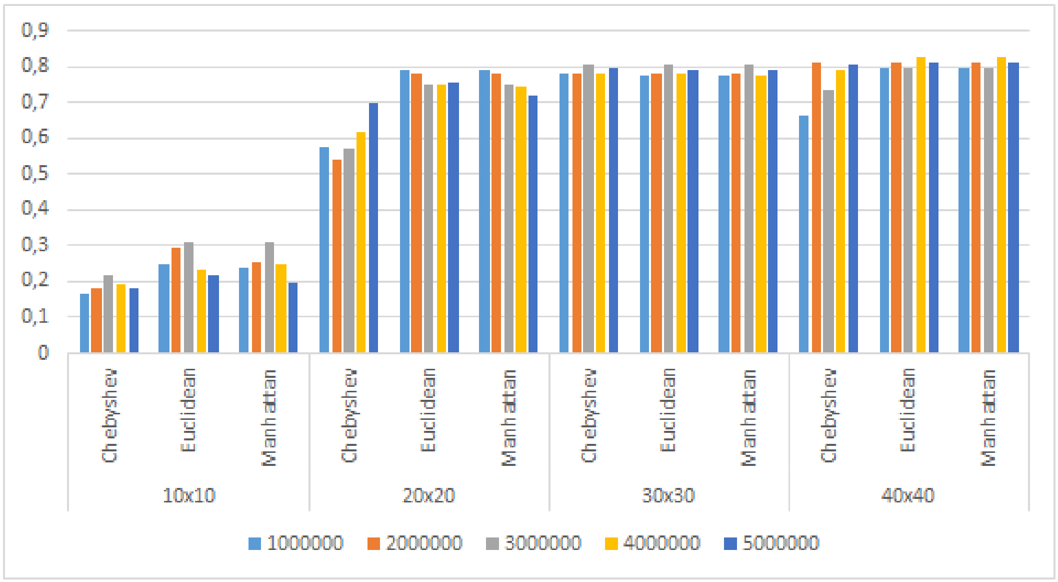


Figure 5.41: Figure showing the MCC-results (vertical axis) for different map sizes, iterations and clustering identifiers without the SMOTE-algorithm.

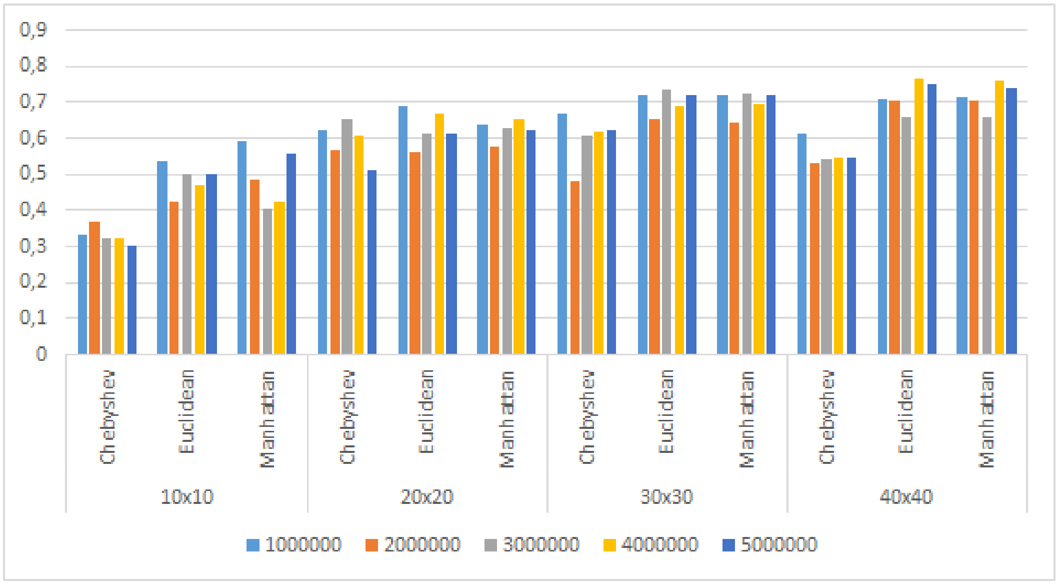


Figure 5.42: Figure showing the MCC-results (vertical axis) for different map sizes, iterations and clustering identifiers with SMOTE of 1 000%.

Some observations drawn from Figure 5.41 and Figure 5.42 are the following:

- Using a map size of 10x10 results in a significant lower MCC-value.
- The number of iterations for the simulation does not seem to be significantly correlated with the MCC-value.
- The MCC-value seem to stagnate after using a map size of 20x20.
- The SMOTE-algorithm seems to have a negative effect on the MCC-value, except for a map size of 10x10.
- With the map size of 20x20, the Chebyshev distance measurement seem to not perform as well as the Euclidean and Manhattan distances.
- It does not seem to be a significant difference by using the Euclidean or Manhattan distance.

The time it took to perform a simulation in order to do a Cross-Validation on the data set varied a lot. For example, when using a map size of 40x40, the simulation lasted for several days, but when using a map size of 10x10 or 20x20, the simulations lasted a few hours. Due to this, and the fact the MCC-value when using a map size of 40x40 do not significantly outperform that of a map size of 20x20, the performance of the SOM was further investigated using a map size of 20x20, and since a million iterations seems to be sufficient to reach a high-performing result, that parameter was also used.

The data showed in Table 5.1 are the results from performing five simulations for every category on the left in the table to get a thorough empirical estimation of the performance measurement with an average and a standard error.

Table 5.1: Average results for Cross-Validated SOM with a map size of 20x20 and 1 000 000 iterations with the Manhattan distance as cluster identifier performed 5 times. *Original Data* stands for an unaltered data set with 9 258 data points. The numbers after *SMOTE* indicates with how many percent that the labeled data is oversampled. The numbers after *Undersampled* indicates how many data points are used in the undersampled data set. Standard error in parenthesis.

	TP	TN	FP	FN	MCC	$\mathbf{P}(P\|PP)$	$\mathbf{P}(N\|PN)$
Original Data	5.75 (0.13)	916.20 (0.62)	3.30 (0.62)	0.55 (0.13)	0.76 (0.03)	0.65 (0.05)	0.99 (0)
SMOTE:							
1000%	5.85 (0.37)	912.30 (0.29)	7.00 (0.29)	0.45 (0.37)	0.65 (0.03)	0.46 (0.02)	0.99 (0)
500%	5.92 (0.08)	912.50 (1.25)	6.96 (1.25)	0.38 (0.08)	0.67 (0.03)	0.49 (0.04)	0.99 (0)
200%	6.05 (0.21)	913.90 (0.79)	5.65 (0.79)	0.25 (0.21)	0.70 (0.01)	0.52 (0.17)	0.99 (0)
Undersampled:							
5000 Data Points	5.82 (0.22)	490.20 (0.19)	3.50 (0.19)	0.48 (0.22)	0.76 (0.02)	0.62 (0.01)	0.99 (0)
2000 Data Points	5.70 (0.39)	191.66 (0.40)	2.04 (0.40)	0.60 (0.39)	0.81 (0.04)	0.74 (0.04)	0.99 (0)

The results from Table 5.1 indicates that when the SOM algorithm is used without applying the SMOTE-algorithm and using the original data set with 9 258 data points and with the undersampled data set with 2 000 data points produce the best outcome based on the MCC and probability measurements. With the mentioned undersampling approach and parameters used to identify labeled trades, the probability that a trade inside the labeled cluster is indeed a high risk trade is around 74%. Another interesting observation from the results is that in all cases, one can be almost 100% certain that a trade outside the labeled cluster is not a risky trade.

5.2.2 K-means

The results from performing the K-means clustering method is shown in Figure 5.43.

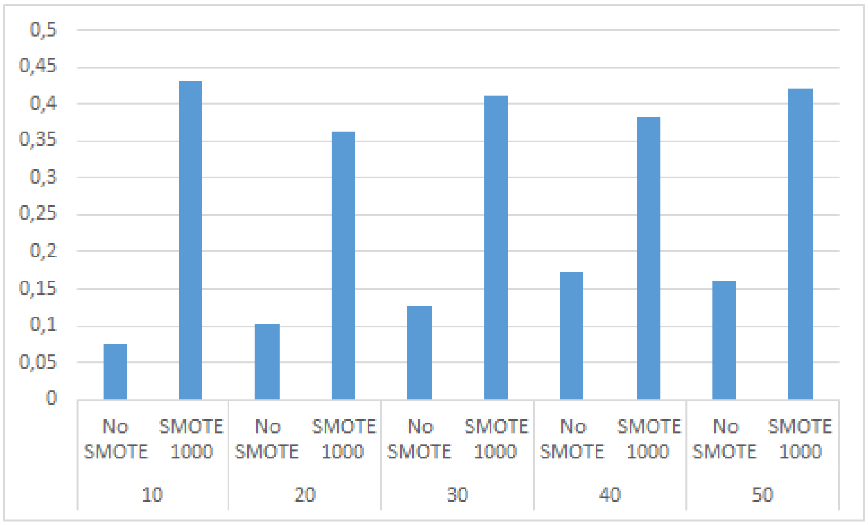


Figure 5.43: Figure showing the MCC-result (vertical axis) with the K-means clustering method

From this figure, it is shown that for the K-means method the MCC-values becomes significantly larger when applying the SMOTE-algorithm on the data set. For the SOM, the MCC reaches a value of around 0.8 without the SMOTE-algorithm, and the result with the SMOTE-algorithm applied for the K-means method reaches up to around 0.4. With this result, the fact that the K-means method produced this result in a few minutes and the fact that the a SOM-algorithm with 1 000 000 iterations and a map size of 20x20 lasts for a few hours, needs to be taken into consideration when comparing these two methods.

Table 5.2: Average results for Cross-Validated K-means with 30 clusters performed 5 times. *Original Data* stands for an unaltered data set with 9 258 data points. The numbers after *SMOTE* indicates with how many percent that the labeled data is oversampled. The numbers after *Undersampled* indicates how many data points are used in the undersampled data set. Standard deviation in parenthesis.

	TP	TN	FP	FN	MCC	$\mathbf{P}(P\|PP)$	$\mathbf{P}(N\|PN)$
Original Data	5.33 (0.22)	728.10 (6.09)	191.40 (6.09)	0.98 (0.22)	0.13 (0.01)	0.03 (0.001)	0.99 (0)
SMOTE:							
1000%	6.03 (0.13)	894.20 (7.01)	25.30 (7.01)	0.28 (0.13)	0.43 (0.05)	0.20 (0.04)	0.99 (0)
500%	5.88 (0.10)	892.25 (3.81)	27.25 (3.81)	0.43 (0.10)	0.40 (0.02)	0.18 (0.02)	0.99 (0)
200%	5.78 (0.15)	892.93 (18.01)	26.58 (18.01)	0.53 (0.15)	0.43 (0.10)	0.22 (0.09)	0.99 (0)
Undersampled:							
5000 Data Points	5.70 (0.27)	431.60 (11.27)	62.08 (11.27)	0.60 (0.27)	0.26 (0.03)	0.09 (0.02)	0.99 (0)
2000 Data Points	6.18 (0.05)	184.90 (1.20)	8.80 (1.20)	0.13 (0.05)	0.62 (0.03)	0.41 (0.03)	0.99 (0)

The results from Table 5.2 indicates that the algorithm has a high performance when it comes to identifying the labeled data points with a high number for True Positive and a low value for False Negatives. The weakness with the K-means however is its capability when it comes to excluding unlabeled data points from the cluster with a low number for True Negatives and high number for False Positives, which reduce the MCC and probability measurements. Nonetheless, when using a version of the K-means method, one is able to say that with approximately a 41% certainty that a trade within the identified cluster is in fact a risky trade that would need further investigation according to these results. Just like the result for when the SOM algorithm is used, the probability of a trade not being in the need of further investigation when it is placed outside of the identified cluster, is more or less with a 100% certainty.

Chapter 6

Discussion

The results from running the numerous SOM simulations both answer and raise questions about the method as a functional way of forecasting and capturing operational risk related to audit change.

6.1 Parameters

Some parameters of the SOM algorithm, such as the initial learning rate and neighborhood size, were not changed through the simulations and were set to values relying on previous work and sources on SOM. The parameters that has been investigated are the number of iterations, the size of the map, different clustering methods, and the use of SMOTE.

6.1.1 Number of Iterations

It seems that the number of iterations that is used to train the map does not have a significant impact on the results, presuming it is at least 1 000 000.

6.1.2 Map Size

The most obvious observation from the visual results is that a 10x10 SOM behaves very differently from the larger SOMs, regardless of whether SMOTE is used or not. The larger SOMs all display clear clusters of the labeled trades, while the 10x10 SOMs do not. This indicates that the size $10x10$ is too small, which is confirmed by the cross-validation that shows that the MCC-values of the $10x10$ SOMs are outperformed by the larger sizes. By looking at the MCC-values from the different simulations with different sizes, the conclusion can be drawn that it should be a clear threshold for the performance of the map somewhere between the map sizes of 10x10 and 20x20. An interesting observation from the MCC-values is that above the size $20x20$, the values stagnate and do not improve significantly for larger sizes.

6.1.3 SMOTE

When using SMOTE the most obvious effect is that the labeled clusters are larger, which in itself is neither positive nor negative. From the visual results it might seem like the use of SMOTE increases the exclusion of unlabeled trades in the labeled clusters, but that is actually disproved by Table 5.1 that shows that the number of False Positives increases when using SMOTE.

Another effect is the formation of a second and even third cluster of labeled trades in the larger SOMs using SMOTE. Since the method used in this report assumes the labeled data is contained in a single cluster, this effect is potentially a big problem, depending on how big the secondary clusters are.

6.2 Cluster Identification

The visual results show some interesting features of the three different methods used to identify the labeled cluster. When not using SMOTE they produce the same or very similar results for the larger SOMs, while they always differ when using SMOTE. That is not a surprise since the use of SMOTE makes the clusters larger. The visual results that are presented do not give any indication of which method is best, but the numbers in Figure 5.41 and Figure 5.42 that are based on more simulations show a better picture. Overall, the Euclidean and Manhattan methods seem to outperform the Chebyshev method, based on the MCC-value. The reason for this might be the fact that the number of neurons in the Chebyshev cluster increases more by each increase in the cluster radius, compared to the other two methods, hence making it less calibrated. An interesting observation is that the strings of dark colors separating different clusters in the U-matrices are often diagonal or close to diagonal, and rarely horizontal or vertical, which is probably caused by the use of the Manhattan distance to compute the neighborhoods. That might be another reason the Chebyshev method is inferior to the others.

For the SOMs using SMOTE, the visual results reveal that none of the methods works perfectly. It seems they often fail to find all neurons that should be in the labeled cluster according to the U-matrix. For size 20x20 only the Chebyshev cluster includes all neurons, for 30x30 only the Manhattan cluster, and for 40x40 only the Euclidean cluster. The reason this happens is that the use of SMOTE makes the labeled cluster larger and less centered. All three methods are based on choosing a center-point and then expanding the cluster from that point, either in the form of a diamond (Manhattan), a circle (Euclidean), or a square (Chebyshev). Thus, they all assume that the labeled trades are centered around one point. When they are not and the labeled cluster instead has a tail, the three methods have problems identifying the whole cluster. This is what happens in Figures 5.30, 5.35 and 5.40.

The cluster identification and the selection of the optimal radius or threshold for the cluster differ in one manner in this thesis. In order to get a satisfactory ROC-curve to base the choice of threshold for the cluster on, all neurons within the threshold are accounted for, even in the cases where neurons only has unlabeled data assigned to them. This is because when calculating the ROC-curve, the total amount of data points is used, so the calculated value may be misleading if those neurons were not accounted for. However, when identifying the labeled cluster used to predict new data points, only the neurons containing at least one labeled data point inside the threshold is included. By observing the figures showing the selected neurons for the cluster, the applied method seem to be able to identify the clear majority of the neurons in the cluster.

The choice of method to determine which neuron should be considered as the center of the cluster could have different implementations. One implementation example is to select the neuron which has the most labeled trades assigned to it as the center of the cluster, since it is observed that the neuron with the most labeled trades often is a part of the desired cluster and therefore base the threshold around that neuron. A problem arises when that neuron is located in the outskirts of the visual cluster, see for example the largest neuron in Figure 5.26. In those cases the threshold would early on include a lot of neurons without any labeled trades and may not capture as many neurons that visually should be included in the cluster. By using the weighted average located neuron and using that as the center of the cluster, the threshold has the ability to capture the entire cluster in a better way when the neuron with the most labeled trades is not located in the center of the cluster.

One issue with the approach of all three cluster methods is that they are all based on the assumption that the SOM algorithm clusters the labeled data in a single cluster. As mentioned in Section 6.1.3 that is not always the case when using SMOTE.

The clustering methods used in the report are implemented the way they are in order to automate the process, but to handle the issues discussed above a more manual method would probably improve the result. One method would be to visually inspect the result of the training and then choosing the labeled cluster/clusters by hand.

6.3 Data Set

The provided data set used to capture underlying indicators for operational risk in the form of audit changes along with the selection of attributes leaves a number of issues.

One issue is that of the sample of trades that has been labeled as high risk consists of 63 trades constructed by SHB and the number of unlabeled trades are 9 195. To get a general prediction of the operational risk, there might be a need for a more balanced data set. The efforts to deal with this issue are to undersample the unlabeled trades and to oversample the labeled trades with the SMOTE-algorithm.

Another issue is the selection of features that are included in the data set. The features are chosen based on which intuitive factors can have an impact on the human error and possibly lead to a risky audit change, and a more thorough investigation about the impact of each attribute might help the performance of the method.

The data set is also implemented with the help of One-hot encoding, or dummy variable, which can lead to some problems for the efficiency of the method in the future. If a trade introduces a new category to the data set, for example a trader that was not included in the original data set, then a new data set must be constructed to match the dimension of the new data point and a new map must be trained. This fact may not be an issue but further work probably has to be done in order for the method to be automated.

The results for this thesis are based on the initial categorization of the trades in the data set. One category is the 63 trades constructed by SHB that are labeled as bad, and all of the other trades in the data set fall into the other unlabeled category. The first category is mimicking high risk and the other category has in this thesis been assumed to be of low risk. That may cause a problem with the performance measurements since it is reasonable to believe that the level of risk varies among the unlabeled trades, and a data point identified as a False Positive may in fact be of relatively high risk, and therefore not completely incorrectly labeled.

6.4 Performance and Efficiency

The results generated by the use of SOM show some promise for the method to be used as a way to predict when an audit change is exposed to critical risks. According to the findings, the probability of capturing a risky trade when it is inside the identified cluster is roughly 74% with the use of a map with a map size of 20x20 and has been train through 1 000 000 iterations with an undersampled data set. Furthermore, when a trade is outside the cluster, that trade is with almost a 100% certainty not a risky trade, which is also a promising result. From looking at the results in Table 5.1, is seems that the best method is to use a data set which has not been altered by the SMOTE-algorithm. The weakness of the map with SMOTE is that the cluster grows too large for it to not exclude enough unlabeled trades, which also is observed in the figures depicting maps trained with a data set altered with SMOTE in Section 5.1. To capture all of the labeled trades however, the use of SMOTE shows some real strengths according the True Positive values in Table 5.1.

In this thesis, a decision about which criteria is the most important between the accuracy of the method and its efficiency, i.e. how fast the method is, and a weighing between the two. As previously mentioned, the MCC-value does not seem to significantly increase after training a map with a map size of 20x20 and 1 000 000 iterations, and thus not compromising the accuracy for efficiency in a severe way when going from a map size of 40x40 to 20x20. However, the choice has to be made for which one of the two criteria is of highest priority when deciding between using a SOM or the K-means method for the task of forecasting operational risk. Compared to the SOM method, the time spent to train the data set is minimal with the K-means method, but loses in accuracy even though it still generates some relatively good values in that aspect as well. The performance of the K-means method seems to improve with the SMOTE-algorithm, though the best results is gained through undersampling according to Table 5.2. If the time aspect was not an issue, the use of a map with the map size of 40x40 and a more thorough investigation with that parameter would of course be of interest, and would probably produce slightly improved results according to Figure 5.41 and Figure 5.42.

From Table 5.1, the results also indicates that the best method to obtain a high MCC-value is to undersample the original data set where the use of 2 000 data points generates a map with an MCC-value of 0.81 and the probability of correctly identifying a risky of 0.74%. Another observation is that the results does not seems to improve when going from the original data set with 9 258 data points to one with 5 000 data points. The problem to use a smaller data set is the loss of information from the removed data points which could lower the accuracy of the categorization in reality. This also means that there does not seem to be an incentive to use a data set with 5 000 data points instead of the original set, but a weighing between the better results and the loss of information that comes with the data set of 2 000 data points and the original has to be done when choosing which one to use for this purpose.

The performance measurement of the Bayesian probability is a way of trying to quantify the performance of the map in an understandable way, where as the MCC-value is more used to rank the different simulations between themselves. As previously mentioned, there is a problem with that the data set only contains 63 labeled data points, and to get a more accurate and true value of the Bayesian probability measurement when using the implemented SOM, an extended data set with more labeled trades could help.

Chapter 7

Conclusion

The overall outcome of this thesis shows promising results for the possibility of using SOMs to identify and forecast operational risk in the form of audit changes. With the use of a map size of 20x20 and training through 1 000 000 iterations, the method was able to correctly identify labeled data points with a certainty of 74%, and unlabeled data points with a certainty of almost 100%. Furthermore, it is observed in Figures 5.41 and 5.42 that the use of a map size of 10x10 is clearly inferior to larger maps, and that after the size of 20x20, the results stagnate. The use of different numbers of iterations do not seem to have a large impact on the results, as long as the iterations are at least 1 000 000.

When considering the visual figures in Chapter 5, the performance of the three cluster-identifying methods varies, but overall the Manhattan and Euclidean methods seems to work more efficiently than the Chebyshev method, which is also supported by some of the results in Table 5.41 and 5.42. The use of the ROC-curve and J-statistic also seem to be a good way, though not a perfect way, to determine the optimal thresholds for the clusters when observing the ROC-curve figures in Chapter 5 and the figures showing the selected neurons for the clusters.

With the exception of when using a map size of 10x10, the results was better without using SMOTE than with it. The main reason for this seems to be due to the number of False Positive data points which can observed in Table 5.1. However, an undersampling of the original data set produced the best results, but the cost of loosing information needs to be taken into consideration for that result.

The SOM method also proved to be superior to the K-means method on the same data set. Both methods seems to capture the underlying similarities of the trades in the data set but the way that the clusters are organized and identified with the SOM outperform K-means in the results. So, if accuracy is considered to out-weigh the speed of the K-means, SOM is to prefer.

7.1 Further Work

Since the methods used to identify the labeled clusters in the trained SOMs proved to have some flaws, further work on the subject would probably benefit from investigating other methods. Probably a clustering method based on the U-matrix would yield better results.

To optimize the method, the design parameters, e.g. the learning rate, the initial neighborhood size and the methods by which these are updated, which were kept unchanged throughout the project would have to be investigated. In particular, the shape of the SOM. This thesis focused on a SOM where a rectangular grid was used to map the data points. Using a hexagonal grid is also common in many applications of SOMs, and would be interesting to explore.

A big part of Machine Learning is data selection, which includes handling of missing data, feature selection, normalization etc. It does not matter how good a Machine Learning algorithm is if the data used has not been preprocessed in a satisfactory way. This was not a big part of this thesis, but would be needed in further work.

Bibliography

[1] S. Pongor A. Kocsor and P. Sonego. Roc analysis: applications to the classification of biological sequences and 3d structures. *Briefings in Bioinformatics*, 9:198–209, 2008.

[2] S. Fang A. Konate, H. Pan and S. Asim. Capability of self-organizing map neural network in geophysical log data classification. *Journal of Applied Geophysics*, 118:37–46, 2015.

[3] Svenska Handelsbanken AB. Annual report, 2017.

[4] Svenska Handelsbanken AB. Risk and capital management - information according to pillar 3, 2017.

[5] J. Bainbridge and S. Furber. Chain: a delat-insensitive chip area interconnect. *IEEE Micro*, 22:16–23, 2002.

[6] Z. Bodur. Operational risk and operational risk related banking scandals. *Maliye Finans Yazilari*, 97:64–84, 2012.

[7] D. Chicco. Ten quick tips for machine learning in computational biology. *BioData mining*, 10(1):35, 2017.

[8] R. Coghetto. Chebyshev distance. *Formalized Mathematics*, 24(2):121–141, 2016.

[9] Basel II Committee. Operational risk transfer across financial sectors, 2003.

[10] A. Hussain. 1997 operational risk management survey. *Managing Operational Risk in Financial Markets*, pages 70–71, 2000.

[11] A.K. Jain. Data clustering: 50 years beyond k means. *Pattern Recognition Letter*, 31:651–666, 2010.

[12] M.T.V. Martinez-Palacios J.F. Martinez-Sanchez and F. Venegas-Martinez. An analysis on operational risk in international banking: A bayesian approach. *Estudios Gerenciales*, 32:208–220, 2016.

[13] L. Hall K. Bowyer, N. Chawla and P. Kegelmeyer. Smote: Synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16:321–357, 2002.

[14] M. Kendall, A. Stuart, J.K. Ord, and A. O'Hagan. Kendall's advanced theory of statistics, volume 1: Distribution theory. *Arnold, sixth edition edition*, 1994.

[15] T. Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78:1464–1480, 1990.

[16] P. Embrechts M. Degen and D.D. Lambrigger. The quantitative modeling of operational risk: between g-and-h and evt. *ASTIN Bulletin: The Journal of the IAA*, 37:265–291, 2007.

[17] B.W. Matthews. Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA)-Protein Structure*, 405(2):442–451, 1975.

[18] J.S. Jordan P. De Fontnouvelle, D. Jesus-Rueff and E.S. Rosengren. Using loss data to quantify operational risk. *SSRN Electronic Journal*, pages 1–10, 2003.

[19] L. Rafferty P. Hung and W. Rafferty. Introduction to big data. *Big Data Applications and Use Cases*, 1:1–15, 2016.

[20] T. Pettersson. Investigating the Potential of Using SOM on Audit Changed Trades. Master's thesis, KTH Royal Institute of Technology, Sweden, 2017.

[21] D.M. Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. 2011.

[22] M. Cruz R. Coleman and G. Salkin. Modeling and measuring operational risk. *Journal of Risk*, 1:63–72, 1998.

[23] I. El Naqa R. Li and M.J. Murphy. What is machine learning? *Machine Learning in Radiation Oncology*, pages 3–11, 2015.

[24] E.F. Schisterman, N.J. Perkins, A. Liu, and H. Bondell. Optimal cut-point and its corresponding youden index to discriminate individuals using pooled blood samples. *Epidemiology*, 16(1):73–81, 2005.

[25] P. Stefanovič and O. Kurasova. Influence of learning rates and neighboring functions on self-organizing maps. In *International Workshop on Self-Organizing Maps*, pages 141–150. Springer, 2011.

[26] A. Ultsch. Self-organizing feature maps for exploratory data analysis. In *Proc. of the International Neural Network Conference (INNC), 1990*, 1990.