Dissertation

Advanced data exploration methods based on Self-Organizing Maps

ausgeführt zum Zwecke der Erlangung des akademischen Grades eines
Doktors der technischen Wissenschaften

unter der Leitung von

ao. Univ.Prof. Dipl.-Ing. Dr.techn. Andreas Rauber
E188 – Institut für Softwaretechnik und Interaktive Systeme

eingereicht an der Technischen Universität Wien
Fakultät für Informatik

von

Georg Pölzlbauer
9725498
Servitengasse 19/5
1090 Wien

Wien, am 21. 10. 2008

Zusammenfassung

Self-Organizing Maps (SOMs) sind ein wichtiges Data Mining Verfahren um Informationen aus großen Datenmengen herauszufiltern. In dieser Arbeit werden drei auf SOMs aufbauende Methoden vorgestellt, die beim Verständnis solcher großer Datenmengen helfen sollen. Zwei dieser Methoden sind Visualisierungsverfahren für SOMs, die dritte ist eine vom SOM Trainingsalgorithmus inspirierte Klassifizierungsmethode für Zweiklassenprobleme.

Die erste der vorgestellten Methoden zeigt den Zusammenhang zwischen dem Datenset, auf dem eine SOM trainiert worden ist, und den Codebookvektoren, aus denen diese SOM besteht. Ausgehend von einem Graphen, der den gegenseitigen Abstand zwischen Datenvektoren darstellt, werden Linien auf einer SOM Visualisierung gezeichnet. Dies zeigt die Dichte einzelner Bereiche der Karte, durch den projektionsbedingten Dimensionsverlust entstandene Topologieverletzungen und die Positionen von Ausreißern.

Die zweite Methode ist ein Visualisierungsverfahren, das die Clusterstruktur einer SOM in verschiedenen Detailliertheitsgraden zeigt. Ein Parameter dient zur Adjustierung der gewünschten Granularität der dargestellten Information. Zur Darstellung der Ergebnisse wird eine Vektorfeldrepräsentation gewählt und eine Metapher für Spezialisten mit ingenieurswissenschaftlichem Hintergrund erzeugt. Diese Methode wird dahingehend erweitert, Gruppen von Variablen gegenüberstellen zu können und somit den Einfluss einzelner Dimensionen auf die Clusterstruktur festzustellen.

Die dritte Methode ist ein Machine Learning Verfahren für binäre Klassifikationsprobleme. Es besteht aus einem Ensemble linearer Klassifikatoren, die jeweils einen Bereich des Eingaberaums abdecken. Der Trainingsalgorithmus, der diese lokalen Klassifikatoren platziert, ist vom SOM Algorithmus abgeleitet. Er baut auf dem von SOMs bekannten Prinzip auf, dass in einer vordefinierten Topologiestruktur benachbarte Einheiten einander beeinflussen.

In dieser Dissertation wird der theoretische Hintergrund dieser Methoden beschrieben. Empirische Evaluierungen werden anhand einer Reihe künstlicher Datensets sowie Benchmark- und Real-World-Datensets durchgeführt. Weiters wird der Nutzen der Methoden aufzeigt, sowie deren Stärken und Schwächen analysiert. Besonderer Wert ist auf die Erstellung aussagekräftiger, die spezifischen Eigenschaften überwachter und unüberwachter Lernverfahren adressierender Datensets gelegt worden.

Abstract

Self-Organizing Maps are an important data mining method for extracting information from a data set. In this thesis, three techniques that are based on SOMs are introduced for helping to understand large amounts of data. Two of them are visualization techniques for SOMs, while the third is a classification method for two-class problems inspired by the SOM training algorithm.

The first of the proposed methods is based on putting the data set a SOM has been trained with in relation with the codebook vectors that define this SOM. Starting from a graph that reflects the mutual distance between data vectors, a set of lines is plotted on top of the output space visualization of the SOM. This shows the density of the areas of the map, violations of the topology due to the projection-induced dimensionality loss, and the location of outliers.

The second contribution is a visualization technique that shows the clustering structure of a SOM on various levels of detail. A parameter is provided to adjust the desired granularity of information that is to be shown. For displaying the results, a vector field representation has been chosen in order to provide a metaphor that appeals to specialists with engineering backgrounds. This method is extended to a setting that contrasts groups of contributing variables in order to single out their influence on the clustering structure.

The third contribution is a machine learning method for binary classification problems. This technique consists of an ensemble of linear classifiers that each cover a portion of the input space. The training algorithm that actually places these local classifiers is influenced by the SOM algorithm. It exploits the SOM principle of aligning nearby units according to a super-imposed topology structure.

The theoretical background for these methods is described in this thesis. Empirical evaluation on a series on artificial, benchmark, and real-world data sets show their applicability, and their strengths and weaknesses are discussed. Much effort has been dedicated at designing meaningful artificial data sets that address specific abilities of supervised and unsupervised learning methods.

To my parents.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background and motivation

Exploring, understanding, and learning from large amounts of data has become increasingly important across all scientific disciplines. With ever growing amounts of data that are collected during empiric experiments and the possibilities to connect existing databases, the need for methods to handle this data is evident.

This thesis consists of several contributions to various domains of data mining, which are all based on or inspired by Kohonen's Self-Organizing Map. SOMs have been used extensively in unsupervised learning and data visualization. They are unique insofar as they combine vector quantization and projection, as well as having a two-way assignment between the feature and the output spaces. This enables a variety of post-processing methods that aim at visualizing results or further digging into what lies concealed somewhere within the data set.

The examples for illustrating the methods come from highly different data sources. Data sets are collected from research in medicine, marketing, demography, biology, and various other domains. The methods described in this thesis are therefor indifferent to the source of the data, decoupled from the original semantic meaning.

The contribution of this thesis lies in proposing three methods that are centered around data mining with SOMs. The first one, the Graph visualization method, is a visualization technique for providing deeper insights into the properties of the data that is being investigated, and further highlights several specific defects that inevitably occur during the creation of a SOM. The second, the Gradient Fields method, is another visualization method that displays results suited for professionals with engineering rather than computing backgrounds. It shows vector fields in an analogy to scientific flow visualizations. The third contribution are Decision Manifolds, which transfer the unsupervised Self-Organizing Maps methodology

to a supervised classification setting. This method allows for an explicit representation of the decision boundary, which is often only encoded implicitly in a trained classifier.

This PhD thesis is based on several publications that have been written during more than three years of research at the Vienna University of Technology. The research has been conducted in cooperation with and financed by an industry partner working in the domain of petroleum engineering. This partner uses SOMs for optimizing the field engineering process, i.e. the maintainance and failure-free operation of oil fields while maximizing the output and minimizing costs. One of the requirements has therefor been the presentation of SOM results for engineers with less focus on computing and visualization.

1.2 Organization and contributions

This thesis is organized in accordance to the timeline of the research and the resulting publications conducted.

Chapter 2 describes related work. The general data mining setting is explained, with introductions to supervised and unsupervised learning. Further, the SOM algorithm and its properties of trained maps are explained in detail, along with the notations that are used in the later chapters. Particular attention is devoted to visualization methods for Self-Organizing Maps. This sections includes findings from a publication that surveys quality measures for SOMs [67].

Chapter 3 introduces the first of the two visualization methods that have been developed for SOMs. This technique is suited for discovering topology violations and for learning about the density of the data that generates the SOM. Work related to this subject has been published at two conferences [75, 74].

Chapter 4 describes the Gradient Field and Borderline visualization techniques, which are based on a vector field representation of displaying information. It is used to show the clustering structure at various levels of detail. Considerable research efforts have been dedicated to this subject, which has resulted in a series of publications at conferences and a journal publication [77, 68, 76, 70, 69, 71].

Chapter 5 describes the final contribution of this thesis. It is a neural classifier algorithm for binary problems. The algorithm itself is based on Self-Organizing Maps. A conference paper [72] and a journal paper [73] have been published relating to this subject.

Chapter 6 summarizes this thesis.

The appendix consists of a variety of additional material that does not fit into the chapters forming the main part of this work without detracting the reader from specific details. In Appendix B, the abbreviations used throughout this work are listed. Appendix C provides definitions of formulas that have been moved to the

appendix as they are quite lenghty and would otherwise clutter the main parts of this thesis. Appendix D provides an extensive description of the data sets used in this work. It lists the sources and characteristics of benchmark supervised and unsupervised data sets, as well as the construction of the artificial data sets. All of the artificial data sets have been designed to show a specific feature or to highlight a particular problem of supervised or unsupervised learning algorithms, and considerable efforts have been dedicated to creating these data sets.

1.3 Notational conventions

This section covers the general style of notational and mathematical expressions used in this thesis. Generally, mostly matrix operations and some concepts from probability theory are used. Matrices are denoted as uppercase boldface letters, e.g. $\mathbf{X}$. Vectors are denoted as lowercase boldface letters, e.g. $\mathbf{v}$. When referring to the i-th row vector of a matrix, the lowercase letter of the matrix letter is used, and a subscript is written to denote the row index, respectively. For example, matrix $\mathbf{X}$'s i-th row vector would be written as $\mathbf{x}_i$. Column vectors are used mainly for component planes and are written with indices in brackets, e.g. $\mathbf{x}_{(j)}$ for the j-th column of matrix $\mathbf{X}$. The notation for scalar values depends on the intended use and context: Parameters are usually written as lowercase Greek symbols, e.g. γ, constants and variables as lowercase italic letters, e.g. c, and calculation results as uppercase italic letters, e.g. E. When referring to a single element of matrix $\mathbf{X}$ in row i and column j, it is written as a lowercase italic letter with subscripts, e.g. x_{ij} or $x_{i,j}$. These are only rough guidelines for formulas introduced in this thesis and some notations may differ, especially if symbols are widely used in other literature, e.g. eigenvectors are denoted as λ despite not being parameters, or the covariance matrix as Σ etc. Appendix A provides a tabular summary of the notational conventions for reference purposes.

Chapter 2

Self-Organizing Maps and related work

In order to lay the foundation for the later chapters, this chapter outlines the general data mining setting. Further, related techniques are presented and details of the Self-Organizing Maps training algorithm and properties of the SOM are provided. Section 2.1 provides an introduction to Data Mining, and introduces some of the methods and concepts that will be referred to in the later sections. Section 2.2 describes the Self-Organizing Map algorithm. Section 2.3 contains an overview of several SOM visualization techniques and SOM quality measures that are used throughout this thesis.

2.1 Introduction to data mining methods

Data Mining (or Knowledge Discovery in Databases) describes the broad concept of finding "interesting" patterns in large collections of data. There are various definitions what the term "Data Mining" actually refers to. Frawley and Piatetsky-Shapiro [23] define it as "the nontrivial extraction of implicit, previously unknown, and potentially useful information from data". Data Mining thus describes the conceptual goals of what needs to be done, and relies upon a wide range of different techniques to achieve them, such as artificial neural networks, cluster analysis, regression and other statistical methods, and information theoretic approaches. This section serves as an introduction to the various approaches and provides definitions to categorize them according to the domain where they can be used.

2.1.1 Preprocessing, features, and distance

In typical Data Mining problems, the data collection and preprocessing phase is usually the first step and has to be addressed with considerable attention [78]. While the task of translating real-world observations into machine-readable form is usually attributed to the domain of information retrieval [106], this thesis is concerned only with data sets that can be represented in a table-like structure where the rows correspond to individual observations (also samples or patterns), and columns refer to variables that have been measured in some way for each observation.

Before the data is used for Data Mining, it is usually transformed and scaled. As many of the methods that will be discussed in this thesis are not able to deal with non-real variables, a transformation has to be applied to convert categorical or ordinal values to an interval scale, usually by 1-to-N coding, where a variable with N distinct values is split up into N variables. Further, as many algorithms work by measuring the distance between points in feature space, the variables have to be normalized to avoid difficulties that stem from variables measured in different units. In this thesis, normalization by subtracting the variable mean and dividing by the standard deviation is performed, unless otherwise noted. As a result ot these preprocessing steps, the data set can now be written as a matrix $X \in \mathbb{R}^{N \times D}$, where N is the number of patterns and D is the number of variables. The i^{th} sample is denoted as vector $\mathbf{x}_i \in \mathbb{R}$.

Most of the methods discussed in this thesis rely on measuring distances between points in this feature space. Distance is of crucial importance especially for clustering, as points with low distance, i.e. with a high degree of proximity in many of the variables measured, are likely to refer to similar real-world objects. Unless otherwise noted, Euclidean Distance will be used:

$$d_F(\mathbf{x}_i, \mathbf{x}_j) = \|\mathbf{x}_i - \mathbf{x}_j\| = \sqrt{\sum_{k=1}^{D} (x_{ik} - x_{jk})^2}. \tag{2.1}$$

Euclidean Distance is defined by the lenght of the straight line connecting points $\mathbf{x}_i$ and $\mathbf{x}_j$. This measure may not be suitable for each and every problem, especially not for very high-dimensional ones. For a discussion of this problem and general limitations of the Euclidean Distance, see for example [51, 2, 108, 20, 1].

2.1.2 Supervised and unsupervised learning

A fundamental distinction between learning methods is whether the data set is labeled or not. The algorithms that deal with either situation are refered to as su-

pervised and unsupervised learning, respectively. Labeled data samples contain a distinguished feature that has to be predicted, for example in a patient database with features describing blood pressure, sex, whether the patient is a smoker etc., the predicted, or output, variable could be whether the patient has lung cancer or not. If the output variable is categorical, this supervised learning setting is referred to as Machine Learning or classification; in the case of a continuous output variable, this is called regression. Within unsupervised learning a core task is clustering, i.e. the process of identifying groups of samples based on their mutual similarity. Clustering is primarily concerned with the definition of data density, and what makes samples similar or dissimilar. Typical problems in clustering algorithms include handling outliers and missing values.

Supervised methods include methods based on or related to Artificial Neural Networks, Information Theory, linear discriminants, prototype vectors, kernel methods, Bayesian statistics and lazy learning. Examples of Artifical Neural Network classifiers [9] are Perceptrons [84], backpropagation multi-layer neural networks [121, 86], and Radial Basis Function Networks [60], which all work by presenting the input data repeatedly to a layered network of simplified neural units that adapt to the signal presented. The non-linear variants of Artificial Neural Networks have been shown to be applicable to every class of learning problem [36].

Methods that are based on Shannon's Information Theory [90] rely heavily on the concepts of Entropy and Information Gain. The most well-known example in Machine Learning are Decision Trees [14] and its numerous variants [80, 105, 79], which recursively split the feature space into rectangular parts. Another more recent and very effective approach are Random Forests [13], which are committees of Decision Trees that have been trained in a certain way that introduces randomness to the otherwise deterministic process.

Methods based on prototype vectors rely on a set of points in feature space that cover a portion of this space and represent the training data points that lie in these regions. The training process consists of moving these prototype vectors to positions that minimize the classification error. The most prominent prototype based methods are the various versions of Learning Vector Quantization [47], where each prototype vector is assigned a class label, and is moved towards the samples of the same class.

Support Vector Machines [107, 15] are powerful classification algorithms that consist of two parts: A kernel function that performs a projection of the original data in a much higher-dimensional data space, and an optimization formulation of fitting a separating hyperplane into the data set. The major advantage of this method lies in the combination of these two parts which allows a very efficient implementation that avoids the complexity problems of other kernel based methods, also known as the "kernel trick". The type of kernel used determines the classes of problems that may be solved, and typical choices are linear, polynomial, and

Table 2.1: A taxonomy of supervised learning methods

Method	Stochastic	lazy	IT-based	linear	binary
Perceptron	yes	no	no	yes	yes
Multi-layer Perceptron	yes	no	no	no	yes
RBF Neural Network	yes	no	no	no	yes
Decision Trees	no	no	yes	no	no
Random Forests	yes	no	yes	no	no
LVQ	yes	no	no	no	no
Linear SVM	no	no	no	yes	yes
Polynomial SVM	no	no	no	no	yes
RBF SVM	no	no	no	no	yes
LDA	no	no	no	yes	yes
Naive Bayes	no	no	no	no	no
k-NN	no	yes	no	no	no

radial basis function kernels.

Many Machine Learning techniques are primarily inspired by statistics, like Linear Discriminant Analysis or Naive Bayes [56]. Linear Discriminant Analysis results in a separating hyperplane that is computed by solving an eigenvector problem, while Naive Bayes relies on Bayesian statistics and computes the posterior probabilities of the samples to be classified, under the assumption that the feature dimensions are independent.

Lazy learning refers to a paradigm in Machine Learning rather than a class of algorithms. It is opposed to the previously discussed algorithms that create models from the training data before actually classifying any unlabeled data. Lazy learners store the input data set and do not perform any learning task during the training step. The actual work is performed during classification and has to be repeated each time a sample has to be labeled. The most prominent lazy learning algorithm is k-Nearest Neighbors, which classifies samples by assigning the majority class of the k training set samples which are closest to the sample that has to be labeled.

The learning algorithms that have been introduced in the previous paragraphs are summarized in Table 2.1 in an attempt to classify them according to several key characteristics. The column "stochastic" refers to algorithms that include some sort of randomness in their computation, as opposed to deterministic ones which always come to the same results given a training data set; "lazy" refers to lazy learning algorithms;"IT-based" refers to techniques that rely on Information Theory; "linear" denotes methods that produce separating hyperplanes; "binary"

refers to whether the method is restricted to two-class output variables. Apart from the training algorithms themselves, major issues in Machine Learning include the measurement of the classification accuracy, model selection for tuning of parameters and improvement of classifier performance, and overtraining. The applications of supervised learning lie in virtually every domain that deals with collection and prediction of data in any respect. Practical Machine Learning problems are often concerned with either selecting the most appropriate algorithm or by tailoring existing methods to the particular problem. A vast number of surveys and books have been dedicated to Machine Learning that describe these issues and other supervised learning algorithm in greater detail [59, 34].

Among the most common unsupervised learning methods are k-means [33], which will be discussed in more detail in the next section, and hierarchical clustering, like single, average, and complete linkage, and Ward's clustering [118]. For comprehensive surveys on the different unsupervised methods refer to [7, 39, 43]. One of the difficulties in clustering lies in evaluating the quality of the resulting partitions, or to determine which of two given sets of clusterings of the same data set is preferable [32]. Other than in the case of supervised learning, there is no clearly defined criterion like classification error that has to be minimized. There are several approaches to assess the quality of clustering. The DB-Index [18], which is discussed in 2.3.4, provides a score that is better for dense, well-separated spherical clusters. The Quantization Error, which is discussed in the next section, emphasizes the overall distance from each sample to its cluster centroid. As the definition of a "good" clustering solution is itself difficult, the approaches that the different clustering techniques take are highly different.

The term "vector quantization" is sometimes used synonymously with unsupervised learning or clustering, but actually refers only to those methods that produce a codebook, i.e. are prototype based. Several clustering methods, such as hierarchical clustering, do not qualify as vector quantization, since only the given training data is clustered without the possibility to assign vectors that are not available during training to any cluster. The goal of vector quantization is to minimize the quantization error, it thus has a clearly defined objective. Vector quantization is used mostly in the domains of signal processing and data compression, where the task is to replace the original high-dimensional data vectors with a low-dimensional representation and some loss of information is allowed. As the number of prototype vector relative to the number of data samples increases, this becomes less of a problem.

2.1.3 k-means and LBG

One of the most prominent clustering algorithms is k-means. It is prototype based, i.e. it contains k codebook vectors of the same dimensionality as the data set

which are used to represent the clusters. The codebook vectors are denoted as $\mathbf{m}_j \in \mathbb{R}^D$ and are initialized randomly. In the sequential training algorithm, as described in [33], the data samples $\mathbf{x}_i$ are presented in random order to the codebook and one prototype vector becomes more similar to the currently selected sample in each step. The LBG method [55] is a batch version of this algorithm that has been shown to be equivalent in terms of convergence to the sequential version. It is computed in two steps, first assigning each data point to its closest prototype vector:

$$I(\mathbf{x}_i) = \arg \min_{j \in \{1,\dots,k\}} \|\mathbf{x}_i - \mathbf{m}_j\|, \tag{2.2}$$

where $I(\cdot)$ represents the index of the prototype vector the data point $\mathbf{x}$ is assigned to[1]. Function $I(\cdot)$ together with the codebook defines a Voronoi tessellation over the feature space. The set of indices of samples mapped to prototype $\mathbf{m}_j$ is denoted as $\mathfrak{S}_j$:

$$\mathfrak{S}_j = \{i | \mathbf{x}_i \in X \wedge I(\mathbf{x}_i) = j\}. \tag{2.3}$$

Further, the subset of samples belonging to prototype vector j is denoted as $X_{\mathfrak{S}_j}$. In the second step of the LBG algorithm, the centroids $\mathbf{n}_j$ are computed as the arithmetic means of the samples assigned to each partition:

$$\mathbf{n}_j = \frac{1}{|\mathfrak{S}_j|} \sum_{\mathbf{x}_i \in X_{\mathfrak{S}_j}} \mathbf{x}_i, \tag{2.4}$$

where $|\cdot|$ denotes the cardinality of a set, i.e. in this case the number of data points assigned to a prototype vector, and $\mathbf{n}_j$ is the center of the data points assigned to node j, which becomes the new codebook vector $\mathbf{m}_j := \mathbf{n}_j$. The process of updating each codebook vector by presenting all the data vectors once is called an epoch. As the prototype vectors move in feature space after each epoch, the assignments $I(\cdot)$ also change. Training is finished after a predefined number of epochs T have been performed, or a stopping criterion is satisfied, such as minimization of an energy function, or when the codebook has converged and subsequent epochs do not result in a change of positions anymore. Once training has finished, the codebook vectors are sufficient to represent the clusters, as all points in feature space can be assigned to one of the clusters according to Equation 2.2.

The quantization error measure is usually associated with vector quantization algorithms. It is computed as the average distance between each sample and its closest codebook vector:

$$E^{\mathrm{Q}} = \frac{1}{N} \sum_{i=1}^{N} \|\mathbf{x}_i - \mathbf{m}_{I(\mathbf{x}_i)}\|. \tag{2.5}$$

[1] In later chapters, the k-th best matching unit will be required, which is written as $I^{(k)}(\mathbf{x}_i)$. Using this notation, the best matching unit would be denoted as $I^{(1)}(\mathbf{x}_i)$, but in this case, the superscript is omitted. For a formal definition of $I^{(k)}(\mathbf{x}_i)$, refer to Section C.1.

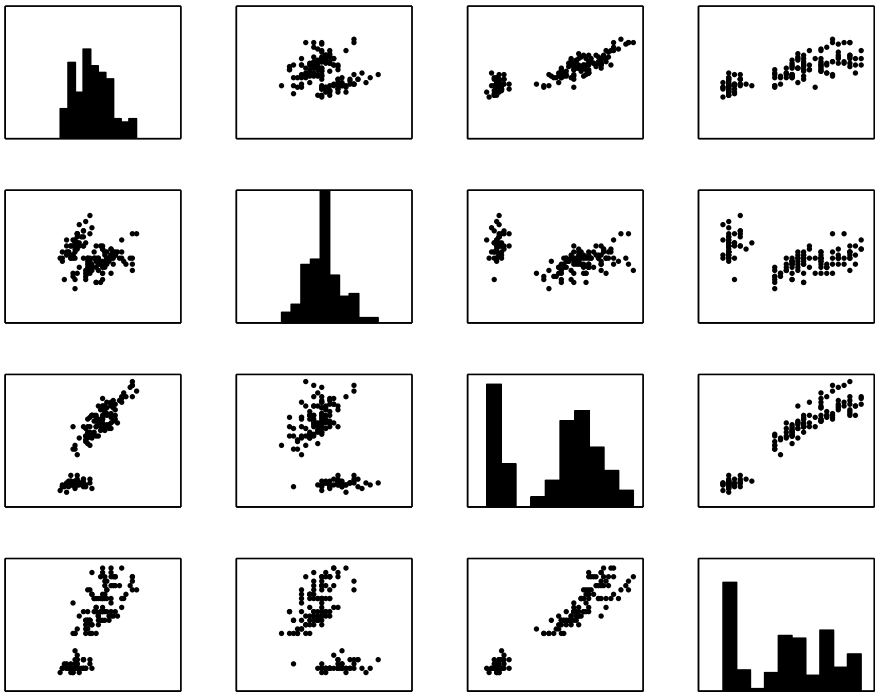


Figure 2.1: Scatterplot matrix of the Iris data set

The quantization error can be lowered simply by increasing the number k of prototype vectors, in which case the size of the codebook increases. The tradeoff between codebook size and quantization error has to be determined in the context of the application domain. For data compression, the total amount of data to be transmitted is the main consideration, but for clustering tasks, the clusters may be analysed by humans who want to understand the output from such an algorithm and therefore want the number of clusters to be low.

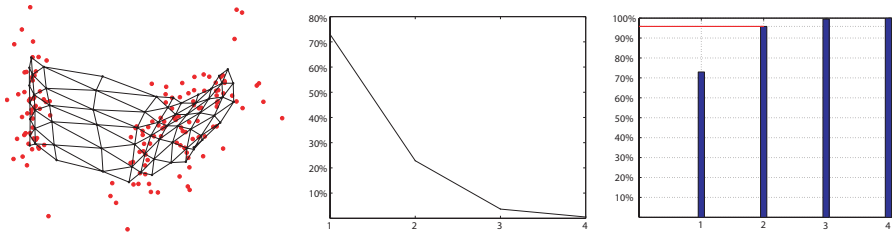
The LBG algorithm is deterministic except for the initial positions of the prototype vectors. The results vary significantly for different initializations and various strategies have been proposed to deal with this problem [65, 12], e.g. by placing the prototype vectors far away from each other.

2.1.4 Vector Projection

An important field of data analysis is visualization, as quantitative data may be effectively communicated through graphical means [100]. As the kind of data that is discussed in this thesis is high-dimensional, visulization is not a straightforward task. One of the most common approaches are scatterplots, where an orthogonal projection onto the plane spanned by two variable axes is shown. By arranging the pairwise scatterplots in a way that each row and column in a matrix is assigned a dimension, and a scatterplot is shown between the row's and the column's variable where the row and column intersect, a scatterplot matrix is created. An example scatterplot of the Iris data set, which is described in Appendix D.1, is shown in Figure 2.1. The characteristics of this data set are detailed in the Appendix in Section D.1.1. However, the number of scatterplots is the square of the number of original variables, such that scatterplot matrices cease to provide an understandable presentation as the dimension exceeds 10 or 20.

This limitation gives rise to the need for comprehensive visualizations that combine as much information as possible in a single figure. Vector projection [28] is the task of mapping points from a high-dimensional input space to a low-dimensional output space while preserving as much information about the overall shape of the data cloud as possible. Usually, this is defined as maintaining the pairwise distances. In the process of vector projection information is necessarily lost due to the reduction in dimensionality. The results from the projection are used either for data preprocessing, where the number of variables is reduced to an amount that can be handled by data mining algorithms that are sensitive to the amount of input variables, or for visualization, in which case the dimension of the output space is commonly two and the projection is plotted to be inspected by a human observer. It has to be noted that the coordinate axes in such a plot do not have any semantic meaning, as opposed to the original data axes which correspond to a measurement. Any projection plot can be rotated and mirrored arbitrarily, as the only information it carries are the relative distances and densities that are preserved through the projection.

There are two types of vector projection algorithms, linear and nonlinear ones. PCA is the most prominent linear projection algorithm. It is deterministic and easy to compute. It is performed by calculating the eigenvectors and eigenvalues λ_i of the covariance matrix of data set X, and by sorting the pair of eigenvalues and eigenvectors in descending order. The data samples are mapped in an orthogonal way onto the subspace spanned by eigenvectors associated with the largest eigenvalues. The ratio of the sum of eigenvalues used for projection to the sum of all eigenvalues indicates the percentage of the variance preserved during the projection process, which is a measure of quality of the mapping. PCA works best if some variables show a high degree of linear correlation. In such a case, a



(a) (b) (c)

Figure 2.2: Iris data set, 6×11 SOM: (a) PCA projection, (b) Variance explained: The horizontal axis denotes the 4 eigenvectors, ordered by importance from left to right, the vertical axis shows the percentage of the variance explained by the corresponding eigenvector, if taken as an axis (c) Cumulated variance explained, the axes are the similar to (b), but the percentage reflects the sum of the variance explained by the space spanned by the eigenvectors up to the current one

low number of axes is sufficient to depict most of the information contained in the data set. If the input space dimension is high, or the data samples have a high intrinsic dimension, i.e. the data manifold is not centered around a low-dimensional subspace of the input space, PCA projections can become very imprecise.

Nonlinear projection methods include Sammon's Mapping [88], Curvilinear Component Analysis [19], Multi-Dimensional Scaling [99]. All of these techniques involve minimizing an energy function that is based on the difference between the pairwise distances in input and output space, and are solved with iterative gradient descent algorithms. The methods differ mainly in the definition of the cost function. Isomap [95], another non-linear projection method, relies on nearest neighbor graphs of the input data and finding the shortest path to construct the latent space. A drawback of these techniques over PCA is that only a given set of points can be projected, and points that are not available in the original data set cannot be projected unless the whole projection is recomputed. An advantage is that nonlinear projection methods produce better mappings with input spaces of higher dimension. Also, they are usually more stable with regard to outliers.

The PCA projection of the Iris data set is depicted in Figure 2.2(a). This projection explains more than 95% of the variance in the data set, thus the projection is performed nearly lossless to 2 of the initial 4 axes. It can be concluded that the Iris data set actually lives in a 2-dimensional subspace, and the remaining 2 axes consist mostly of noise. In Figure 2.2(b), the amount of variance explained per axis is shown. Figure 2.2(c) shows the cumulated sum of variance explained.

2.2 Self-Organizing Maps

Self-Organizing Maps [46, 50] are artificial neural networks that combine the properties of vector quantization and vector projection. SOMs are therefore an unsupervised learning method, as no output variables have to be predicted. Similar to k-means, the SOM consists of a number of M codebook vectors $\mathbf{m}_j$. In matrix form, the codebook is written as $\mathbf{M} \in \mathbb{R}^{M \times D}$. In the next sections, the basic properties of the SOM will be explained, with an emphasis on the similarities and differences to the k-means algorithm. The most fundamental of these differences is the introduction of an output space that puts the prototype vectors into relation to each other. The topology of this output space is described in Section 2.2.1. Neighborhood kernels, which transform the output space distance into a measure of the degree of mutual influence, are discussed in Section 2.2.2. In Section 2.2.3, the SOM training algorithm is described.

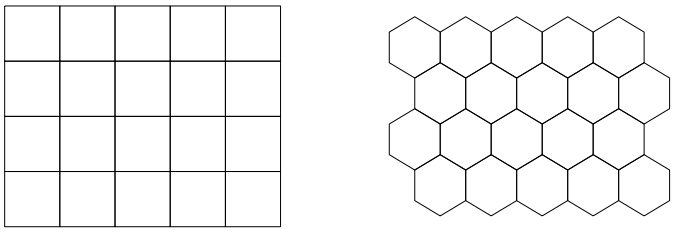
2.2.1 Topology of the output space

The main difference between the SOM and other prototype-based vector quantization methods is that the prototype vectors are ordered and are thus not interchangable. The prototype vectors are put in relation to each other by their position on a discrete output space lattice of dimension L, thus each codebook vector is assigned an additional set of coordinates that are not related to the feature space. The map unit in output space is denoted as $\xi_j \in \mathbb{N}^{M \times L}$ and its coordinates as ξ_j^k, where k specifies the coordinate. In the two-dimensional case, the output space is also referred to as the "map lattice", and the horizontal and vertical coordinates of the map units are written as ξ_j^{u} and ξ_j^{v}, respectively. For higher-dimensional output spaces, the labels of the axes are integers $1 \leq k \leq L$, and the indices are written as ξ_{jk}. The index $1 \leq j \leq M$ serves as a link between the map unit and its prototype vector. Distances can be measured between the positions in output space, formally

$$d_O(\xi_i, \xi_j) = \|\xi_i - \xi_j\| = \sqrt{\sum_{k=1}^{L} (\xi_{ik} - \xi_{jk})^2}. \tag{2.6}$$

For training purposes, the pairwise distances in output space between units are stored in the symmetric topology matrix $\mathbf{A}$, and its element a_{ij} in row i and column j refers to the distance $d_O(\xi_i, \xi_j)$. As direct neighborhood is sometimes of interest, the set of indices of the units adjacent to unit j is defined as

$$\mathbf{N}_j = \{k | d_O(\xi_j, \xi_k) = 1\} \tag{2.7}$$



(a) (b)

Figure 2.3: Map lattices for $\{4 \times 5\}$ topologies: (a) rectangular grid, (b) hexagonal grid

The map nodes are not placed arbitrarily but in an equidistant way, either in a rectangular or hexagonal manner. Hexagonal maps are used for the part of this thesis dealing with visualization, and rectangular topologies for Chapter 5. The number of nodes for each axis is thus sufficient to fully describe the discrete topology of the output space. The number of units along axis k is referred to as $\mathfrak{d}_k$, and the lattice structure of the map is denoted as $\mathfrak{L} = \{\mathfrak{d}_1 \times \mathfrak{d}_2 \times \cdots \times \mathfrak{d}_L\}$. For example, a map with a two-dimensional output space $L = 2$, a horizontal axis of $\mathfrak{d}_u = 5$ units, and a vertical axis of $\mathfrak{d}_v = 4$ units has a $M = 5 \cdot 4 = 20$ units and a topology structure of $\mathfrak{L} = \{5 \times 4\}$. Figure 2.3 shows the output spaces for this topology for rectangular and hexagonal grids.

2.2.2 Neighborhood Kernel

A crucial concept in the training and post-processing of SOMs is the neighborhood kernel h_σ, which is a monotonously decreasing function $h_\sigma : \mathbb{R}^+ \rightarrow \mathbb{R}^+$. It takes an output space distance as an argument and thus converts a distance $d_O(\xi_i, \xi_j)$ between two map units into a measure of relative proximity. Far apart units ξ_i and ξ_j have low kernel values. Thus, the kernel acts as a weighting function for the influence of nodes ξ_i and ξ_j onto each other. Kernels are used in many fields of statistics such as probability density estimation; however, for use with SOMs, the kernel does not have to be a probability function with unit area. The neighborhood kernel is written as a function of the unit coordinates, such that $d_O(\xi_i, \xi_j)$ is computed internally to make some of the equations in the later parts of this thesis more comprehensible.

There are numerous kernel functions, the one that is most commonly used is

the Gaussian kernel h_σ^G, which resembles a bell-shaped curve

$$h_\sigma^{\mathrm{G}}(\xi_i, \xi_j) = \exp\left(- \frac{d_O(\xi_i, \xi_j)^2}{2\sigma} \right). \tag{2.8}$$

The kernel value for distant nodes is decreasing exponentially and is close to zero for $d_O > \sigma$. For computational reasons, the kernel can be cut off at a threshold ("cut-off Gaussian kernel") to reduce the number of calculations:

$$h_\sigma^{\mathrm{G'}}(\xi_i, \xi_j) = \begin{cases} h_\sigma^{\mathrm{G}}(d_O(\xi_i, \xi_j)) & \text{if } d_O(\xi_i, \xi_j) \leq \sigma \\ 0 & \text{otherwise} \end{cases} \tag{2.9}$$

A very simple version of a neighborhood kernel is the bubble kernel, which is a step function that is defined as

$$h_\sigma^{\mathrm{b}}(\xi_i, \xi_j) = \begin{cases} 1 & \text{if } d_O(\xi_i, \xi_j) \leq \sigma \\ 0 & \text{otherwise} \end{cases} \tag{2.10}$$

It relies solely on the concept of cutting off outside the radius σ, weighting all distances up to this point equally. The bubble and cut-off Gaussian kernels do not resemble continuous functions.

Another choice is the inverse proportional kernel

$$h_\sigma^{\mathrm{ip}}(\xi_i, \xi_j) = \begin{cases} 1 - \frac{d_O(\xi_i, \xi_j)^2}{\sigma^2} & \text{if } d_O(\xi_i, \xi_j) \leq \sigma \\ 0 & \text{otherwise} \end{cases} \tag{2.11}$$

which is similar to the Gaussian Kernel but eventually declines to zero, and is not discontinous as the cut-off Gaussian kernel.

All of the above kernels are constructed as non-linear functions. The linear kernel, which decreases linearly from one to zero, is an exception:

$$h_\sigma^{\mathrm{l}}(\xi_i, \xi_j) = \begin{cases} 1 - \frac{d_O(\xi_i, \xi_j)}{\sigma} & \text{if } d_O(\xi_i, \xi_j) \leq \sigma \\ 0 & \text{otherwise} \end{cases} \tag{2.12}$$

In all of the above definitions of kernel functions, the parameter σ determines the breadth of the neighborhood function, such that very high values correspond to high influence of far-away and close units alike, and very low values emphasize only the direct neighbors of the map unit. It is used in the visualization techniques in Chapter 4 to control the granularity of the structures detected, serving as a smoothing parameter.

2.2.3 SOM Training

The SOM training algorithm is very closely related to the k-means training algorithm in that the prototype vectors are updated by iteratively presenting data samples. The training algorithm of the SOM moves the prototype vectors towards their final positions with the goal of performing both vector quantization and projection. At first, the model vectors have to be initialized. The final state depends heavily on this initialization, but to a lesser degree than k-means. To make this process more predictible and deterministic, it is preferrable to initialize the codebook in an orderly fashion rather than randomly. The approach applied throughout this thesis will be to perform PCA on the data set and to initialize the prototype vectors along the L eigenvectors associated with the largest eigenvalues. Various other initialization methods have been proposed [4].

After initialization, the SOM is trained for a number of T epochs. In the original sequential version of the SOM training algorithm [46] the data samples are presented to the codebook in random order. For sample $\mathbf{x}_i$ and current epoch t, the codebook is updated in the following way:

$$\mathbf{m}_j(t+1) = \mathbf{m}_j(t) + \alpha(t) \cdot h_{\sigma(t)}(\xi_j, \xi_{I(\mathbf{x}_i)}) \cdot [\mathbf{x}_i - \mathbf{m}_j(t)] \tag{2.13}$$

This update step has to be repeated for all the $\mathbf{m}_j$ for every $\mathbf{x}_i$ presented. Here, $\alpha(t)$ is the learning rate, which is a function that decreases monotonously during training. The assignment function $I(\cdot)$ is defined in Equation 2.2, and $\xi_{I(\mathbf{x}_i)}$ is the best matching unit (BMU), the map unit whose prototype vector has the shortest distance to sample $\mathbf{x}_i$. The kernel h is any of the neighborhood functions defined in the previous section. As the prototype vectors are updated to a certain degree even if they are not the best matching unit, the kernel ensures that the topology of the output space is reflected in the ordering of the prototype vectors in feature space. The kernel width $\sigma(t)$ is itself a function that decreases monotonously with time. This leads to the phenomenon that the map initially, as σ is relatively large, represents the data cloud in a crude way, and each prototype vector is aligned to even its far away neighbors in output space, while the codebook vectors increasingly specialize to the sample vectors they represent at later stages. The current epoch has finished after all the $\mathbf{x}_i$ have been presented to the SOM.

A computational shortcut to classic sequential SOM training is the Batch SOM training algorithm [49], which is similar in spirit to the LBG algorithm defined in Section 2.1.3. Other than sequential training, it does not depend on the ordering of the data samples, and does not require a learning rate parameter $\alpha(t)$. It is thus deterministic in case the SOM initialization is deterministic. Further, it allows a very efficient matrix-based implementation [113]. In the Batch SOM training algorithm, the update step of epoch t is performed by calculating the new prototype

vectors $\mathbf{m}_j$ as

$$\mathbf{m}_j(t+1) = \frac{\sum_{k=1}^{M} |\mathfrak{S}_k| \cdot h_{\sigma(t)}(\xi_k, \xi_j) \cdot \mathbf{n}_k(t)}{\sum_{k=1}^{M} |\mathfrak{S}_k| \cdot h_{\sigma(t)}(\xi_k, \xi_j)} \tag{2.14}$$

where $|\mathfrak{S}_k|$ and $\mathbf{n}_k$ are defined analogous to Equations 2.3 and 2.4. This last step, which introduces a concept of smoothing over the map topology, is the only computational difference from the LBG algorithm. As with the sequential version of the training algorithm, depending on the width $\sigma(t)$ of the neighborhood kernel, the mutual influence of neighboring map units ensures that their topology will be maintained to a certain degree in input space, thus placing topological neighbors close together in feature space.

When training has finished, the map should represent both the original data points in a vector quantization sense, and the output space topology such that prototype vectors which are close in output space are also close in input space. The width $\sigma(T)$ of the very last epoch is crucial for the shape of the SOM. If $\sigma(T)$ is high, the topological relationship between the codebook vectors is emphasized, and the quantization effect is neglected, compared to a setting with relatively low $\sigma(T)$. Thus, the value of $\sigma(T)$ is indicative of the degree to which vector projection is favored over vector quantization.

2.3 Visualization of the SOM

Visualization is one of the main strenghts of the SOM [110, 109]. In this section, a short survey of visualization techniques for the SOM is provided. Most of these are post-processing tools to aid the observer in understanding the characteristics of the data set.

The following sections are structured according to Vesanto's taxonomy of SOM visualizations [110] that groups them as codebook projections, SOM as visualization platform, and further subcategorizes them based on whether the visualization uses a data set to be shown in connection with the codebook or not. Section 2.3.1 explains the inspection of the codebook directly as a projection of input space. In Section 2.3.2, the SOM as a visualization platform is discussed, which means displaying information on the two-dimensional map grid. Section 2.3.2 provides an overview of methods that rely on information derived from the codebook, disregarding the relation of the map to the data samples. Section 2.3.2 summarizes visualization techniques that take the data distribution into account. Section 2.3.4 provides an introduction to common SOM quality measures and quality visualization.

2.3.1 Projection of the SOM Codebook

One of the most intuitive ways of inspecting the relation of the SOM to the data set is to inspect both the prototype vectors and the data samples simultaneously. This cannot be performed easily, since the input space is usually very high dimensional. A possible solution is to perform a vector projection that shows both the SOM codebook and the data vectors on a 2D plane.

Codebook projections are especially interesting for visualizing the training process by showing the way the map folds onto the data set after each epoch and for validating the quality of the map by verifying whether the shape of the data cloud fits the shape of the map. Figure 2.4 shows an example of a $\{6 \times 11\}$ SOM training process, where a map is trained on the Iris data. In Figure 2.4(a), the map is depicted directly after initialization. As a PCA-based linear initialization has been performed, the initial layout of the SOM axes are parallel to the plot axis, which is a PCA visualization based on the very same eigenvectors used to initialize the map. Figure 2.4(b) shows the map after the first epoch of batch training. The prototype vectors have moved to positions such that the map resembles the rough overall shape of the data cloud. Figure 2.4(c) depicts the projections after 5 epochs. At later stages of training as the neighborhood radius decreases, the positions are fine-tuned and the individual codebook vectors move to the centers of the data points in their immediate proximity. Figure 2.4(d) shows the map at the end of the training process, and the codebook vectors are in their final positions and ready for post-processing.

2.3.2 Visualization on the SOM lattice

The two-dimensional SOM grid can be used as a platform to display information. The lattice defined by the topology of the SOM provides the static coordinate system for such a plot, and information can be shown, for example, as color codes, markers, or the size of the background nodes. The visualization techniques discussed in the rest of this chapter are based on this approach. One possible drawback of this method is that the axes of the map are meaningless, and this can be a source of confusion to observers who are accustomed to crossplots that plot one variable against another one. The analogy to a geographical map is better suited for this kind of visualization, as relative distances are what both geographical maps and SOMs actually show.

Before discussing the various visualizations in detail, the basic visual tools that can be used for representing information on the SOM lattice are described:

- Printing numbers and text into the patches
- Printing graphics or symbols into the patches

- Coloring the patches
- Resizing the patches
- Coloring the map by interpolation across several units
- Drawing lines on the map and connecting patches

Single numeric values on the map

A common task is visualizing a numeric value for each map node, for example the number of data points with a certain characteristic mapped to a each map node. An example is given in Figure 2.5(a), where the numbers are displayed on top of the map patches. As it is not easy to grasp the overall distribution of the values, in most cases a different approach is taken and the numbers are shown color coded, as depicted in Figure 2.5(b). Another possibility is showing the values by plotting a marker onto the map patches, where the sizes of the markers are adjusted accordingly, as in Figure 2.5(c).

Multiple numeric values on the map

More challenging is simultaneously displaying multiple numeric values, or linking various layers of information. These complex forms of visualization are one of the topics of this thesis, especially of Chapter 4. Several examples are shown in Figure 2.6. Bar charts, as shown in Figure 2.6(a), display each numeric value as a colored bar in each map patch. While there is no theoretical limit for the number of bars that can be put into a patch, this number should be kept in a low range as a high number of bars is hard to read. Pie charts, shown in Figure 2.6(b), can display a number of non-negative variables as pie slices, where the size of each piece corresponds to the relative amount as part of the total of all the values. The total value of all values added can be shown as the size of the pie. Pie chart visualizations are useful for showing how an effect that can be measured on the map can be divided into contributing factors. Multiple markes, as shown in Figure 2.6(c), are an extension of the single markers, where the size of the marker represents a numeric value. This form of visualization is suited for showing distributions on the map, i.e. where samples of a certain class are located, as described for hit histograms in Section 2.3.2.

Chernoff faces [17], as used by Vesanto [110], are another way to display multidimensional values in the shape of a human face. They are complex parameterized markers and are able to display up to 10 variables as features of the face, such as nose size or eye spacing. What makes this technique unique is that the

glyphs are designed to be easily distinguishable by the human perception, which is very sensitive to changes in facial features and expressions.

There are numerous other ways to display multidimensional coordinates, but not as a visualization in combination with the SOM, for example parallel axes [37]. For an in-depth survey of these methods, please refer to [110]. A problem of these methods is overloading of a single plot with information, which makes the visualization difficult to read and understand. Thus, complex visualization techniques have to be carefully designed.

Metaphors

Other methods of displaying information on the SOM lattice exploit analogies and metaphors. The first of these display information that looks like an aerial view on islands. It is used in [63] to display the smoothed data histograms described in Section 2.3.2. It displays numeric values and shows high values as islands and low values as oceans. An interpolation is performed for the values in between the discreet positions of the map lattice, resulting in the smooth transitions that can be seen in Figure 2.7.

Another visualization metaphor is that of cartographic maps. In [94], the data is displayed along with cartographical information, depicted in Figure 2.8(a). It depicts boundaries between areas of the map that are characterized by a certain feature, in this case the occurence of a category that many of the samples mapped to a region have. The labels show the names of these characteristic categories. The relief-like visualization in Figure 2.8(b) is similar to the island visualization described in the previous paragraph.

In [29], various layers of information are presented as aquarium-like underwater scenes with fish as glyphs, depicted in Figure 2.9(a). The relief-like ground is again an interpolation of numeric values. The fish are similar to the markers that show numeric values and differ in size and color.

Another metaphor is the SkySOM [54], where a night-sky visualization is shown, as depicted in Figure 2.9(b). It shows glyphs in the form of stars. These are positioned not at the discrete positions in the map lattice, but can be placed somewhere between these positions. The rationale for doing this is to show density not by increasing the marker or glyph size but by increasing the number of stars. The stars actually represent data samples that are mapped to the SOM and their position is calculated from their distance to its best-matching unit and its surrounding units.

Finally, Mnemonic SOMs [57, 58] modify the shape of the ouput space in order to create metaphors. The example shown in Figure 2.10 is a SOM trained on the data extracted from audio files of Mozart's works, and the SOM is designed to meet the shape of Mozart's head as a recognizable metaphor.

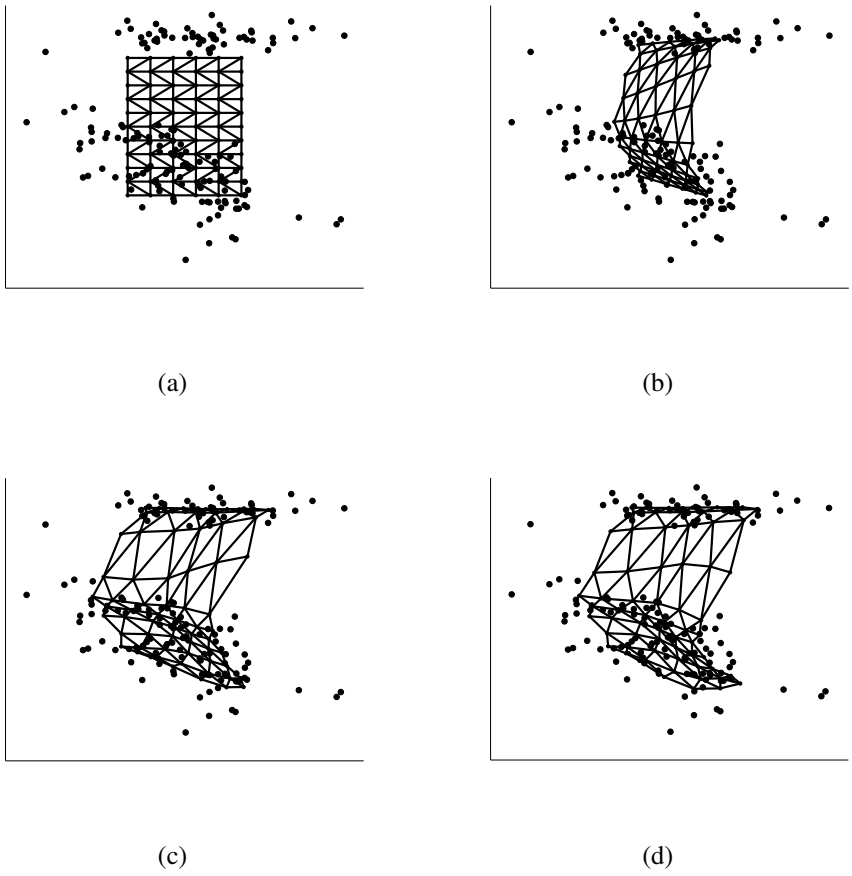
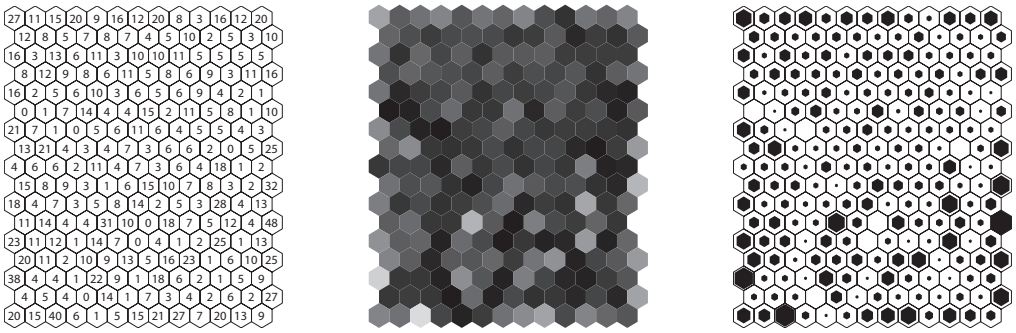
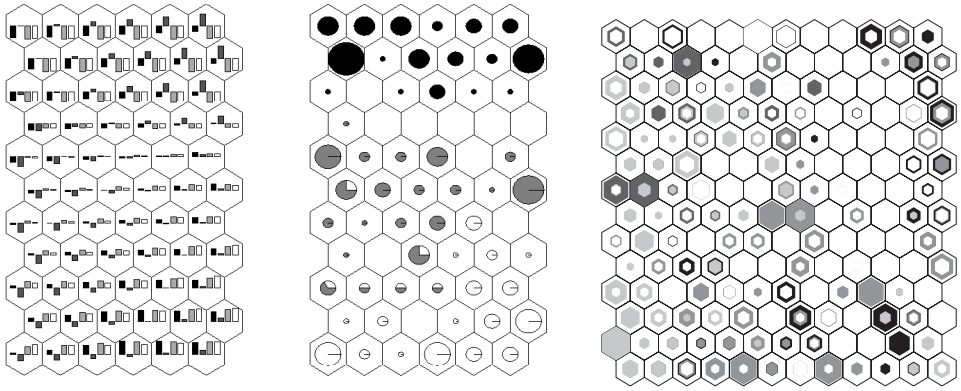


Figure 2.4: Iris data, 6×11 SOM, PCA projections of data and map at various stages during training: (a) after initialization, (b) after first epoch, (c) after 5 epochs, (d) after training



(a) (b) (c)

Figure 2.5: Visualization of numeric values on top of map lattice: (a) numbers printed directly into the patches, (b) patch is colored according to value on color scale, (c) marker size adjusted according to value



(a) (b) (c)

Figure 2.6: Visualization of multiple numeric values on top of map lattice: (a) Bar charts, (b) pie charts, (c) multiple markers

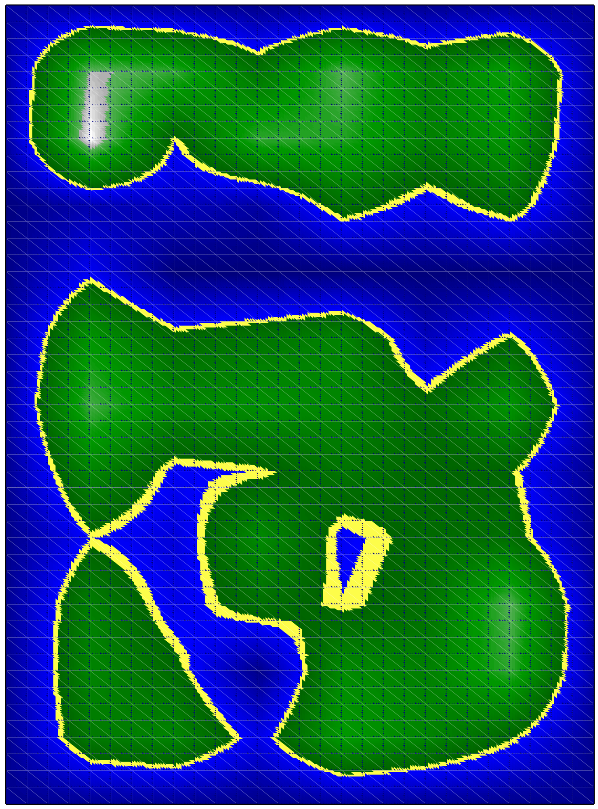
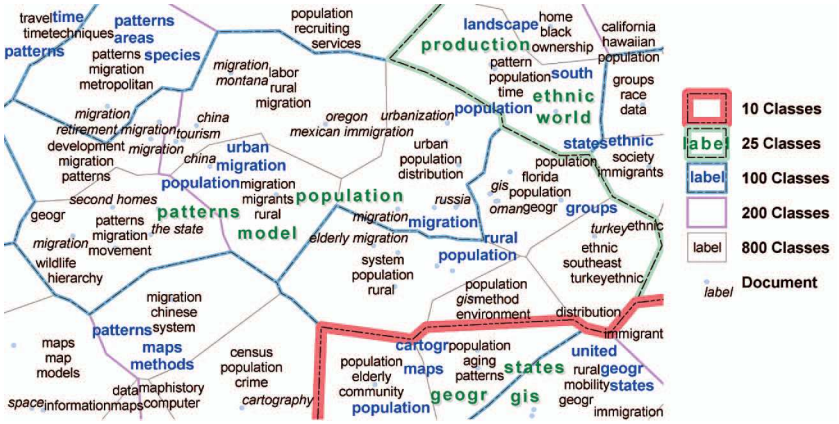
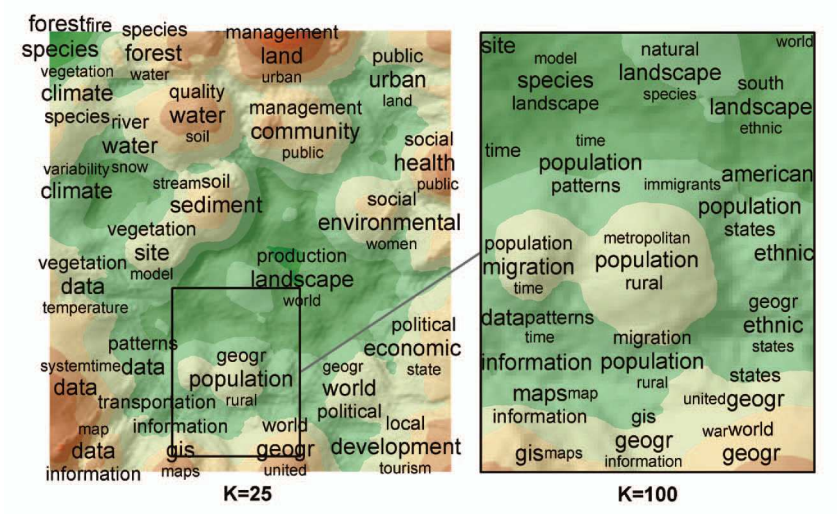


Figure 2.7: Alternative visualization methods: Islands of Music

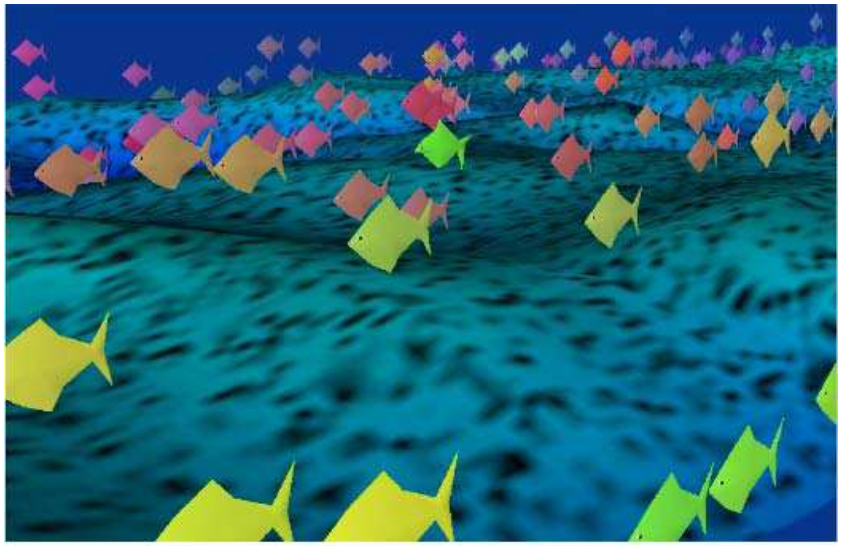


(a)

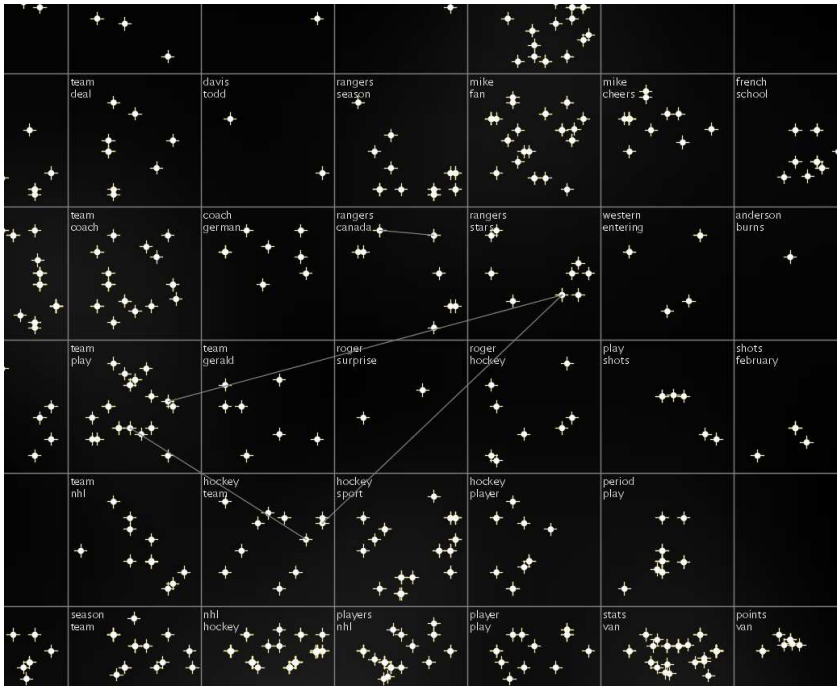


(b)

Figure 2.8: Alternative visualization methods: Visualization as cartographic map



(a)



(b)

Figure 2.9: Alternative visualization methods: (a) ReefSOM, (b) SkySOM

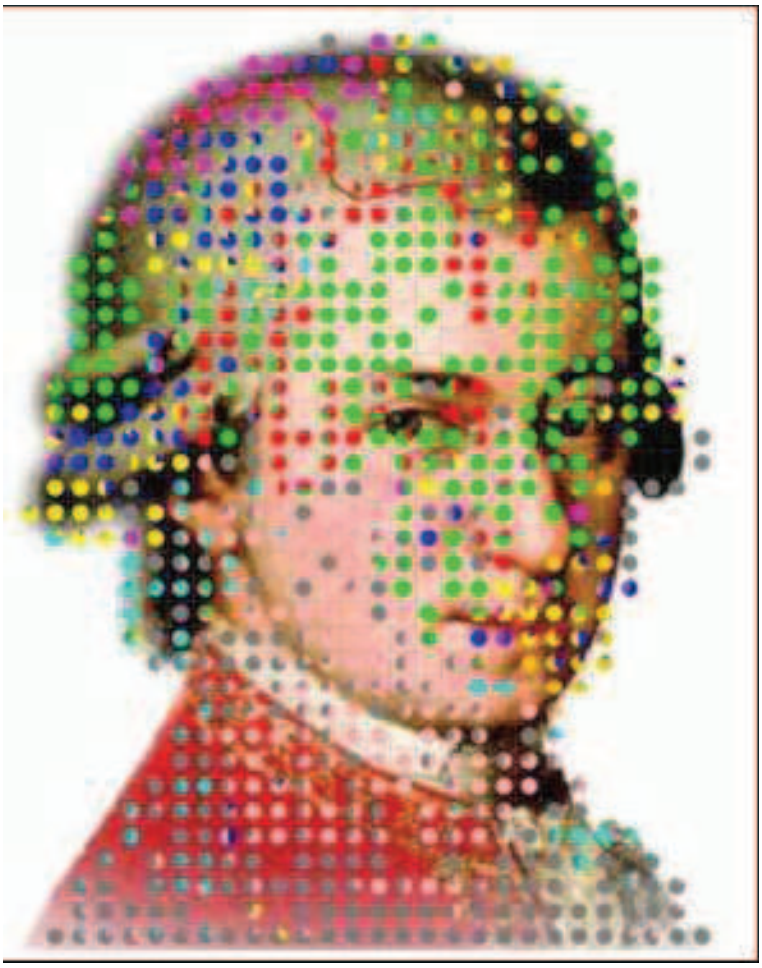
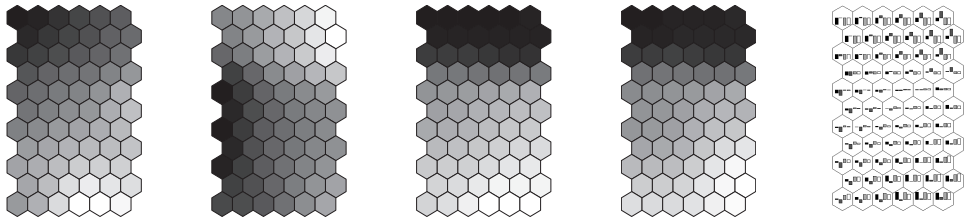


Figure 2.10: Alternative visualization methods: Mnemonic SOMs



(a) (b) (c) (d) (e)

Figure 2.11: Component planes of a $\{6 \times 11\}$ SOM on the Iris data: (a) Sepal length, (b) sepal width, (c) petal length, (d) petal width, (e) bar-chart of component planes

Codebook based visualization methods

The techniques in this category of visualization methods rely solely on the codebook vectors, disregarding the data samples. The methods that fall into this category are component planes, the U-Matrix, and clustering of the map. Considering only the codebook implies that the trained map contains most of the characteristics of the data manifold, and can thus be used as an appropriate replacement. To ascertain that this assumtion holds, or to assess the degree to which it applies, the quality measures described in Section 2.3.4 can be used. The methods described in this section are:

- Component planes
- U-Matrix
- Clustering of the codebook

Component planes

The most comprehensive method for SOM visualization are component planes. All the information contained in the codebook is displayed simultaneously. A single component plane shows the value of a selected input space variable of the codebook. The component plane of variable i is written as $\mathbf{m}_{(i)}$, the i^{th} row of matrix $\mathbf{M}$, formally $\mathbf{m}_{(i)} = (m_{1i}, \ldots, m_{Mi})$. The numeric values of $\mathbf{m}_i$ are then displayed, for example, as shown in Figure 2.11 for the Iris data set. It can be seen that sepal length, petal length, and petal width are strongly correlated, as the component planes have high and low values in similar positions. Sepal width is

the only variable that differs from this pattern. Another possibility to display component planes in a single figure is by plotting a bar-chart for each node, with a bar for each variable, as shown in Figure 2.11(e). One disadvantage that limits the use of component planes is that for high dimensional data the number of component planes becomes prohibitively large for visual inspection. Most visualization techniques are thus aimed at aggregating some aspect of the information contained in the codebook into a single plot, or by clustering the individual component planes, grouping correlated variables, as described in Section 2.3.3.

U-Matrix

The most widely used method of this sort is the unified distance matrix, or U-Matrix [104]. It is calculated as the distance in feature space between prototype vectors, for which the map units in output space are adjacent, formally

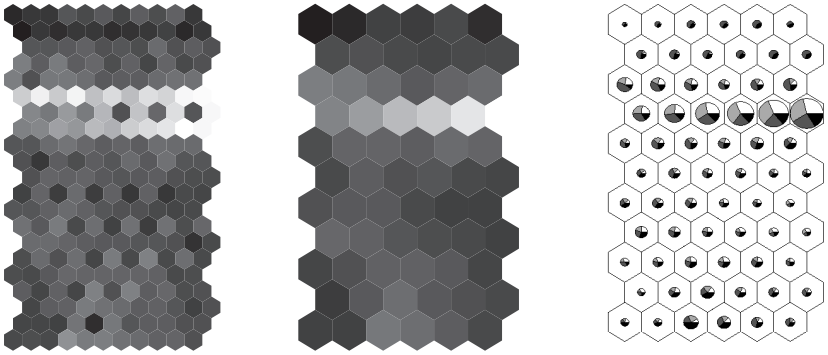
$$u(\xi_j, \xi_k) = \|\mathbf{m}_j - \mathbf{m}_k\| \tag{2.15}$$

These values are distances between nodes ξ_j and ξ_k, and for visualization purposes, the average for each node has to be computed:

$$\bar{u}(\xi_j) = \frac{1}{|\mathfrak{N}_j|} \sum_{k \in \mathfrak{N}_j} u(\xi_j, \xi_k), \tag{2.16}$$

where $\mathfrak{N}_j$ is the set of indices of units adjacent to j as defined in Equation 2.7. $u(j)$ is the average input space distance to its topological neighbors. The U-Matrix can be depicted in two ways, either by showing the inter-node distances, or by showing only the average values. The former method requires insertion of additional patches to the map lattice between nodes, and the original positions of the nodes are assigned the average values as in Equation 2.16. Figure 2.12(a),(b) show examples of both methods. It can be seen that high values are primarily located horizontally below the first third of the map. When compared to the PCA projection in Figure 2.4(d), the nodes that are connected by long lines correspond to these units with high U-Matrix values. These units are referred to as interpolating units, as they do not represent any or only very few data samples in a vector quantization sense, but serve the purpose of connecting clusters of dense areas. The U-Matrix is thus suitable to identify possible interpolating units and outliers, as well as dense regions, where inter-node distances are low. Figure 2.12(c) shows another possibility for depicting the U-Matrix. The values $u(\cdot)$ are represented by the size of the pies, and the pie slices indicate how much each input space variable contributes to the distance between the prototype vectors, formally

$$u_i(j) = \frac{1}{|\mathfrak{N}_j|} \sum_{k \in \mathfrak{N}_j} |m_{ji} - m_{ki}|. \tag{2.17}$$



(a) (b) (c)

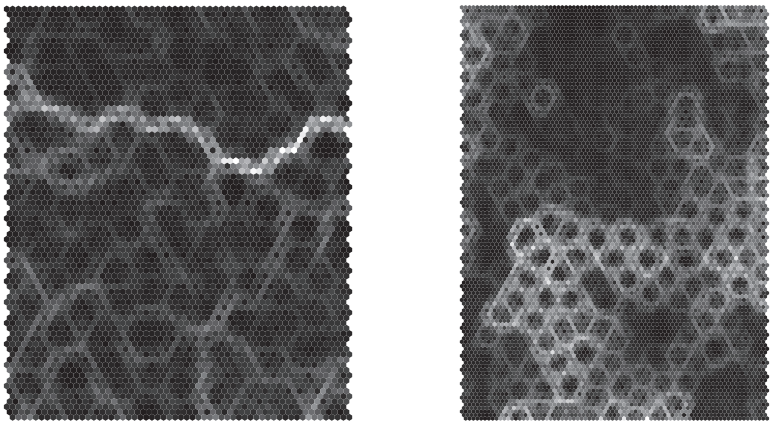
Figure 2.12: U-Matrix of a $\{6 \times 11\}$ SOM on the Iris data: (a) Inter-node distances, (b) averaged values, (c) pie-chart showing the relative contributions of the variables (from left to right: Sepal length, sepal width, petal length, petal width)

In the example in Figure 2.12(c), the differences in the boundary region are caused mainly by the petal variables, and least affected by the sepal length.

The U-Matrix explains much of the clustering structure of the SOM, especially in the case when the number of nodes is smaller than the number of training data samples. However, if the prototype vectors outnumber the training samples [101], the U-Matrix shows artifacts around the positions where the data samples are mapped, which overshadow the actual cluster boundaries. This effect can be observed in Figure 2.13 for large maps for the Iris and Ionosphere data sets.

Clustering of the map

Related to the U-Matrix in what is to be shown, clustering of the codebook vectors [112] hints at which areas of the map are dense and mutually similar in feature space. Clustering is performed by using the prototype vectors as input data for any clustering algorithm, and visualizing the results. Clustering the SOM codebook can be regarded as a two-step vector quantization process, as the data samples are quantized by performing the SOM training, and then the clustering step. The results for Ward clustering at various levels are shown in Figures 2.14(a)–(c), and k-means clustering with $k = 3$ in Figure 2.14(d), which also shows the proximity to the cluster centroid by the patch sizes. Ward's clustering is an example of hierarchical clustering, thus the partitions of higher levels, i.e. a lower number of clusters, are calculated by joining two clusters from lower levels. Stochastic partitional techniques, such as k-means, can result in different clusters each time



(a) (b)

Figure 2.13: U-Matrices of maps with a higher number of nodes than data samples: (a) Iris $\{30 \times 40\}$ SOM , (b) Ionosphere $\{40 \times 60\}$ SOM

it is performed. For practical reasons, it can be beneficial to exclude interpolating units from the codebook. The term "interpolating unit" is somewhat vague, a possible policy that could be used is to skip all units that are not selected as BMU at least once. Clustering results are often slightly better if this is performed, especially for unstable methods like single and complete linkage.

The clusters can be visualized as a projection of feature space, as discussed in Section 2.3.1, which reveals additional information on the distribution of both the data and the prototype vectors. A PCA projection is shown in Figure 2.14(e). The data samples are assigned the same color as their best-matching unit, and the convex hull of each cluster is shown. This can lead to overlapping clusters even for methods that aim at finding spherical clusters, because due to the projection some of the information is lost; in this case, however, around 95% of the variance is explained and thus the projection can be considered as very stable. This visualization method is very sensitive to outliers and leads to overlapping polygons, but can provide a good overview on the form of the clusters.

Visualization methods based on data samples

The visualization methods discussed in this section are concerned with showing the relationship between the map and a data set. This data set does not necessarily have to be the same that has been used for training, but usually it is. Depending on what the SOM is used for, any data set can be used that lives in the same

feature space. This is actually a big advantage over other visualization methods such as Sammon's mapping, as an abstraction of the original data (the codebook) is generated during the training process, which can be used for similar data sets without losing its applicability.

The methods discussed in this section are:

- Hit histograms
- Smoothed data histograms
- P-Matrix
- U*-Matrix
- Visualizations for Generative Topographic Mapping

Hit histograms

The most basic visualization technique in this category are hit histograms. They show the distribution of the data set by counting the number of times each map node is the BMU for any sample, formally

$$b(\xi_j) = |\{\mathbf{x}_i | I(\mathbf{x}_i) = j\}| = |\mathfrak{S}_j|. \tag{2.18}$$

Examples of this visualization are shown in Figures 2.15(a),(b). When compared to Figures 2.12(a) and 2.14, the horizontal gap in Figure 2.15(a) is again characteristic for an area of interpolating units, i.e. prototype vectors that occupy a region in feature space that acts as a bridge that connects two well-defined clusters. Figure 2.15(b) shows the distribution of a data set with fewer samples than map units. The prototype vectors are arranged in such a way that each sample is mapped to a node that has no other samples assigned to it, surrounded by nodes without any hits. This makes it hard to identify cluster boundaries, as the data samples are spread almost uniformly over the map, and quantization does not occur due to the large number of model vectors.

There are several similar methods that are based on the assignment of data vectors onto the map. In case the data set is labeled, the distribution of the categories can be shown with pie-charts, for example, where each slice represents the relative frequency of a class. Examples are shown in Figure 2.15(c), where all the pies are equally large, and Figure 2.15(d), where the size of each pie reflects the number of hits on the node, similar to the marker size in Figure 2.15(a). Another possible way of showing the distribution of classes is plotting different hit histograms for each class, as shown in Figures 2.15(e)–(g).

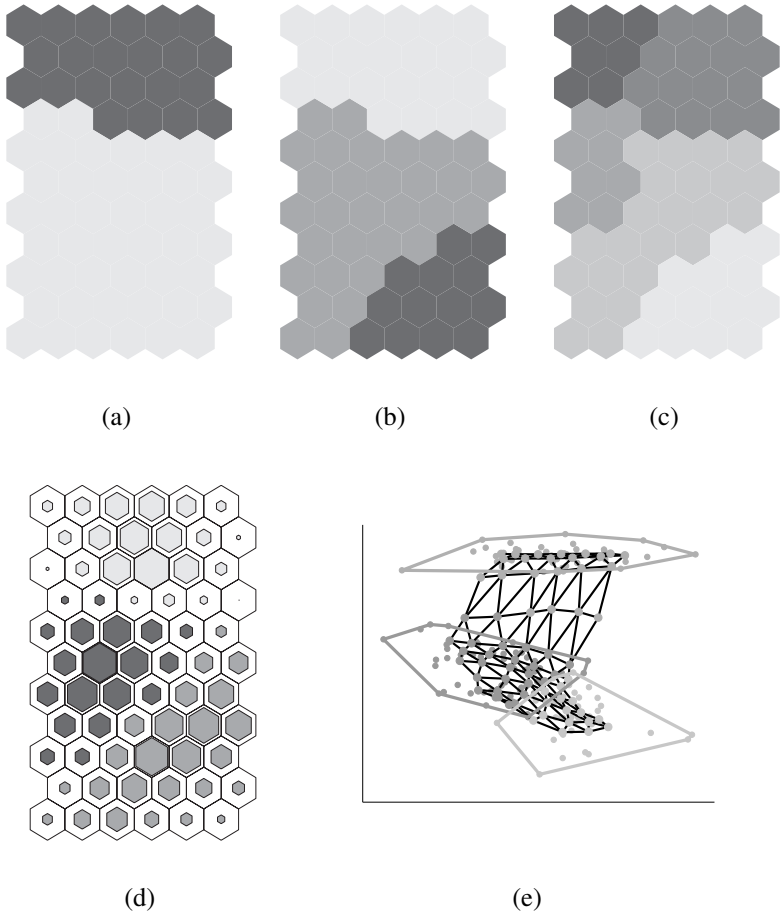


Figure 2.14: Visualization of clustering algorithms of Iris $\{6 \times 11\}$ SOM: (a) Ward, 2 clusters, (b) Ward, 3 clusters, (c) Ward, 5 clusters, (d) k-means with $k = 3$, (e) PCA projection of k-means with $k = 3$

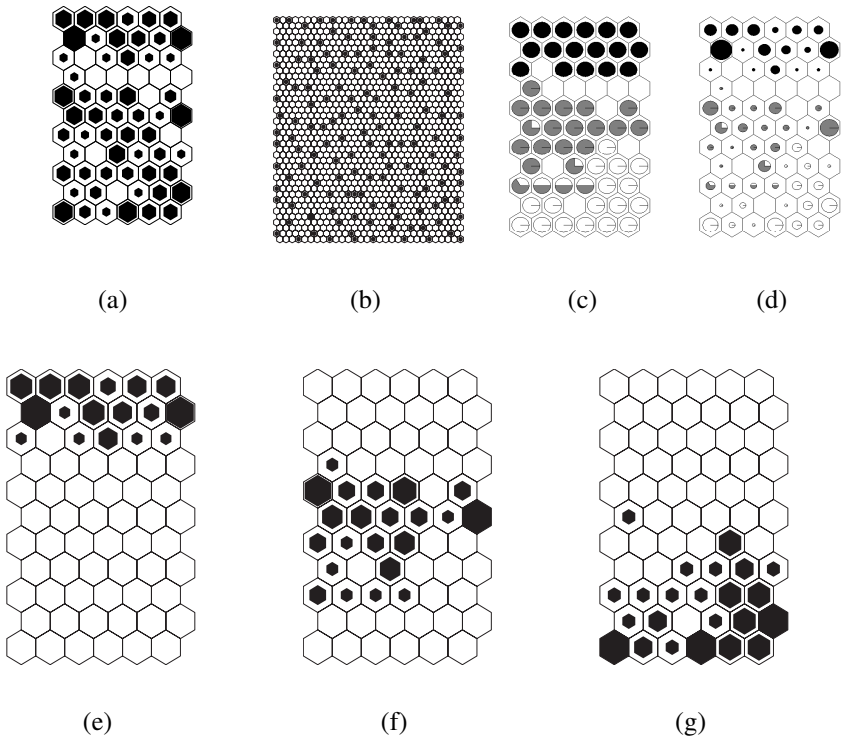


Figure 2.15: Hit histograms for the $\{6 \times 11\}$ Iris SOM: (a) small map, (b) large map; Category pie charts (black = setosa, gray = virginica, white = versicolor): (c) equal sized pies, (d) pie size according to total hits in this node; Markers by category level: (e) Setosa, (f) Versicolor, (g) Virginica

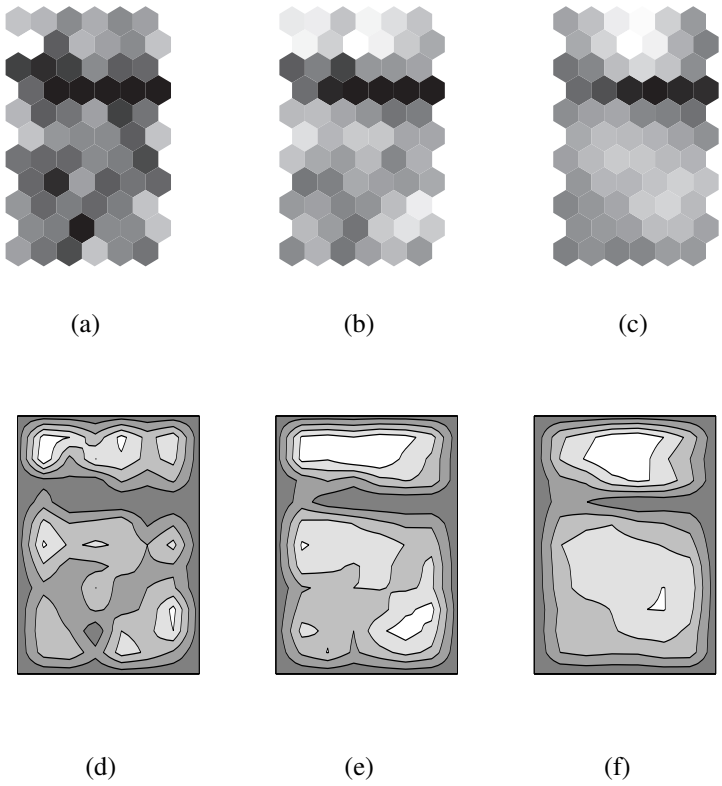


Figure 2.16: Smoothed Data Histograms for Iris data set and $\{6 \times 11\}$ SOM: (a)–(c) colored nodes, (d)–(f) smoothed contours; (a),(d) $s = 3$, (b),(e) $s = 7$, (c),(f) $s = 15$

Smoothed data histograms

A similar, but more sophisticated approach are smoothed data histograms (SDH, [64]). It is computed by increasing the counter for the BMU's bin $b(\xi_j)$ for each sample, and increasing the counter of up to k best matching units to a lesser degree. The SDH requires a spread parameter s which serves as an upper limit to k. Setting s to higher values increases the smoothing effect. The binned values can be calculated in various ways. One method is the Nearest Neighbors assignment, where the bins for the k best matching units are increased by a constant value:

$$b(\xi_j) = \sum_i^N \begin{cases} 1 & \text{if } I^{(k)}(\mathbf{x}_i) = j \wedge k \leq s \\ 0 & \text{otherwise} \end{cases} \tag{2.19}$$

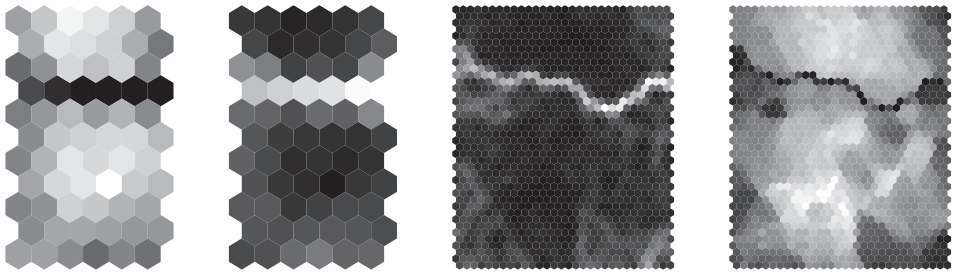
where N is the number of samples, s is the spread parameter, j is the index of the codebook vector and the map unit, and $I^{(k)}(\cdot)$ is the ranking function as described in Appendix C.1. Another calculation method is to increment the counter by $1/k$ for each sample for the k^{th}-best matching unit, resulting in declining values for more distant units. This is formally written as

$$b(\xi_j) = \sum_i^N \begin{cases} \frac{1}{k} & \text{if } I^{(k)}(\mathbf{x}_i) = j \wedge k \leq s \\ 0 & \text{otherwise} \end{cases} \tag{2.20}$$

Another method is the inverse ranking method, where the bins can be increased by up to $s - 1$ points, reduced by the rank of the distance between unit and sample:

$$b(\xi_j) = \sum_i^N \begin{cases} s - k & \text{if } I^{(k)}(\mathbf{x}_i) = j \wedge k \leq s \\ 0 & \text{otherwise} \end{cases} \tag{2.21}$$

The latter two calculation methods have been shown to perform better in terms of visualization results. Examples of the SDH visualization are shown in Figure 2.16. In the upper row of this figure, the per-node values are shown, the bottom row shows the same results with a different representation, such that the values are interpolated as in the original publication. Lower smoothing ranges have a similar appearance as the hit histogram, while increasing s has a blurring effect on the visualization, such that similar nodes in feature space share the same hit value. In Figures 2.16(d),(f) the largest spread value is shown, analyzing clusters at different levels of granularity. The gap that represented the interpolating units is still visible. The SDH visualization is thus, among other things, suitable for visualizing clusters.



(a) (b) (c) (d)

Figure 2.17: Iris data, small $\{6 \times 11\}$ and large $\{30 \times 40\}$ SOM: (a),(c) P-Matrix; (b),(d) U*-Matrix

P-Matrix

Another data-based visualization technique is the P-Matrix [102] that aims at depicting the feature space density of a map. Magnification factors [83, 115, 31] refers to the phenomenon that the available space on the map lattice is assigned roughly proportionally to the number of samples it represents, such that dense areas are stretched in output space and depicted in greater detail. Due to this effect it is not always possible to communicate the whole cluster structure through visualizations like the U-Matrix, because the density of a cluster is not sufficiently represented. The P-Matrix is computed by counting the number of samples that lie within a sphere of a certain radius r_P around each prototype vector. This radius is not a user-specified parameter but computed as the 20% percentile of the pairwise data sample distances. It is called the Pareto radius. The node values are computed as

$$p(j) = |\{\mathbf{x}_i | \|\mathbf{x}_i - \mathbf{m}_j\| < r_P\}| \tag{2.22}$$

Low values for p hint at either interpolating units or outliers. This visualization tends to be more useful for larger maps. Figures 2.17(a),(c) show the P-Matrix for a small and a large map trained on the Iris data. Both show the gap in the upper third of the map as low values. For the large map, it can be seen that in the lower right of the map the values are also low, which hints at outliers.

U*-Matrix

In an effort to make the U-Matrix usable for sparse, large maps, the U*-Matrix has been introduced [103], which combines the cluster boundary information from the U-Matrix with the density information from the P-Matrix. As discussed previ-

ously and shown in Figure 2.13(a), the U-Matrix shows high values in between the positions where the data samples are mapped to, which do not represent cluster boundaries. The U*-Matrix is computed from the U-Matrix by smoothing over the boundaries that are in dense areas, which are thus likely artifacts. The values are computed as

$$u^*(j) = u(j) \cdot \left(\frac{p(j) - \bar{p}}{\bar{p} - \max_i p(i)} + 1 \right) \tag{2.23}$$

where $\bar{p}$ denotes the arithmetic mean $\bar{p} = \frac{1}{M} \sum_{k=1}^{M} p(k)$, M denotes the number of map units. Examples of the U*-Matrix are shown in Figures 2.17(b),(d) for the Iris data. The U*-Matrix for the small map is almost identical to the U-Matrix. The large map visualization shows plateaus in a similar way as the U-Matrix does for the smaller map, which is a desired behavior. The small boundaries that have distorted the U-Matrix have been smoothed out.

Visualizations for Generative Topographic Mapping

The Generative Topographic Mapping (GTM; [10]) has been introduced as an alternative to the SOM with a continuous output space that models the probability distribution in feature space. Visualizations for the GTM are thus relevant in the context of SOM visualizations. The magnification factors visualization [8] depicts local stretching of the mapping as ellipsoids in a discrete number of latent space centers. This method is related to the Vector Fields visualization technique introduced in Chapter 4 as it explains directional changes. Magnification factors can also be computed for the SOM, where a continuous interpolation of the differences between neighboring units is applied to the discrete SOM lattice in order to perform differential analysis. Apart from that, Vector Fields differ mainly in the way that smoothing is applied: While magnification factors for the SOM show similar results as the U-Matrix, a smoothing according to an adjustable parameter is applied that defines the width of the area over which the differences are aggregated and investigated. Further extensions of magnification factors for GTM investigate their curvature [98] by investigating local geometric properties, where local directional changes are investigated and visualized. This can be used for detecting distortions in the geometry of the GTM projection manifold.

2.3.3 Visualization of correlated variables

Uncovering the factors that contribute to the hidden structure in the data sample is one of the major topics of this thesis. One of the most common data analysis techniques to find relationships between variables in the data sample is linear correlation, which indicates whether the data samples are aligned on or close to

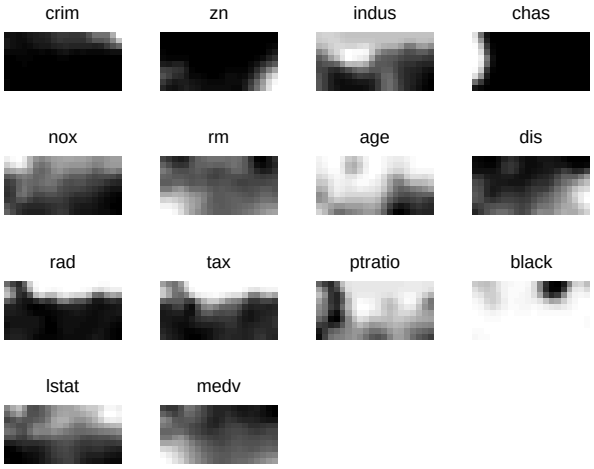


Figure 2.18: Boston housing data set, rectangular 10×20 SOM: Component planes

a linear subspace of the feature space. In this section, correlation between variables, and also nonlinear relations between variables are investigated visually with the help of the SOM. The methods presented are demonstrated with a $\{10 \times 20\}$ SOM with rectangular patches trained on the Boston housing data set, which is described in Section D.1. The component planes of this map are shown in Figure 2.18. Even as the number of variables is only 14, it is hard to spot correlation when looking at this figure.

Component plane reordering

In [111], a method is proposed to rearrange the component planes, such that the ones that have a high absolute correlation between component planes $\mathbf{M}^i$ are grouped together. The algorithm that performs the placement given this distance information can be any vector projection technique such as PCA or another SOM, the latter of which is shown in Figure 2.19. The variables for which the component planes are close on this projection are supposed to be highly correlated. Variables with component planes that are far away are likely uncorrelated. It can be seen from the figure that some components are placed close to each other, such as “rad” and “tax”, “dis” and “age”, and “medv” and “rm”. The correlation values for these variable pairs are faily high, at 0.91, -0.74, and 0.76.

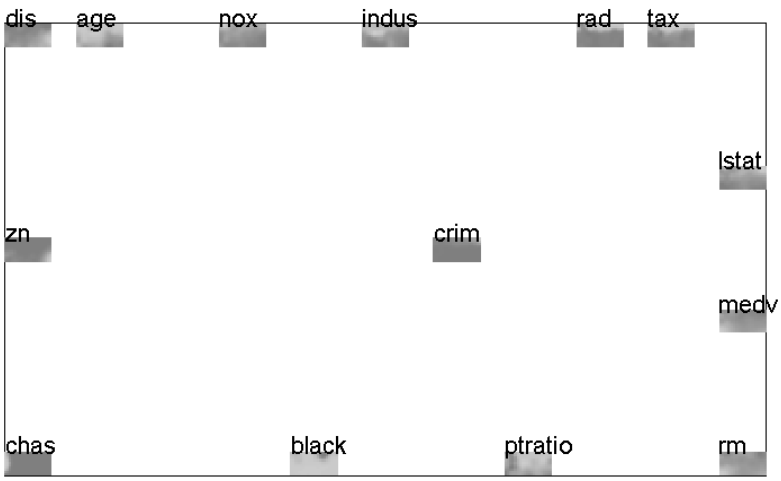


Figure 2.19: Reordered component planes for the 10×20 SOM trained on the Boston housing data

Clustering of the component planes

When clustering the transposed data or codebook matrices $\mathbf{X}^t$ and $\mathbf{M}^t$, the resulting clusters relate to the feature dimensions rather than the data samples. It is thus possible to calculate a hierarchical clustering, which requires a matrix of distances as an input. The dendrogram of clustering the variables of the data set and the component planes with Ward's clustering are shown in Figures 2.20 and 2.21, respectively. It shows how the variables are similar to each other, clustering those variables with high correlations, such as "rad" and "tax", "indus" and "nox", and "zn" and "dis" at low levels. By moving up the dendrogram, the variables are distributed into larger clusters. When comparing it to the SOM reordering in Figure 2.19, one difference is that "dis" and "age", for example, are not joined at a low level. This demonstrates that this method is able to provide a good overview but can be unreliable in the details.

When comparing the dendrograms in Figures 2.20 and 2.21, which are based on the data set as well as the codebook of a SOM trained with this data set, the differences are marginal between these two pictures. This leads to the conclusion

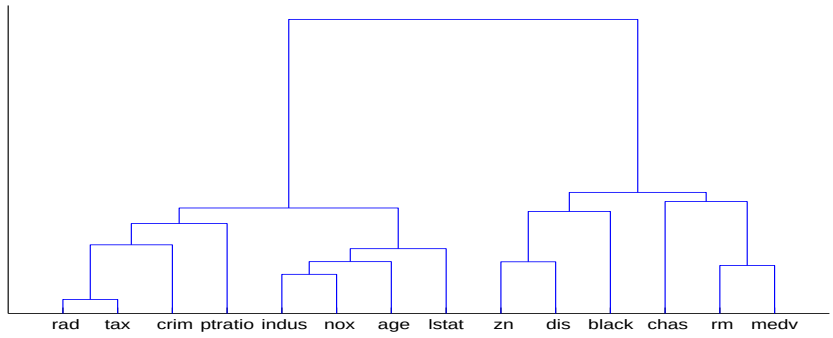


Figure 2.20: Boston housing data set: Dendrogram of Ward's clustering performed on the feature dimensions of the data set

that the codebook is a good proxy for the data set for investigation of intervariable relations. The correlation results between the original data vectors and the prototype vectors may differ, as the data set has been subjected to the nonlinear SOM projection and the codebook is not a perfect replacement for the data set. However, the difference is not very big, which leads to the conclusion that the relationship between the variables is preserved after SOM projection.

The Metro visualization

Another recent approach aims at visualizing the correlation structure in a single plot with the metaphor of metro plans [61, 62]. This method transforms component planes into curves that run along its gradient, which can be plotted on top of the map lattice. Plotting all of these lines at once results in a chaotic picture, as there are as many lines as variables, which quickly becomes confusing at higher dimensions. Using an approach that is similar to the one described in the previous paragraphs, the lines can be aggregated by replacing two similar lines by one that lies between them. An example is shown in Figure 2.22, where supposedly similar lines, which each represent a variable, are combined into a clustered view. Again, the highly correlated pairs "rm" and "medv", "zn" and "dis", and "rad" and "tax" are in the same cluster.

Local factors

Another method that is concerned with gradients investigates local factors that contribute to the clustering structure of the SOM [42], which is related to the method described in Chapter 4. This method looks for changes in the gradient of

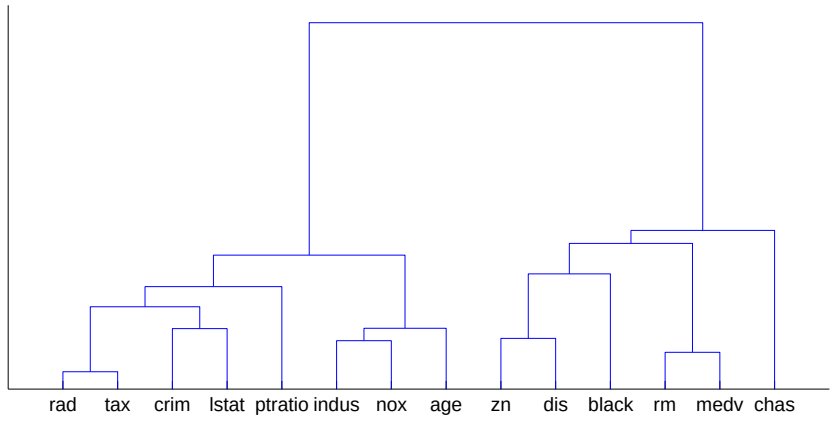


Figure 2.21: Boston housing data set, rectangular 10×20 SOM: Dendrogram of Ward's clustering performed on the feature dimensions of the codebook

the data sample density in order to identify clusters and their characteristics. The relation between the gaps separating the clusters and the discriminating factors contributing to this separation is revealed.

2.3.4 SOM quality measures

As the SOM is unsupervised, and as a certain trade-off between vector quantization and projection exists, the problem of assessing the quality arises. Several quality measures have been introduced that try to answer the question of whether a given SOM provides a good solution to either a vector projection or quantization problem. The most important of these are described in this section from a practical point of view, using the Boston Housing data set as a baseline for comparison. For surveys that cover the theoretical side, refer to [66, 67, 45, 25]. The methods discussed in this section are:

- Quantization error
- Davies-Bouldin index
- Topographic error
- Intrinsic distance
- Topographic product

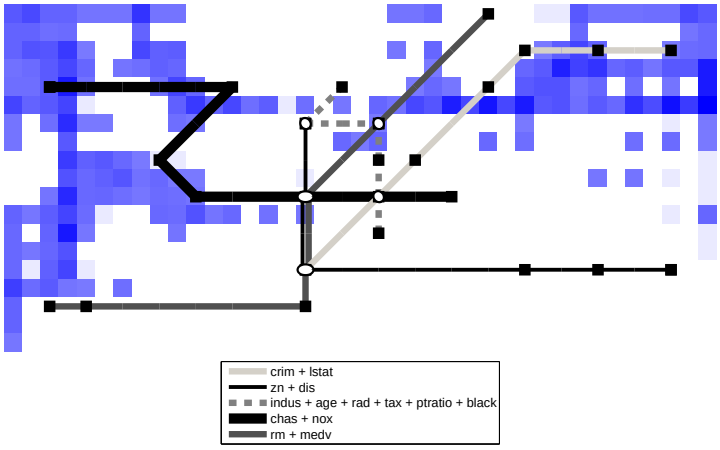


Figure 2.22: Boston housing data set, rectangular 10×20 SOM: Metro visualization on top of U-Matrix

- Topographic function
- SOM distortion

Quantization error

The quantization error, as described in Section 2.1.3, is the most widely used and most basic technique for assessing the vector quantization properties of the map. It can be used as a single index E^{Q} defined in Equation 2.5, or on a per-unit basis:

$$e_j^{\mathrm{Q}} = \sum_{i \in \mathfrak{S}_j} \|\mathbf{x}_i - \mathbf{m}_j\| \tag{2.24}$$

$$\bar{e}_j^{\mathrm{Q}} = \frac{e_j^{\mathrm{Q}}}{|\mathfrak{S}_j|}, \tag{2.25}$$

where e_j^{Q} is the total quantization error that is computed for each unit ξ_j by adding all distances from the unit's prototype vector to the data samples it represents, and $\bar{e}_j^{\mathrm{Q}}$, which is the average version that is obtained by dividing by the number of samples assigned to this unit. These values can be visualized, as shown in Figure 2.23 for a SOM trained on the Boston Housing data set. The total error resembles, to a certain degree, the hit histogram, as a higher number of hits increases the sum of the error. On the edges of the lattice, the total error is highest due to the border

(a) (b) (c)

Figure 2.23: Boston housing data set, rectangular 10×20 SOM: (a) Hit histogram, (b) total quantization errors e_j^{Q}, (c) mean quantization errors $\bar{e}_j^{\mathrm{Q}}$

effect. The mean quantization error is spread fairly evenly across the map, apart from a few gaps that indicate interpolating units. The equal distribution provides evidence that the training process has been working properly. A deviation from such an even pattern would hint at problems with training such as wrong parameterization or insufficient training duration. A possible use for the quantization error is thus to serve as a stopping criterion for the SOM training algorithm. Another scenario for using the quantization error is as a model selection criterion, where several maps are trained on the same data set and the one with the lowest error is picked. A limitation to its use is that it can only compare maps of the same size, as the quantization error usually decreases as the number of map units increases. Further, topology-preserving properties are totally disregarded, as the SOM is regarded solely from the vector quantization perspective. For this reason, when comparing the quantization errors resulting from a k-means clustering with a SOM with the same number of prototype vectors, both trained on the same data set, the k-means algorithm will usually perform better, as there is a tradeoff between projection and quantization aspects for the SOM. The quantization error can also be used to test whether the original training data set and a different data set have follow roughly the same distribution by calculating quantization errors and visually inspecting the resulting visualizations, testing whether the values are spread evenly across the lattice.

Davies-Bouldin index

The Davies-Bouldin index [18] is a prominent cluster validity measure. The way it is computed assures that spherical, internally compact clusters are rewarded that

have high distances to the other clusters. Formally, the DB-Index is computed as

$$E^{\text{DB}} = \frac{1}{c}\sum_{i=1}^{c}\max_{j\neq i}\left\{\frac{\frac{1}{|\mathfrak{C}_i|}\sum_{k\in\mathfrak{C}_i}\|\mathbf{x}_k-\mathbf{n}_i\|+\frac{1}{|\mathfrak{C}_j|}\sum_{k\in\mathfrak{C}_j}\|\mathbf{x}_k-\mathbf{n}_j\|}{\|\mathbf{n}_i-\mathbf{n}_j\|}\right\}, \quad (2.26)$$

where c denotes the number of clusters, and $\mathfrak{C}_i$ is the set of indices for cluster i, and $\mathbf{n}_i$ is its centroid. The denominator investigates the separation of the clusters, and the numerator measures whether the data samples are close to their centroids. The DB-Index is often described as penalizing intra-cluster variance and rewarding inter-cluster variance. A lower value for E^{DB} indicates a better clustering.

In the context of the SOM, the DB-Index is not helpful for evaluating the map as such. The SOM usually has more units than one would want to have clusters. The DB-Index can be used for evaluating the quality of a clustering of the prototype vectors, as described in Section 2.3.2. The DB-Index is not useful for comparing partitions produced by different clustering algorithms, as it always favors methods that result in spherical clusters over ones that find clusters of arbitrary shape. Single linkage, for example, is usually penalized. Further, it cannot be used to compare different partitionings with different numbers of clusters, as the index tends to increase with more clusters. The DB-Index can be used to compare different similar clustering algorithms on the same data, with different initializations, to select the partitioning with the lowest E^{DB}.

Topographic error

The topographic error E^{T} is a simple SOM-specific error measure that assesses the quality of the vector projection, disregarding its quantization properties. It is defined as the percentage of data samples for which the best matching unit is not adjacent to the second-best matching unit, formally

$$\mathfrak{T} = \{\mathbf{x}_i | d_O(\xi_{I(\mathbf{x}_i)}, \xi_{I^{(2)}(\mathbf{x}_i)}) > 1\} \quad (2.27)$$

$$E^{\text{T}} = \frac{|\mathfrak{T}|}{N} \quad (2.28)$$

The set $\mathfrak{T}$ contains the data samples that are considered to uncover a topology violation when projected onto the map. The topographic error is normalized to a range between zero and one, where values close to zero indicate few topology violations. Usually, the topographic error is lower the more map units are used. The topographic error should be reasonably low for a SOM, otherwise the training process has probably been interrupted prematurely.

The topographic error can be conveniently visualized by counting the number of violations that occur at each map unit that is best matching unit:

$$e_j^{\text{T}} = |\{\mathbf{x}_i \in \mathfrak{T} | I(\mathbf{x}_i) = j\}| \quad (2.29)$$

(a) (b) (c)

Figure 2.24: Boston housing data set, rectangular 10×20 SOM: (a) Topographic error per unit, (b) lines connecting units where topology violations occur, (c) intrinsic distance with 5 examples of shortest path trajectories

This per-unit error e_j^{T} can then be plotted on the map lattice, as shown in Figure 2.24(a) for a SOM trained on the Boston housing data set. Light patches indicate high errors. In total, for 53 out of the 506 samples the best matching unit is not next to the second-best matching unit, resulting in $E^{\mathrm{T}} = 0.104$, or 10.4%. Another visualization connects map nodes between which the violations occur, such that a line is drawn from the best- to the second-best matching unit of each $\mathbf{x}_i \in \mathfrak{T}$. An example is given in Figure 2.24(b). The rationale for these visualizations is to look for systematic irregularities, or clusters of errors. In the example given, the errors are not spread evenly, as the lower right-hand side of the map is occupied by many of the errors. From the U-Matrix in Figure 2.22, it can be seen that this part of the map does not have any boundaries. It is therefore likely that this region is a cluster. It may seem counter-intuitive that the topology violations occur in this area, but as the dense parts in the data cloud are stretched due to magnification factors during SOM training, data samples may be represented by units where the second-best match is not adjacent. The topographic error is a very simple quality measure, and in this case does not produce a good result, as the remaining quality measures in this section will produce contradicting results.

Intrinsic distance

A quality measure that combines and extends the ideas of topographic and quantization error is the intrinsic distance [41]. As with the topographic error, for each data sample the BMU and 2^{nd} BMU are determined. The error is computed by adding the distance from the data sample to its BMU in feature space, which is the quantization error, and the distance from the BMU and the 2^{nd} BMU. The latter distance is computed as the shortest path between these units, taking into account the map topology, such that each path may only run along neighboring

units' prototype vectors. Formally, the intrinsic distance is written as

$$e^{\text{ID}}(\mathbf{x}_i) = \|\mathbf{m}_{I(\mathbf{x}_i)} - \mathbf{x}_i\| + \min_{\mathfrak{K}} \sum_{j=1}^{|\mathfrak{K}|-1} \|\mathbf{m}_{\mathfrak{K}_k} - \mathbf{m}_{\mathfrak{K}_{k+1}}\|, \tag{2.30}$$

where $\mathfrak{K}$ is an ordered set of indices of a possible path from unit $I(\mathbf{x}_i)$ to $I^{(2)}(\mathbf{x}_i)$. The subscripts denote the index of the prototype vectors, so $\mathfrak{K}_1 = I(\mathbf{x}_i)$ and $\mathfrak{K}_{|\mathfrak{k}|} = I^{(2)}(\mathbf{x}_i)$. The intrinsic distance for a map is calculated as the sum over all individual errors of the samples:

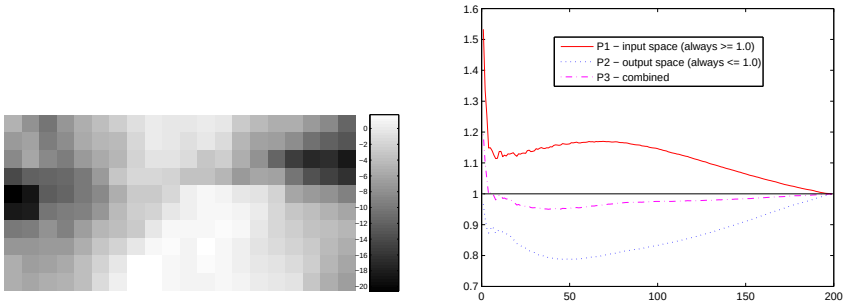
$$E^{\text{ID}} = \frac{1}{N} \sum_{i=1}^{N} e^{\text{ID}}(\mathbf{x}_i). \tag{2.31}$$

The intrinsic distance is computationally very light despite relying on non-trivial algorithms such as shortest path calculation. An example of the intrinsic distance for a SOM trained on the Boston housing data set is shown in Figure 2.24(c). Analogous to the topographic error, the errors are plotted for the map units by aggregating each prototype vector's assigned sample vectors $e^{\text{ID}}(\mathbf{x}_i)$. The light values denote units with high total topographic errors. In this figure, lines are plotted for symbolizing the shortest path for 10 random data vectors.

One of the intentions of the original paper was to introduce a method for comparing SOMs, which it is well suited for as it measures both quantization and projection quality. However, the capabilities to assess SOMs of different sizes are limited, as both the component similar to the quantiztion error, which is the first part of Equation 2.30, and the component that computes the shortest path usually decrease as the number of map units increases. When compared to the topographic error in Figure 2.24(b), the visualizations look similar as the errors are largest where BMU and second BMU are not next to each other. However, the error values do not peak at the same units, as the intrinsic distance adds a term that is related to quantization effects.

Topographic product

A measure that investigates solely the vector projection quality is the topographic product [6]. Other than the previously discussed measures, it is calculated from the codebook only, disregarding the data set it has been trained on. The result of calculating the topographic product indicates whether the intrinsic dimensionality of feature and ouput space are mismatched. The algorithm relies heavily on ranking the prototype vectors and their corresponding unit vectors in both feature and output spaces according to their proximity.



(a) (b)

Figure 2.25: Boston housing data set, rectangular 10×20 SOM: (a) Topographic product P per unit, (b) functions $P_1(k), P_2(k), P_3(k)$ of the k nearest neighbors in feature and output spaces

The topographic product is computed based on two ratios, $P_1(k)$ and $P_2(k)$. The distortion in feature space is measured in P_1 and the distortion in output space in P_2. A distortion is defined as a mismatch in the ranking of the k closest prototypue vectors in feature and output space, respectively. If the ranking is perfectly equal in both spaces, P_1 and P_2 are equal 1. The value $P_3(k)$ is the geometric mean of $P_1(k)$ and $P_2(k)$. The measures P_1, P_2 and P_3 can be evaluated for single map units only. A combined value for all map units is obtained by summing up all values of P_3 for all k, resulting in the topographic product P. This value is convenient to read: If $P < 0$, the map is too small, it either has too few map nodes, or its output space dimension is too low. In case $P > 0$, the map is too big for the data space it represents. $P = 0$ indicates a perfect match.

An example of the topographic product of a SOM trained on the Boston housing data is shown in Figure 2.25. In 2.25(a), the per unit values of P are shown. A unit value greater than 0 indicates that in the part of the feature space represented by this unit, the structure is locally too high-dimensional, and a value less than 0 hints at a too low-dimensional topology. In this example, the bright area in the center that extends to the lower right-hand part of the map has values close to zero. This means that the output space is able to match feature space in dimensionality. In the regions closer to the border, the value becomes negative, thus the output space cannot match the feature space. The overall value for this map is $P = -0.023$, which is sufficiently close to zero to conclude that both the number of units and the dimension of the output space are appropriate.

The topographic product has several shortcomings. The data set that the SOM

should ultimately represent is not taken into account. In certain circumstances, the topographic product penalizes folds that correctly relate to the topology of the data manifold, especially when the shape of the data set is folded and the ends of this shape are closer to each other than to the middle part [116]. In most cases, increasing the output space above 2 is not an option anyway to deal with a case where $P < 0$, but it can be used to measure the degree of topology violation.

Topographic function

The topographic function [116] extends the topographic product in taking the training data set into account. Similar to the topographic product, the topographic function measures the topology preservation quality of a SOM, from feature to output space, and vice versa. Quantization properties are not considered. The topographic function uses a data-based definition of connectivity between map units in input space. Two prototype vectors are considered neighboring if there is at least one data sample for which they are best and second best matching units. Using this definition, it avoids classifying correct folds as errors. As another consequence, however, parts of the prototype vectors may be isolated from the rest of the map, as there may be clusters with interpolating units in between that are not BMUs at all. This is especially problematic for interpolating units that are never selected as BMU or second BMU, but may also happen when the data set has clearly defined clusters that are well separated, in which case areas on the map that represent these clusters may not be connected at all. To avoid this problem, some additional assumptions are required. In the original publication, it is suggested to use maps with at least 60 times the number of data samples as prototype vectors. Another suggestion is to relax the definition of connectivity by using information from the training process, such that connections can be created after each training epoch between BMU and second BMU for each sample, most likely increasing the number of connections. Both approaches reduce the frequency that this problem occurs, but cannot guarantee its prevention.

The topographic function requires a definition of adjacency in both feature and output space, the one of the output space is defined by the SOM topology, and the one for the feature space is defined by the connectivity described in the previous paragraph. Next, two functions are computed: One is based on units that are adjacent in output space, but the prototype vectors of which are not next to each other in feature space; the other one counts the number of prototype vectors that are next to each other in feature space but not adjacent on the map. The result of the topographic function is derived from these two functions. In case this value is 0, the topology preserving quality is perfect, for positive results, there are topology violations present.

For the topographic function, the example using the Boston housing data set

cannot be computed as the map does not fulfill the criterion of having 60 times as many data than prototype vectors. For this data set, the graph is not connecting all prototype vectors and thus contains isolated areas. The topographic function can not be computed in this case, as the distance calculation between two codebook vectors requires the existance of a path connecting them in the graph.

SOM distortion

As opposed to k-means, the SOM has been shown not to minimize an energy function in the general case [21]. However, for a constant radius σ such an energy function does exist, which is called the SOM Distortion [48, 27]:

$$E^{\mathrm{D}} = \sum_{i=1}^{N} \sum_{j=1}^{M} h_{\sigma}(\xi_{I(\mathbf{x}_i)}, \xi_j) \cdot \|\mathbf{x}_i - \mathbf{m}_j\|^2 \tag{2.32}$$

The SOM Distortion equation has some similarity to the batch SOM training algorithm in Equation 2.14. The SOM Distortion equation can be reshaped to show the contribution of a single sample or prototype vector to the overall distortion, the latter of which is more important:

$$e^{\mathrm{D}}(\mathbf{m}_j) = \sum_{i=1}^{N} h_{\sigma}(\xi_{I(\mathbf{x})}, \xi_j) \cdot \|\mathbf{x}_i - \mathbf{m}_j\|^2 \tag{2.33}$$

The above decomposition on a per-unit basis enables visualization of the SOM Distortion Measure. An example is shown in Figure 2.26(a).

The SOM Distortion can be further decomposed into three terms [53, 114] that provide insight into the cause of the distortion, whether it is caused by quantization or projection effects:

$$E^{\mathrm{D}} = E^{\mathrm{D}}_{\mathrm{Qu}} + E^{\mathrm{D}}_{\mathrm{NB}} + E^{\mathrm{D}}_{\mathrm{NV}} \tag{2.34}$$

$$E^{\mathrm{D}}_{\mathrm{Qu}} = \sum_{j=1}^{m} H_j \cdot \sum_{\mathbf{x} \in \mathfrak{S}_j} \|\mathbf{x} - \mathbf{n}_j\|^2 \tag{2.35}$$

$$E^{\mathrm{D}}_{\mathrm{NB}} = \sum_{j=1}^{m} |\mathfrak{S}_j| \cdot \|\mathbf{n}_j - \bar{\mathbf{m}}_j\|^2 \tag{2.36}$$

$$E^{\mathrm{D}}_{\mathrm{NV}} = \sum_{j=1}^{m} |\mathfrak{S}_j| \cdot \sum_{k=1}^{M} h(\xi_k, \xi_j) \cdot \|\mathbf{m}_k - \bar{\mathbf{m}}_j\|^2 \tag{2.37}$$

where the weighting factor H_j is the cumulated influence of other map units on unit ξ_j:

$$H_j = \sum_{i=1}^{M} h_{\sigma}(\xi_i, \xi_j), \tag{2.38}$$

and $\bar{\mathbf{m}}_j$ is the kernel-weighted mean of the prototype vectors

$$\bar{\mathbf{m}}_j = \frac{1}{H_j} \sum_{k=1}^{M} h_\sigma(\xi_j, \xi_k) \cdot \mathbf{m}_k \tag{2.39}$$

The first of these components is the quantization error term $E^{\mathrm{D}}_{\mathrm{Qu}}$, which measures the distance between the prototype vector and the data samples assigned to it. As it is weighted by the kernel factor Equation 2.38, it is not identical to the definition of E^{Q} in Equation 2.24. The quantization error, as the name implies, is a measure of the quantization properties of the map. It is almost identical to the quantization error as introduced in Formula Equation 2.5, except for the weighting factor. The weighting factor reduces the distortion of units at or close to the border of the map. This actually explains the border effect, as these units are more densely populated than the ones in the center due to there is less penalty associated with causing errors at the edges.

The second component, the neighborhood variance $E^{\mathrm{D}}_{\mathrm{NV}}$, is a measure for the topology preservation qualities of the SOM. It is computed as the squared deviation from the centroid of prototype vectors, weighted by the neighborhood kernel and the number of samples assigned to a prototype vector. It is lowest for stiff maps that seldomly fold. The higher the number of map units, the lower the neighborhood variance given the same training data set.

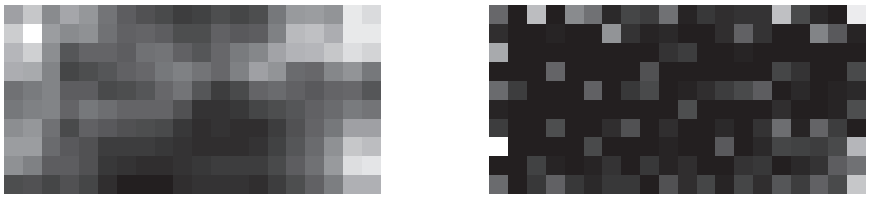
The last of the three components is the neighborhood bias $E^{\mathrm{D}}_{\mathrm{NB}}$, which is a hybrid measure for quantization and projection quality. It is computed as the squared deviation of the centroid $\mathbf{n}_j$ of the samples assigned to prototype vector j from the centroid $\bar{\mathbf{m}}_j$, which is computed over the whole data set, but weighted by the kernel according to distance from the best-matching unit.

Examples of the SOM distortion of the Boston housing SOM are shown in Figure 2.26. The total distortion is shown in Figure 2.26(a). The light values indicate high distortion and coincide with the region that contains the interpolating units that occupy the gap between the well-separated clusters in this data set. The relative contribution of each of these factors is shown in Figure 2.26(e). It can only be shown for units that are occupied by at least one data sample, thus some of the patches are empty. When attributed to the distinct factors, 23.2% of the SOM distortion is caused by quantization error, 41.2% by neighborhood bias, and 35.6% by neighborhood variance. This is a fairly normal distribution, since neither of the components is too dominant. For maps where the number of codebook vectors is very low when compared to the data samples, the quantization error will dominate; for very high-dimensional maps, the neighborhood variance will dominate. The neighborhood bias value is strongest for maps where the map size is not too small or the dimension too high, thus a situation where it is high indicates that the map is good at representing the data set, which is the case

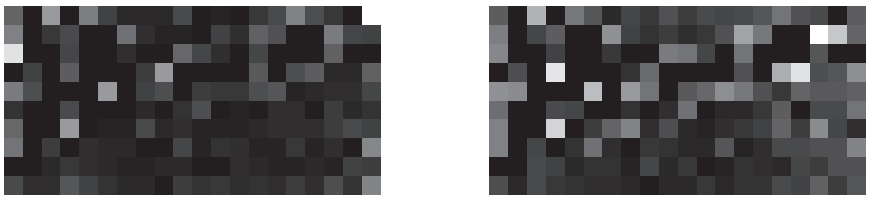
here. From Figure 2.26(e), it would seem that the neighborhood variance, and thus topology violation effects are dominant in the center and lower right parts of the map; however, Figure 2.26(f) shows the same pie chart, only the sizes of the pies are scaled to the amount of total distortion of each unit. The center and lower right regions do not cause much of the overall error, and the main contributors are located mostly at the borders, which is similar to the result from the topographic product, but contradicts the topographic error.

For computation of the SOM Distortion measure, the data set is required in addition to the trained map. As it is not originally thought to measure the quality of a mapping as such, but the energy function that the SOM training algorithm minimizes, it can serve as a criterion to stop training when the training iterations do not improve the distortion any more. The SOM Distortion can be used for selection of the best map when several maps have been trained on the same data set in parallel to avoid suboptimal solutions, which are local minima of the SOM distortion. If the SOM distortion is used to quantify the error of a map and visualize it, the neighborhood kernel has to be selected and parameterized. It should be the same as the kernel used in the training phase along with the final value for the radius $\sigma(t)$. For a radius of 0, the SOM Distortion equals the total quantization error.

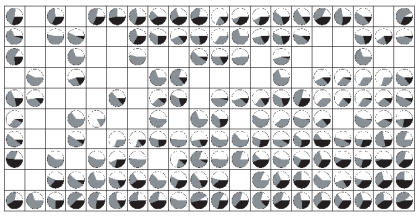
The SOM Distortion measure is very efficient to compute. Apart from having a solid theoretical background, other advantages include that the SOM Distortion can be decomposed and the contribution of each data sample can be computed, and the SOM Distortion can be attributed to each map unit. The SOM Distortion is well suited for model selection, but is not applicable for comparing maps of different sizes.



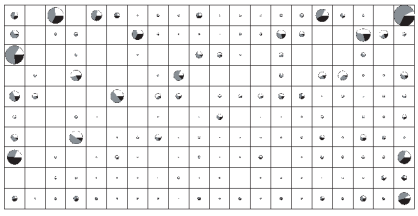
(a) (b)



(c) (d)



(e)



(f)

Figure 2.26: Boston housing data set, rectangular 10×20 SOM: (a) Total SOM distortion, (b) Quantization Error $e^{\mathrm{D}}_{\mathrm{Qu}}$, (c) Neighborhood Bias $e^{\mathrm{D}}_{\mathrm{NB}}$, (d) Neighborhood Variance $e^{\mathrm{D}}_{\mathrm{NV}}$, (e) relative contribution (black = $e^{\mathrm{D}}_{\mathrm{Qu}}$, gray = $e^{\mathrm{D}}_{\mathrm{NB}}$, white = $e^{\mathrm{D}}_{\mathrm{NV}}$), (f) relative contribution, same as (e), but scaled to the size of the SOM distortion for each unit

2.4 Summary

In this chapter, several data mining techniques have been described. First, preprocessing of table-format data has been described, then the difference between the supervised and the unsupervised settigs have been discussed. Finally, vector projection and vector quantization are described, which are important features of the methods discussed in the later chapters.

An overview of the Self-Organizing Map has been given, which is the main data mining technique that is used in this thesis. Special emphasis is laid on the topology of the map, which refers to the layout of the map nodes in the output space lattice. Neighborhood kernels, which are mathematical functions that transform the distance between two points into a measure of their proximity, have been described along with the various variants of this function. The SOM training algorithm, which creates the map out of the training data set, has been introduced.

The largest part of this chapter has been dedicated to an overview of postprocessing methods for trained SOMs, i.e. visualization methods. There are many ways that information can be displayed, such as coloring the map patches, labelling the map, or printing markers and glyphs on top of the map. The actual visualization methods that highlight the properties of the data set and the map have been described, which use these visual tools. These visualization methods include techniques that are based on the codebook only, such as U-Matrix, component planes, and codebook clustering. They depict the variables and the clustering structure of the codebook, where the codebook is seen as a meaningful proxy for the data it has been trained on.

The next class of visualization techniques that has been descussed are those that put the data and the map into relation. The most important of these is the hit histogram, which shows how the data is distributed across the map. Smoothed data histograms and the P-Matrix are centered around showing where data is particulary dense. The U*-Matrix is a technique for especially large maps to show clustering structure where the U-Matrix is not applicable.

The next class of visualization methods for SOMs that has been introduced highlights correlation between variables, including reordering of the component planes in order to move similar ones close to each other, hierarchical clustering of component planes, and the Metro visualization that plots the gradients of component planes as lines and clusters similar ones, creating a plot that resembles a subway plan.

Finally, visualizations that show the quality of the SOM in terms of quantization and projection have been described. While the quantization error and DB-Index measure the quantization properties of the map, the topographic error, intrinsic distance, topographic product, and topographic function aim to uncover topology violations. The former two methods thus measure the effect that comes

from the reduction of data samples to a lower number of codebook vectors, while the latter measure the effect of reducing the number of variable dimensions to the usual output space dimension of two. The SOM distortion has been shown to be a combined measure for quantization and projection effects. It is the most natural of the quality measures as its minimization is the energy function of the SOM algorithm as a optimization problem.

Chapter 3

Graph based cluster visualization

3.1 Introduction

In this chapter, a method is introduced that combines the visualization of how densely data is distributed within clusters with the visualization of how clusters relate to each other. This technique requires the definition of a graph structure that describes the pair wise proximity of data set vectors. This graph is then transferred into the output space, where it can be visualized. Several examples are shown that compare this visualization method to similar ones like P-Matrix and SDH. Work on the Graph visualization has first been published in [74, 75].

Section 3.2 discusses basic concepts of density visualization and how to display topology violations. This chapter uses artificial data sets and several benchmark data sets for experiments, which are detailed in Section D.1 and Section D.2, respectively. Among the benchmark data sets, the focus lies on the Ionosphere data set because it is a good example of a data set that is almost evenly split between samples occupying dense and sparse regions. Two of the artificial data sets are especially introduced for benchmarking how an algorithm copes with pure clustering or dimensionality reduction scenarios. The third of the artificial data sets is used to demonstrate how the algorithm performs when it is simultaneously presented subclusters with different densities and shapes. Section 3.3 introduces the graph visualization. A series of experiments is given in Section 3.4. In Section 3.5, the method is systematically analyzed and put into context of other visualization methods, and guidelines for its use are given. Section 3.6 concludes the findings of this chapter and summarizes its results.

3.2 Data sets

3.2.1 Data density and cluster proximity

Data density in feature space, as described in Section 2.3.2, can be identified with a variety of visualization methods. Providing a means for identifying which clusters are close in feature space is more difficult. For demonstration of the Graph method that is introduced in this chapter, two artificial data sets are defined, the Equidistant clusters data set and the Fully-connected data set. Both are constructed from a fully connected graph structure with c vertices, where the vertices are placed in a $(c-1)$ dimensional input space, such that the distance between any pair of vertices is the same. The vertices and edges of this structure resemble familiar geometric shapes, such as equilateral triangles in two dimensions, and tetrahedrons in three dimensions.

The Equidistant clusters data set is constructed by using the vertices as centers for Gaussian clusters. The resulting clusters are well separated if the standard deviation of the normal distribution is low compared to the common inter-cluster distance. The challenge in visualizing this data set with a projection algorithm like the SOM or PCA lies in displaying it such that the clusters can be identified. The Equidistant clusters data set is thus used to test the quantization properties of an algorithm that also performs vector projection. Linear projection algorithms, such as PCA, are expected to perform worse on this data set than non-linear ones, due to the way that the clusters are positioned.

Examples for the Equidistant clusters data set are shown in Figures 3.1–3.3, where data points are sampled from 3-, 5-, and 8-cluster settings, respectively. For each cluster 100 data points are generated. The figures contain visualizations of the U-Matrix, hit histograms, PCA projection of data set and codebook, SDH, U*-Matrix, and P-Matrix. In the first example in Figure 3.1, there are only 3 clusters, the centers of which lie on a two-dimensional plane. As the projection is almost lossless, apart from the Gaussian noise around the cluster centers, the SOM visualizations show a clear separation of the clusters, as well as revealing the equidistant nature of the three regions in the PCA plot. The SDH and P-Matrix visualizations show the clusters themselves, while the U-Matrix and U*-Matrix show the gaps between the clusters, i.e. the equidistant separation between the clusters. In Figure 3.2, the same set of visualizations is shown for 5 clusters. As in the previous example, the clusters are well-separated, as evident from the hit histogram visualization. Here, the problem of the dimensionality reduction becomes obvious. The clusters cannot possibly be arranged on the two-dimensional output space such that the regions appear equally distant to each other. The PCA projection fails to discover any structure in the data set. In Figure 3.3, an Equidistant clusters data set is shown with 8 clusters, thus an originally 7-dimensional structure

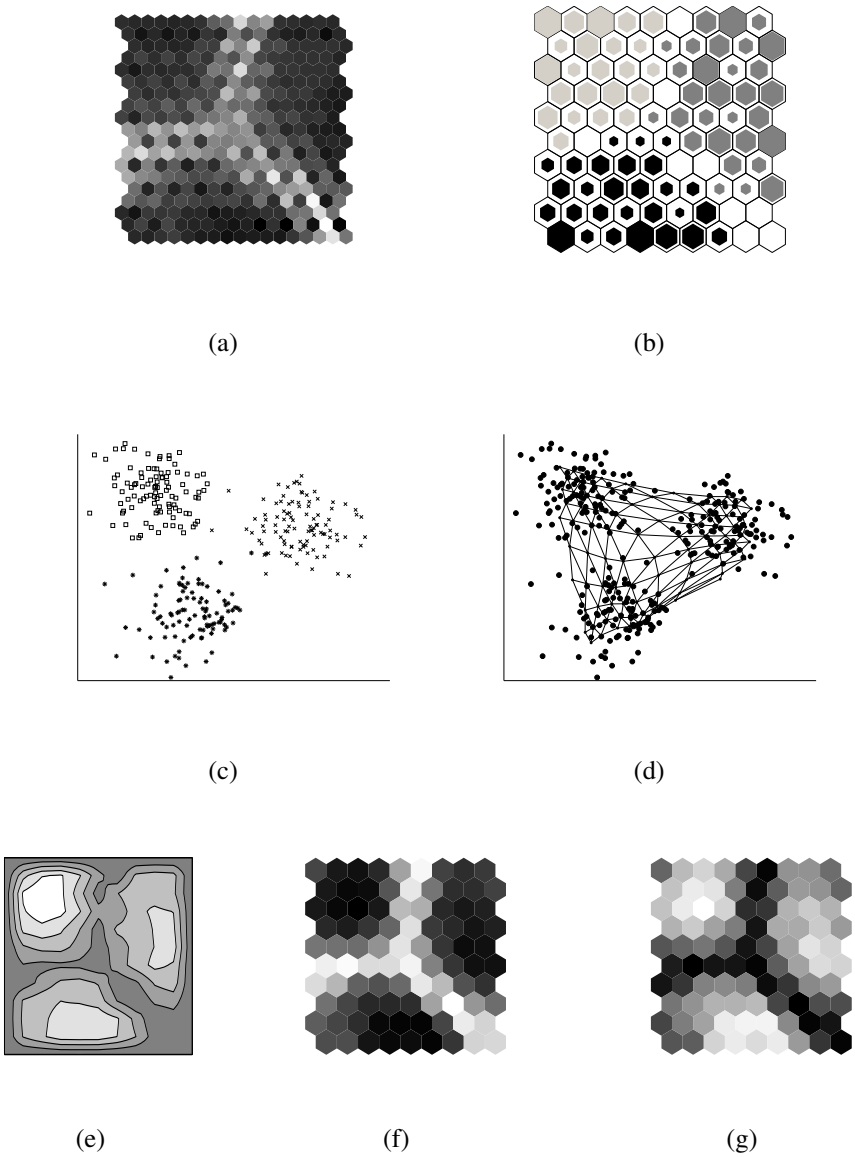


Figure 3.1: Equidistant clusters data set with 3 vertices, $\{9 \times 10\}$ SOM: (a) U-Matrix, (b) hit histogram, (c) PCA projection of the data set, (d) PCA projection of the data set and the codebook vectors, with lines indicating adjacent map units, (e) SDH with $s = 10$, (f) U*-Matrix, (g) P-Matrix

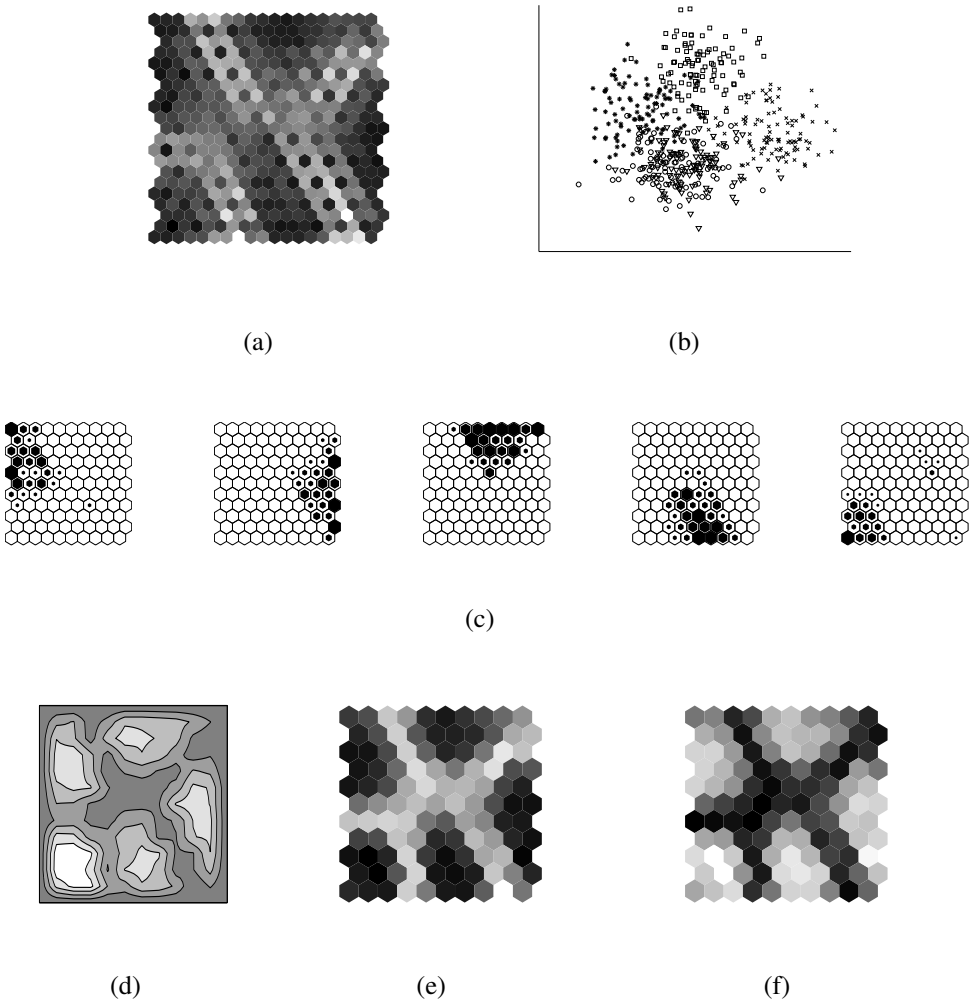


Figure 3.2: Equidistant clusters data set with 5 vertices, $\{10 \times 11\}$ SOM: (a) U-Matrix, (b) PCA projection of the data set, (c) hit histograms for different clusters, (d) SDH with $s = 15$, (e) U*-Matrix, (f) P-Matrix

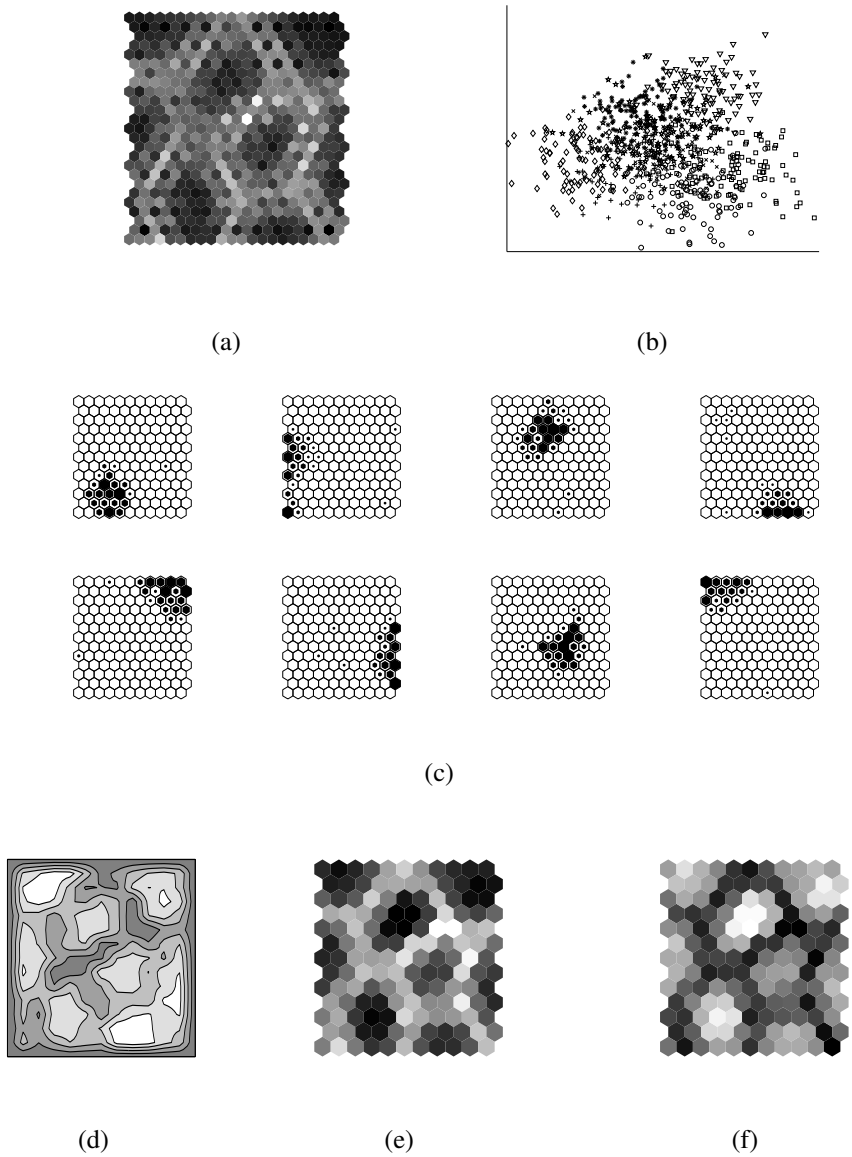
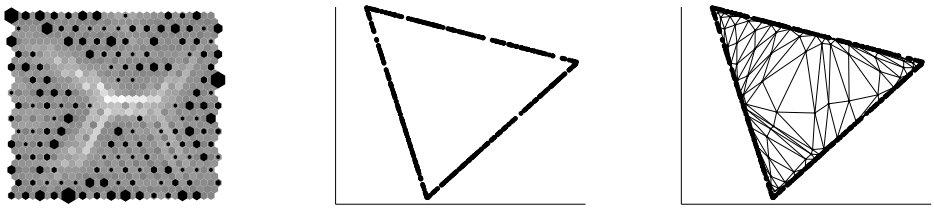
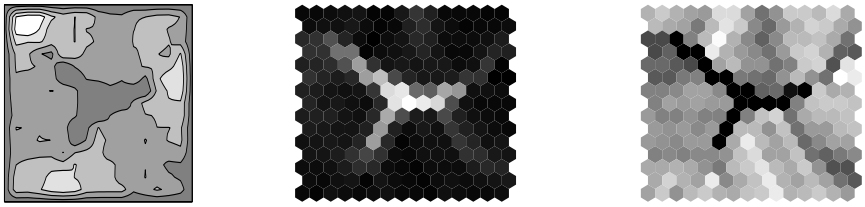


Figure 3.3: Equidistant clusters data set with 8 vertices, $\{11 \times 13\}$ SOM: (a) U-Matrix, (b) PCA projection of the data set, (c) hit histograms for different clusters, (d) SDH with $s = 18$, (e) U*-Matrix, (f) P-Matrix



(a) (b) (c)



(d) (e) (f)

Figure 3.4: Fully-connected data set with 3 vertices, $\{15 \times 15\}$ SOM: (a) U-Matrix and hit histogram, (b) PCA projection of the data set, (c) PCA projection of the data set and the codebook vectors, with lines indicating adjacent map units, (d) SDH with $s = 10$, (e) U*-Matrix, (f) P-Matrix

is projected onto the two-dimensional map. Again, the SOM is good at finding the cluster structure, while PCA is not. The placement of the clusters on the map seems random, as the original distance between the cluster centers is equidistant.

The second artificial data set used in this chapter is the Fully-connected data set. It is defined by the same graph structure as the Equidistant clusters data set, but the data points are distributed along the edges of the graph structure rather than around the vertices. This data set is designed to test the projection qualities of an algorithm. After mapping to the output space, the data points that are close to each other in feature space should be projected to nearby positions in output space. A map unit usually does not represent different regions in input space. While both PCA and the SOM try to ensure that points close in input space are also close in output space, only the SOM also tries to ensure that points close in output space are also close in input space. If, for example, two points lie along the projection direction in the case of PCA, they will be projected onto the same point in output space, even though their distance in feature space may have been

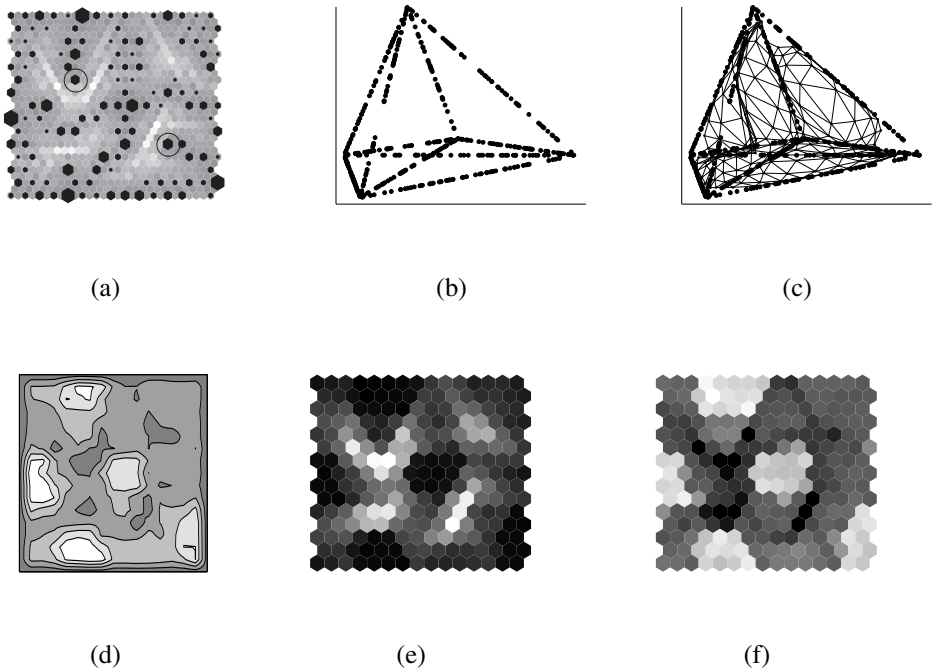


Figure 3.5: Fully-connected data set with 5 vertices, $\{15 \times 15\}$ SOM: (a) U-Matrix and hit histogram, (b) PCA projection of the data set, (c) PCA projection of the data set and the codebook vectors, with lines indicating adjacent map units, (d) SDH with $s = 15$, (e) U*-Matrix, (f) P-Matrix

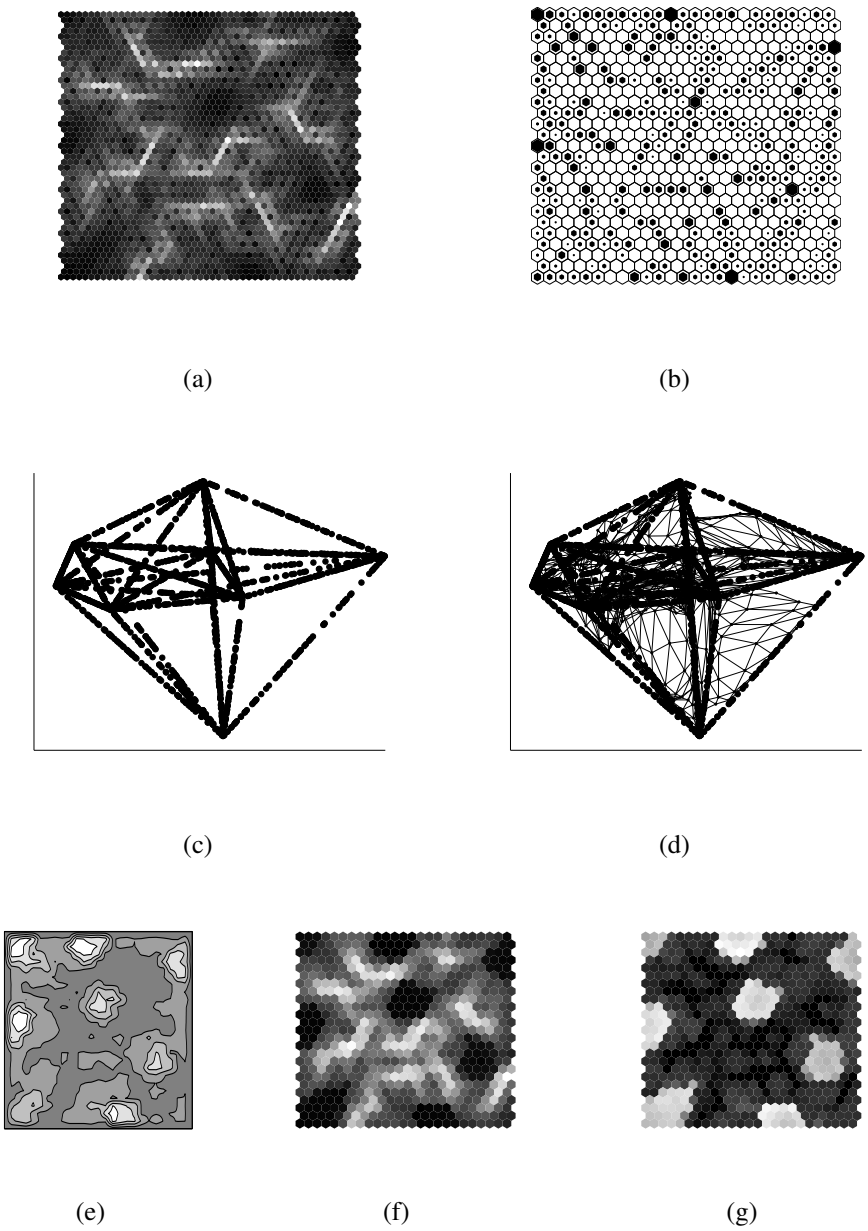


Figure 3.6: Fully-connected data set with 8 vertices, $\{25 \times 25\}$ SOM: (a) U-Matrix, (b) hit histogram, (c) PCA projection of the data set, (d) PCA projection of the data set and the codebook vectors, with lines indicating adjacent map units, (e) SDH with $s = 18$, (f) U*-Matrix, (g) P-Matrix

very high. In case of the SOM, the mutual distances are always minimized on the output space, thus such a situation is very unlikely to occur. This means that the data points that are close on the output map are usually also close in the original feature space, which is not ensured for linear projection algorithms. This feature of the SOM can be seen as similar to a bijective assignment from the part of the feature space where the data is dense to the map lattice. As the data set consists of edges that cannot be depicted on a plane without intersecting, it is especially suited for investigating the effects of topology violations. For details on the construction of these two data sets, please refer to Section D.2.

Examples for the Fully-connected data set are shown in Figures 3.4–3.6 for graphs with 3, 5, and 8 vertices, respectively. The figures contain U-Matrix, hit histogram, PCA projections, SDH, U*-Matrix, and P-Matrix visualizations of the data set and the codebook. In the case with 3 vertices, the graph resembles an equilateral triangle. 500 data points are sampled from this structure. Both the PCA projection and the SOM produce a good representation of this data, as there is no dimensionality reduction involved: A two-dimensional structure is mapped onto a two-dimensional plane. It can be observed from the U-Matrix and the hit histogram that the data points are ordered on the map resembling the original topology, continuously following the borders of the lattice in a circular way. The center part is not populated at all. The P-Matrix also confirms the even distribution across the map. The U*-Matrix is almost identical to the U-Matrix. The SDH shows something very interesting: The vertices of the graph become visible here, due to the fact that even though the points are sampled from the connecting lines with equal probability, the next couple of best matches after the BMU are more likely at the corners. The corners are thus closer to more data samples than to the centers of the lines connecting them. In Figure 3.5, the same visualizations are shown for a graph structure with 5 vertices. The edges are projected as series of neighboring map units that are surrounded by interpolating units. The hit histogram illustrates this, as the edges are placed in corridors with many hits surrounded by gaps with no or few hits and a high U-Matrix value. As the graph structure cannot be depicted in two dimensions without intersecting edges, there are topology violations. These are indicated in Figure 3.5(a) as circles, and denote discontinuities in the projection that result in dead ends and high U-Matrix values. The P-Matrix and SDH visualizations show an interesting feature that is more obvious here than in the case with 3 vertices: The vertices are clearly visible here as regions with high density. A sphere around the center of an edge usually will include only one line segment. If a sphere of the same radius is placed around a vertex, it will include 5 line segments, each with half the length of the ones from the center of the edge, thus increasing the density of points within this sphere by 2.5. As the number of dimensions and vertices is increased, this effect becomes more apparent. In Figure 3.6, a SOM for a Fully-connected data set with

8 vertices is shown, where these topology violations are more frequent. The identification of the violations is fairly easy, as they are dead ends that are visible in the U-Matrix. However, there is no straight-forward remedy to these violations, as it is not easy to tell how the projection could be continued, i.e. how the dead ends could be patched together. This is one of the motivations for introducing the Graph visualization method.

The point of the discussion above is not to criticize either PCA or the SOM, but to highlight the properties of each method. PCA cannot sensibly be expected to solve clustering problems, because PCA simply is not a clustering algorithm. Another goal of introducing the two artificial data sets is to use them as a benchmark to investigate two important characteristics that are present in each general data set to some degree: The Equidistant clusters data set tests for the ability to handle clusters that apparently have no connection to each other, and the Fully-connected data set tests for the ability to handle continuous structures without any clusters. These two concepts, density and connectivity, are opposed to each other, where the former can be seen as a typical quantization problem, and the latter as a typical projection problem.

3.2.2 The Multi-challenge data set

A third artificial data set that is used in this and the next chapters is the Multi-challenge data set. The generation of this data set is outlined in this section, while a detailed discussion is given in the appendix in Section D.2.3. The Multi-challenge data set consists of 5 smaller data sets, each with different characteristics. These subsets are then each normalized individually with zero-mean-unit-variance normalization. Finally, they are placed in a 10-dimensional data space. The data set is shown in Figure 3.7. The data set is generated with 500 samples for each subset, and a $\{40 \times 60\}$ SOM is trained on it. The characteristics of the data subsets are summerized below, while the numbers in Figures 3.7(a) and 3.7(c) refer to the subsets:

- The first subset consists of a Gaussian cluster and another cluster that is itself divided into three Gaussian clusters, all of them living in a three-dimensional space. This subset is used to demonstrate how an algorithm deals with different levels of granularity in cluster structures.

- The second subset is a three-dimensional data set that consists of two overlapping Gaussian clusters. The distribution of points into these clusters is skewed: While the clusters both share the same covariance matrix and thus the same density, the first cluster has twice the amount of points than the second one. The point of introducing this data set is to test how a data analysis method copes with clusters with different numbers of data samples.

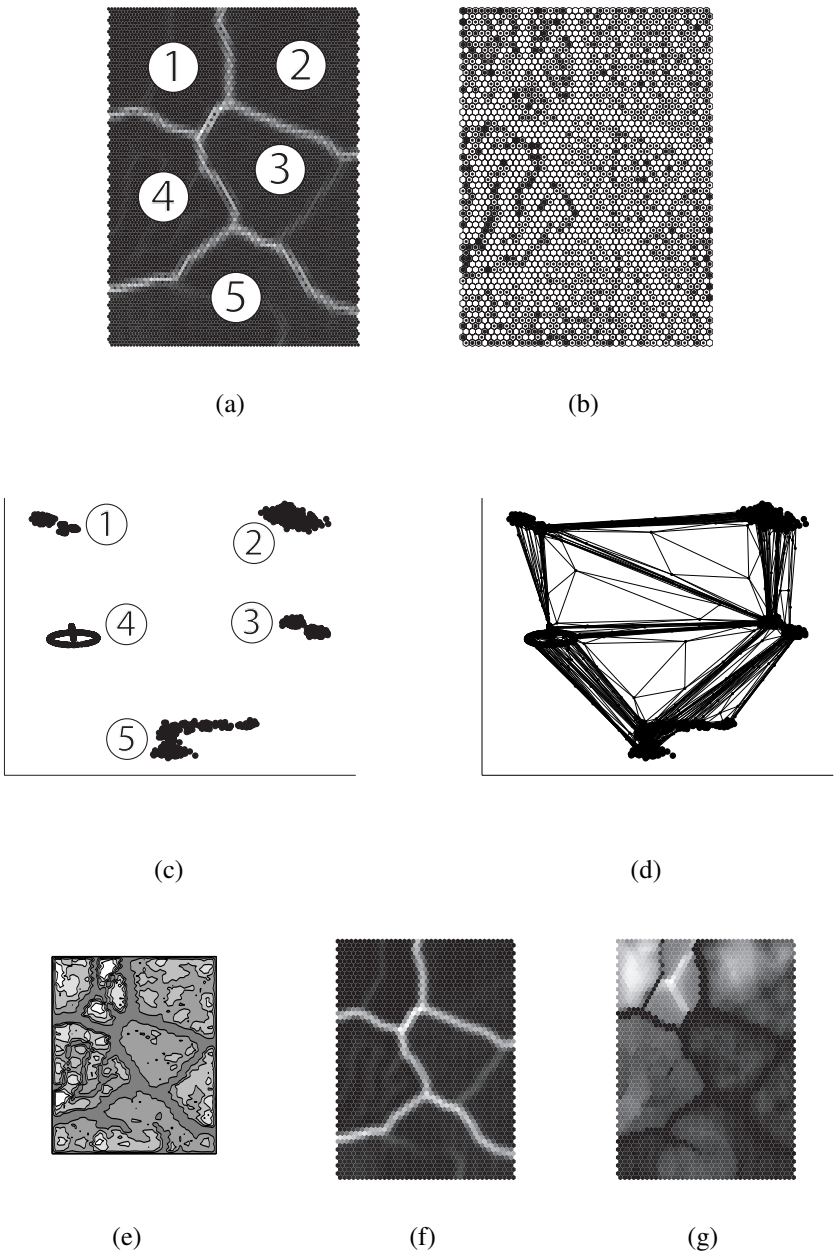


Figure 3.7: Multi-challenge data set, $\{40 \times 60\}$ SOM: (a) U-Matrix, (b) Hit histogram, (c) PCA projection of the data set, (d) PCA projection of the data set and the codebook vectors, with lines indicating adjacent map units, (e) SDH with $s = 15$, (f) U*-Matrix, (g) P-Matrix

- The third subset is a 10-dimensional set of two well-separated Gaussian clusters with the same covariance matrix. It is used to contrast higher- with lower-dimensional structures, as the other four subsets are of lower dimensionality.
- The fourth subset is a classic intertwined rings data set. It lives in a three-dimensional data space. This data set is used for showing how a method deals with non-intersecting structures that cannot be separated in a linear way.
- The fifth subset is sampled along a curve that consists of 4 short lines that are patched together at the endpoints. The data points are arranged along this curve with a level of noise that increases close to the end of the curve. This subset lives in a four-dimensional space. It is introduced in order to show how data analysis methods cope with piecewise linear structures that extend to multiple dimensions.

A PCA projection of the data set is shown in Figure 3.7(c). The projection preserves almost 98% of the variance in the data set. It can be seen that the subsets are placed at considerable distances from each other. The same PCA plot with the structure of the SOM is depicted in Figure 3.7(d), showing how the map folds onto the data cloud. Figure 3.7(a) shows the U-Matrix of the SOM. As gaps between the subsets are high, the U-Matrix cannot find the more fine-grained distances that separate the subclusters. The hit histogram in Figure 3.7(b) shows the distribution and cluster structure of some of the subsets, for example the intertwined rings, which is labeled "4" and is located on the left hand side of the map. Several observations can be made up to this point: Although the number of points in each subset is equal, and all the subsets have been normalized individually, the subsets do not occupy the same amount of space on the map lattice. For example, the high-dimensional clusters in subset "3" is spread across a considerably larger region on the map than subset "1", the cluster-within-clusters subset. This is mainly due to the curse of dimensionality, which describes the phenomenon that pairs of data points sampled out of comparable distributions have higher distances in higher dimensions. As a consequence, the samples from the seemingly less compact cluster in 10 dimension are projected such that the distances are preserved in output space, dominating the distances from the other, more compact subsets. The number of points in cluster "1" is therefor higher, which is also evident from the P-Matrix in Figure 3.7(g), and the SDH visualization with spread $s = 15$ in Figure 3.7(e), where the peak densities all appear in subsets "1", "2" and "4". The U*-Matrix is shown in Figure 3.7(f), showing the smoothed version of the U-Matrix. A slight gap can be identified in the highdimensional subset "3" between the clusters, but the more fine-grained clusters cannot be identified.

3.2.3 The Ionosphere data set

Apart from the artificial data sets, the Ionosphere data set is used in this chapter as the main benchmark data set for demonstration purposes, because its data samples are separated into several dense and sparse areas. Figure 3.8 shows the P-Matrix, U*-Matrix, and the SDH for the Ionosphere data set. The P-Matrix shows that the samples mapped to the upper part of the map are very dense while the lower part of the map covers sparse regions in feature space. The number of data samples is approximately equal in both regions. The U-Matrix confirms the finding that the data sampes in the lower part of the map are less dense and thus farther away from each other, resulting in higher distances between model vectors.

3.3 The Graph method

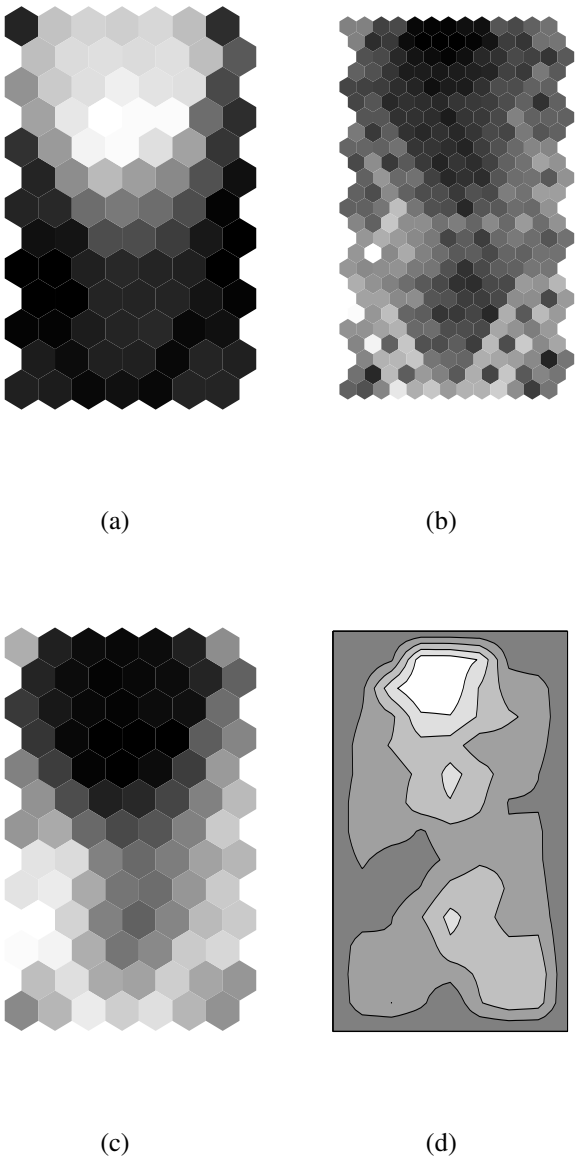
Following the motivation in the previous section, the aim of the Graph method is to show the density of a cluster, and to show which clusters are close to each other in feature space, but possibly far apart on the map lattice due to topology violations. The visualization is computed by defining a density graph structure G in feature space, projecting the graph onto the map lattice, and visualizing the edges such that the projected structure is connected visually by lines.

A definition for the notation of graphs follows: A graph G is a tuple of a set of vertices V and a set of edges $E \subset V \times V$, formally $G = \langle V, E \rangle$. The edges can be represented by a square matrix $\mathbf{E}$, where elements e_{ij} are either 1 or 0 and refer to whether there is an edge between vertices i and j. The elements in the diagonal represent edges connecting a vertex with itself, which is never the case for the graphs in this chapter. If $e_{ij} = e_{ji} \quad \forall i, j$, the matrix is symmetric, and the graph is undirected.

The density graph consists of one node for each data sample, and edges representing whether the data samples are close in feature space. Closeness is a vague term, and in this thesis, two different models are applied, both of which allow for a parameter to interactively control the granularity of the structures analyzed. The first way employs a simple k-nearest neighbors scheme, where edges are introduced from each sample to its k closest peers in feature space. The index of the k-th nearest neighbor of sample $\mathbf{x}_i$ within data set $\mathbf{X}$ is written as $N^{(k)}_{\mathbf{X}}(\mathbf{x}_i)$, following the definitions in Section C.1. Further, the set of k nearest neighbors is denoted as

$$\mathfrak{N}^{(k)}(\mathbf{x}_i) = \bigcup_{j=1}^{k} N^{(j)}_{\mathbf{X}}(\mathbf{x}_i). \tag{3.1}$$

The neighborhood graph contains edges from each sample vector to its k near-



(a) (b)

(c) (d)

Figure 3.8: $\{7 \times 13\}$ Ionosphere SOM density visualizations: (a) P-Matrix, (b) U-Matrix, (c) U*-Matrix, (d) SDH with $s = 2$

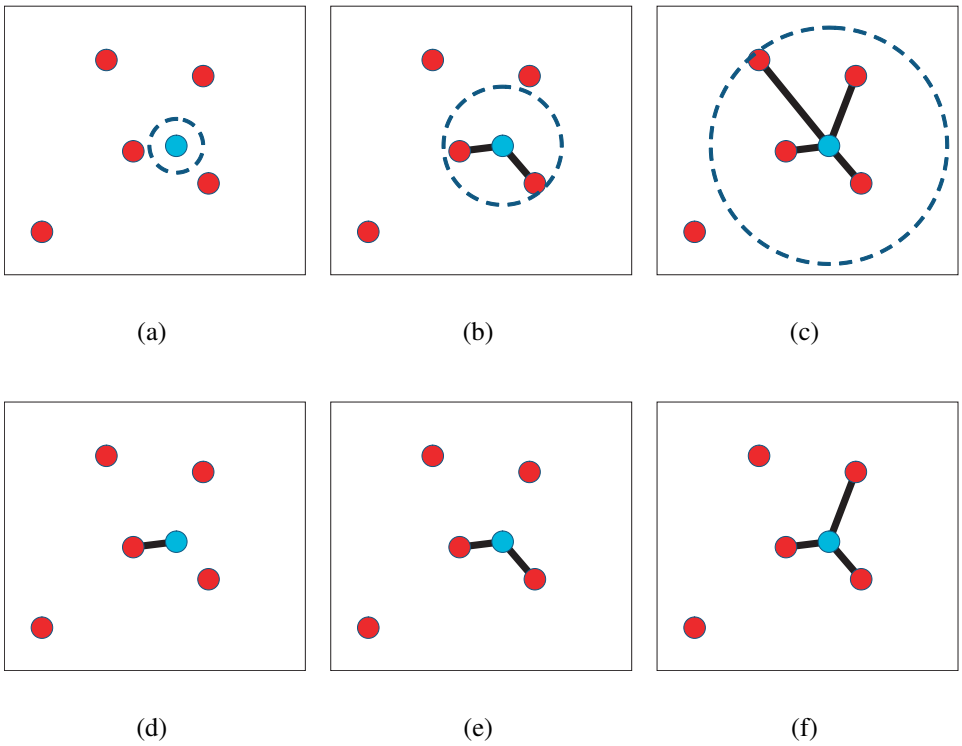


Figure 3.9: Schematic outline of radius and nearest neighbors graph construction: (a)–(c) Graph obtained from radius method with increasing radius, (a) small radius, (b) medium radius, (c) large radius; (d)–(f) Graph obtained from radius method with increasing number of neighbors, (d) 1 neighbor, (e) 2 neighbors, (f) 3 neighbors

est neighbors, constructed by filling the entries of matrix $\mathbf{E}^{\mathrm{NN}}$ with

$$e_{ij}(k) = \begin{cases} 1 & \mathbf{x}_j \in \mathfrak{N}^{(k)}(\mathbf{x}_i) \vee \mathbf{x}_i \in \mathfrak{N}^{(k)}(\mathbf{x}_j) \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

Another possibility to construct this graph is to connect samples with a distance below a threshold value r, which is equivalent to the samples that lie within the hypersphere of radius r around a sample. The edges $\mathbf{E}^{\mathrm{rad}}$ are defined by

$$e_{ij}(r) = \begin{cases} 1 & \|\mathbf{x}_i - \mathbf{x}_j\| < r \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

A schematic outline of how this graph is calculated is shown in Figure3.9. Figures 3.9(a)–(c) show how the graph is constructed with the radius method at

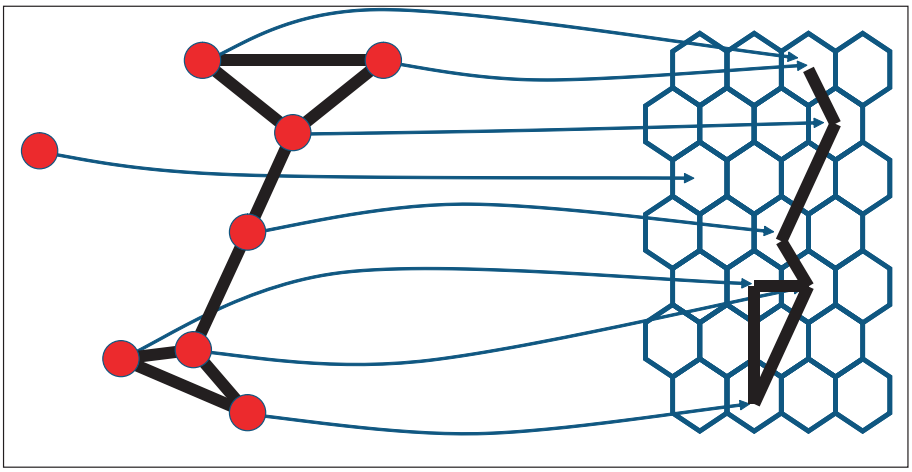


Figure 3.10: Schematic outline of graph projection: The data points are shown in feature space on the left, along with a graph structure that represents their proximity; the data points are projected onto the map, the assignment of the data points to the map nodes is denoted by the thin arrows; finally, the graph is reconstructed in output space by connecting the corresponding winner units

various levels of r. Figures 3.9(d)–(f) depict the construction of the nearest neighbors graph.

The resulting graphs, defined in either of these ways, represent a concept of proximity between the sample vectors. The parameter k or r determines the density or sparsity of the graph. The next step in the computation of the visualization method is projecting the data set onto the map, which just means computing the BMU for each sample. The graph structures can also be projected, now between the prototype vectors as vertices $\hat{V}$. Edges are present between BMUs of two samples that have been connected in input space. Ideally, the BMUs of connected samples should either coincide or be close to each other on the map. Proximity in feature space, as described by the graph structure, should be preserved after the mapping. The adjacency matrix of the graph in output space can be computed as

$$\hat{e}_{ij} = \begin{cases} 1 & \text{if } i \neq j \wedge \exists x_k, x_l : e_{kl} = 1 \wedge m_i = I(x_k) \wedge m_j = I(x_l) \\ 0 & \text{otherwise} \end{cases} \tag{3.4}$$

where $\hat{e}_{ij}$ stands for the feature space graphs, either $e_{ij}(r)$ or $e_{ij}(k)$. From the definition above, the graph does not have connections from nodes to themselves, even if connected samples are mapped to the same node. Further, the output space graph $\hat{G}$ has at most as many edges as the feature space graph G, regardless of whether there are more samples or map units.

This projection approach is outlined in Figure 3.10. The left hand side denotes the data samples in feature space and the right hand side denotes output space. The graph structure is projected by first projecting the data points and then connecting their winner units according to the original graph in feature space.

The output space graph can be visualized at this stage. This is performed by simply plotting a line between map patches for which an edge exists. Examples are given in the next section.

In order to produce comprehensible visulizations that do not contain too many lines, the parameters have to be adjusted. Concerning the choice of the radius r, experimental results have shown that it should lie within the 2–5% quantile of all inter-data sample distances. If the number of samples is high, higher radii should be chosen. The choice of the number of nearest neighbors is more difficult. Experiments have shown that values for k in the range of 1 to 25 produce good results.

3.4 Experiments

In this section, experimental results are presented on the artificial and benchmark data sets.

3.4.1 Experiments on artificial data sets

The first set of examples is based on the Euqidistant data set. It is investigated with varying numbers of vertices, which is depicted in Figures 3.11–3.13. These data sets consist of a number of Gaussian clusters around a set of points that have a constant mutual distance from each other. In each of these figures, a hit histogram is shown where the markers are colored corresponing to the class labels.

Figure 3.11 shows the Equidistant clusters data set with 3 clusters. This can be mapped without distorting the inter-cluster distances. The clusters are clearly visible with the U-Matrix. In the settings with a low number of neighbors or a low radius, which can be observed in Figures 3.11(b) and 3.11(e), the points within the clusters are connected. This is also the most basic use of the Graph visualization method, as it can be used for cluster identification. As these levels are increased, as in Figures 3.11(d) and 3.11(g), the boundaries between the clusters are crossed by various lines. As the clusters are all equally far apart, they should be equally strongly connected by the lines. This is the effect that is being investigated with the Equidistant clusters data set, especially in higher dimensions where the clusters cannot be mapped such that they are all equally far apart on the output space.

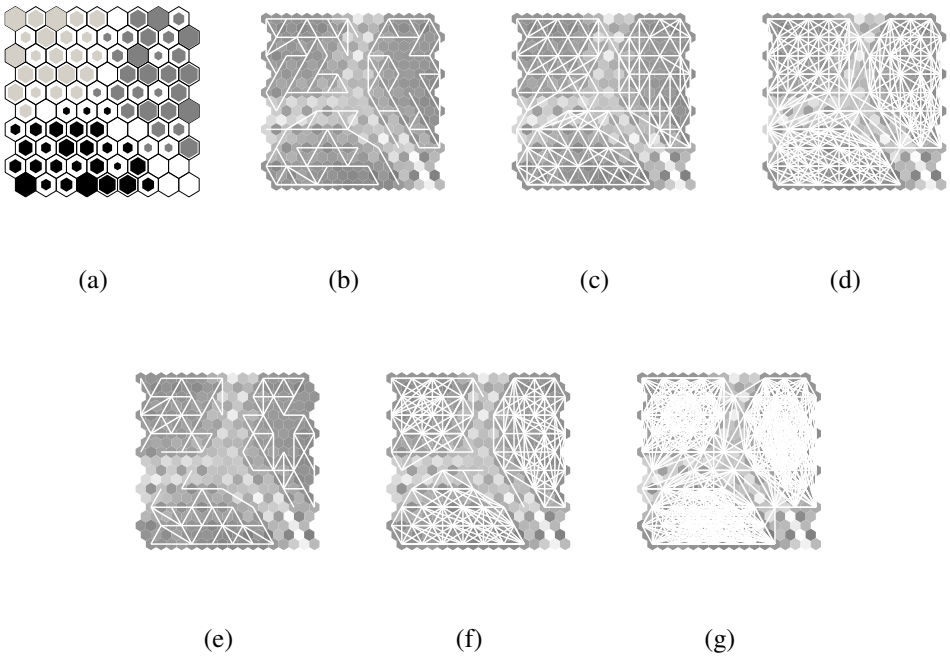


Figure 3.11: Equidistant clusters data set with 3 vertices, $\{9 \times 10\}$ SOM, (a) hit histogram colored according to class labels, (b)–(d) nearest neighbors method: (b) $k = 3$, (c) $k = 10$, (d) $k = 15$; (e)–(g) radius method: (e) $r = 0.2$, (f) $r = 0.4$, (g) $r = 0.8$

In Figure 3.12, the number of clusters is increased to 5. The nearest neighbors method does not produce usable results anymore, which is due to the connectivity of the clusters that overloads the visualization with short lines. The clusters are highly dense, but there is no connectivity between the clusters. The radius method shows the clusters as densely connected areas, with an increasing number of cross-connections as the threshold r increases. For the data set with 8 clusters, as depicted in Figure 3.13, the results are similar. The clusters can be clearly identified, and the connections between the clusters increase as the radius is increased. The connections between the clusters seem arbitrary, i.e. the lines connect clusters regardless of the map distance. This is an intended result, as the clusters are equidistant and their placement on the output space is almost random.

In the next set of examples, the Fully-connected data set is investigated. As demonstrated previously in Figures 3.4–3.6, the SOM mapping is unable to cope with such complex high-dimensional structures without introducing topology vi-

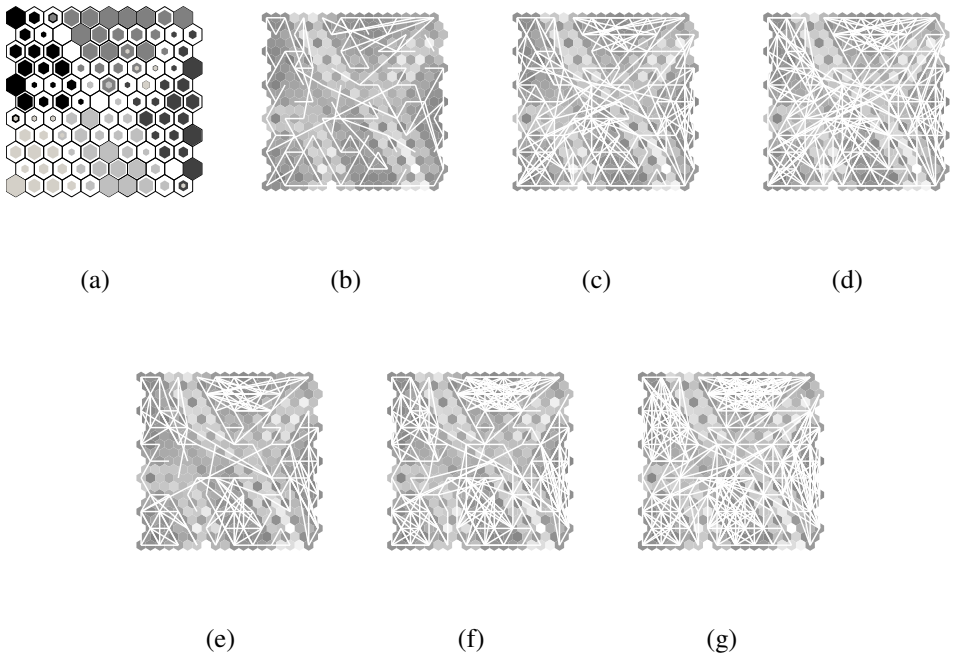


Figure 3.12: Equidistant clusters data set with 5 vertices, $\{10 \times 11\}$ SOM, (a) hit histogram colored according to class labels, (b)–(d) nearest neighbors method: (b) $k = 1$, (c) $k = 3$, (d) $k = 5$; (e)–(g) radius method: (e) $r = 0.6$, (f) $r = 0.7$, (g) $r = 0.8$

olations. Figures 3.14–3.16 shows how the Graph method is applied onto maps trained on the Fully-connected data set with various parameterizations. The figures correspond to either 3-, 5-, or 8-vertex settings, and the subfigures show the Graph based method with the nearest neighbors- and radius-based methods at different levels. The lines of the Graph visualization are plotted on top of an U-Matrix visualization. Again, a hit histogram where the labels are colored according to class membership is included for all three data sets. The classes of the data points are attributed according to the label of their closest vertex.

In Figure 3.14(b), the output space for a toy example with 3 vertices is shown, which resembles an equilateral triangle in feature space. In this first example, the 3 nearest neighbors are connected. It can be seen that only those nodes are connected that lie on a dark region in the U-Matrix, indicating an area that is close in feature space. The lines usually run from the edges to the center, hinting at a distortion in these directions. For example, the left border has lines running horizontally up to the triangular delimiting shape with high U-Matrix values. The

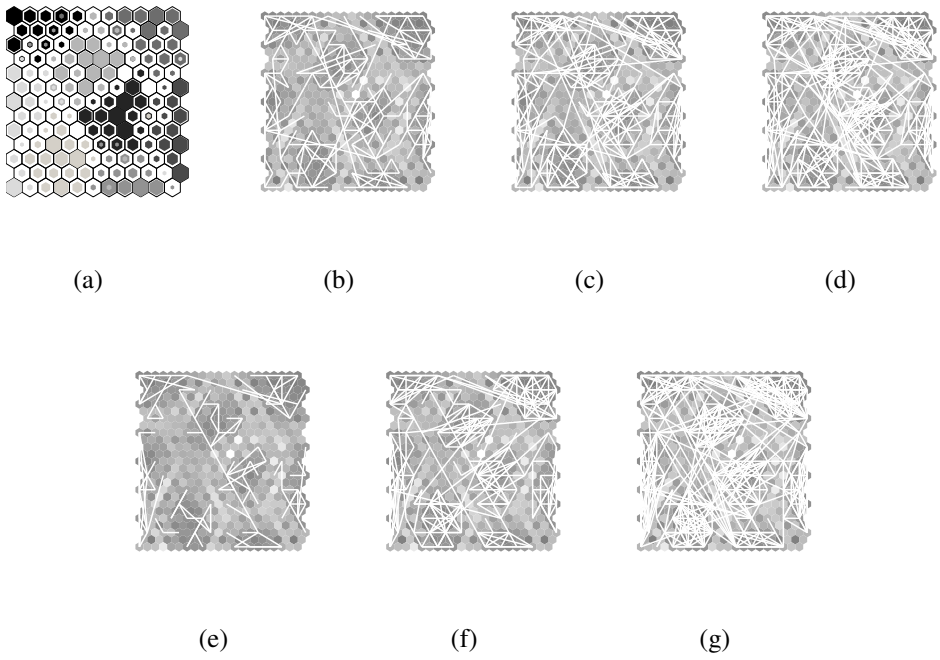
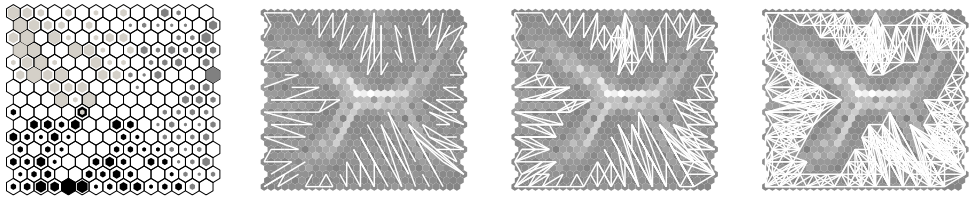


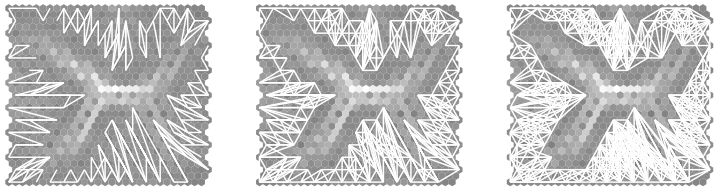
Figure 3.13: Equidistant clusters data set with 8 vertices, $\{11 \times 13\}$ SOM, (a) hit histogram colored according to class labels, (b)–(d) nearest neighbors method: (b) $k = 1$, (c) $k = 2$, (d) $k = 3$; (e)–(g) radius method: (e) $r = 1.1$, (f) $r = 1.3$, (g) $r = 1.5$

cause for this is that the original shape in feature space consists of a set of lines only, but now has to fill an entire plane. The shape is projected onto the border of the two-dimensional output space, which matches the topology of the triangle. But the area within the rectangle of the output space also has to be filled, thus the data is distributed in a way that similar samples are distributed between the borders and the center, creating the pattern that can be observed in Figure 3.14(b).

By increasing the number of neighbors that are considered, more pairs of data samples are classified as topology violating, resulting in more lines. Figures 3.14(c),(d) show graphs with 10 and 25 nearest neighbor graphs, respectively. The resulting visualizations reveal that the topology violations are restricted to the phenomenons described in the previous paragraph. There are no further violations. The highly connected areas especially in the middle of the upper and lower parts of the map are very close in feature space, which is also confirmed by the U-Matrix. One of the fundamental differences between the U-Matrix and the Graph method is that the U-Matrix considers solely the output space, while the



(a) (b) (c) (d)



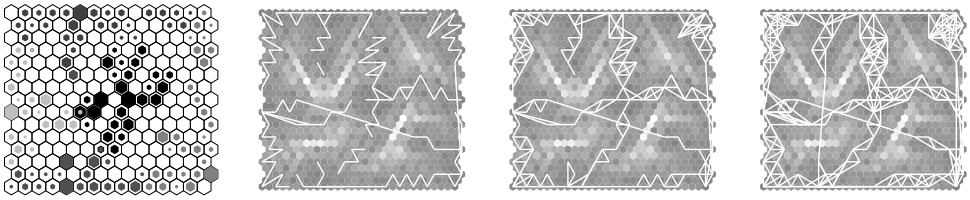
(e) (f) (g)

Figure 3.14: Fully-connected data set with 3 vertices, $\{15 \times 15\}$ SOM, (a) hit histogram colored according to class labels, (b)–(d) nearest neighbors method: (b) $k = 3$, (c) $k = 10$, (d) $k = 25$; (e)–(g) radius method: (e) $r = 0.1$, (f) $r = 0.2$, (g) $r = 0.3$

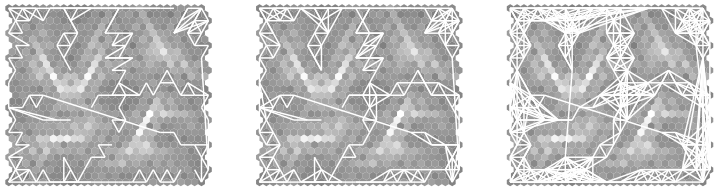
Graph method reveals topology violations induced by the distribution of the data cloud in feature space. Figures 3.14(e)–(g) show the radius based Graphs at levels 0.1, 0.2, and 0.3, respectively. The results are very similar to the nearest neighbors Graphs in this case. The differences between the two methods are highlighted in the next set of examples.

Figure 3.15 shows the Graph visualizations for the Fully-connected data set with 5 vertices. As highlighted previously in Figure 3.5(a), there are identifyable topology violations. The continuation of these dead ends is performed by the Graph based method. The figures, especially the ones with higher parameter levels for the radius or the number of neighbors, show where the original data shape has been broken and which parts fit together.

In Figure 3.16, the visualization of the 8 vertex Fully-connected data set is too overloaded to be comprehensible. There are lots of topology violations, and the picture seems chaotic. What can be seen, however, is that the lenght of the connections is high on average for the nearest neighbors visualization, indicating



(a) (b) (c) (d)



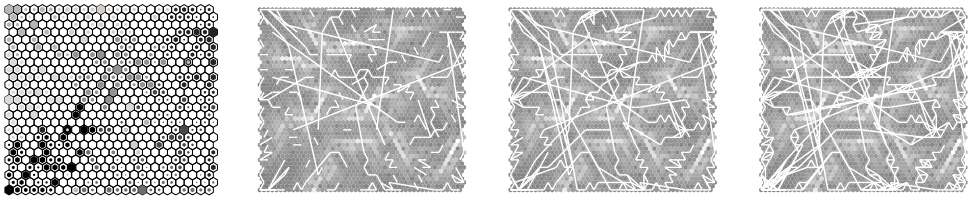
(e) (f) (g)

Figure 3.15: Fully-connected data set with 5 vertices, $\{15 \times 15\}$ SOM, (a) hit histogram colored according to class labels, (b)–(d)nearest neighbors method: (b) $k = 4$, (c) $k = 10$, (d) $k = 15$; (e)–(g) radius method: (e) $r = 0.35$, (f) $r = 0.5$, (g) $r = 0.85$

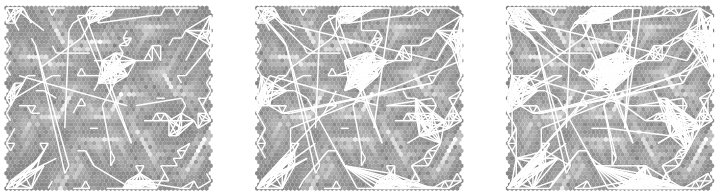
many violations. The output space is not of high enough dimensionality to capture the data structure. The radius method differs from the nearest neighbors method as it shows cluster-like shapes. This is due to the increased density close to the vertices of the Fully-connected data set, which becomes obvious as the dimension is increased. As there are 7 edges connecting the vertices, the number of data samples around the vertices is higher than in the middle of the edges. The radius method unveils these dense areas, but obscures the connectivity of these clusters, when compared to the nearest neighbors method.

Finally, the last of the artificial data sets is investigated, the Multi-challenge data set. The results for the Graph method with both the nearest neighbors and radius methods are given in Figure 3.17. It has been constructed using 500 data samples for each subset, resulting in a total of 4500 samples for the whole data set. The map is trained with a $\{40 \times 60\}$ topology.

The nearest neighbors method is evaluated at levels $k = 1$, 3, 5, and 10, which



(a) (b) (c) (d)



(e) (f) (g)

Figure 3.16: Fully-connected data set with 8 vertices, $\{25 \times 25\}$ SOM, (a) hit histogram colored according to class labels, (b)–(d) nearest neighbors method: (b) $k = 2$, (c) $k = 5$, (d) $k = 10$; (e)–(g) radius method: (e) $r = 0.6$, (f) $r = 0.8$, (g) $r = 1.0$

is shown in Figures 3.17(a)–(d). The different subsets have to be discussed separately. The clusters of the cluster-of-clusters subset in the upper left corner are joined by grouping the three small clusters and the remaining big cluster. Although the number of nearest neighbors is increased, and consequently the number of lines drawn, the 3 small clusters are never joined. The second subset is located at the upper right corner, and is the 3-dimensional overlapping Gaussians where one cluster has twice as many points as the other one. The cluster with the higher number of samples is on the upper left, the other one at the lower right. As the clusters are actually very close in feature space, the separation is barely visible. The third subset consists of well-separated 10-dimensional Gaussians and can be found at the right hand side and the center. The clusters become visible at fairly low parameter values. This case is very similar to the Equidistant clusters data set. The intertwined rings data set on the center of the left border shows the connectivity of the two rings. Topology violations are present as the rings cannot be projected to a two-dimensional plane without breaking one of the rings.

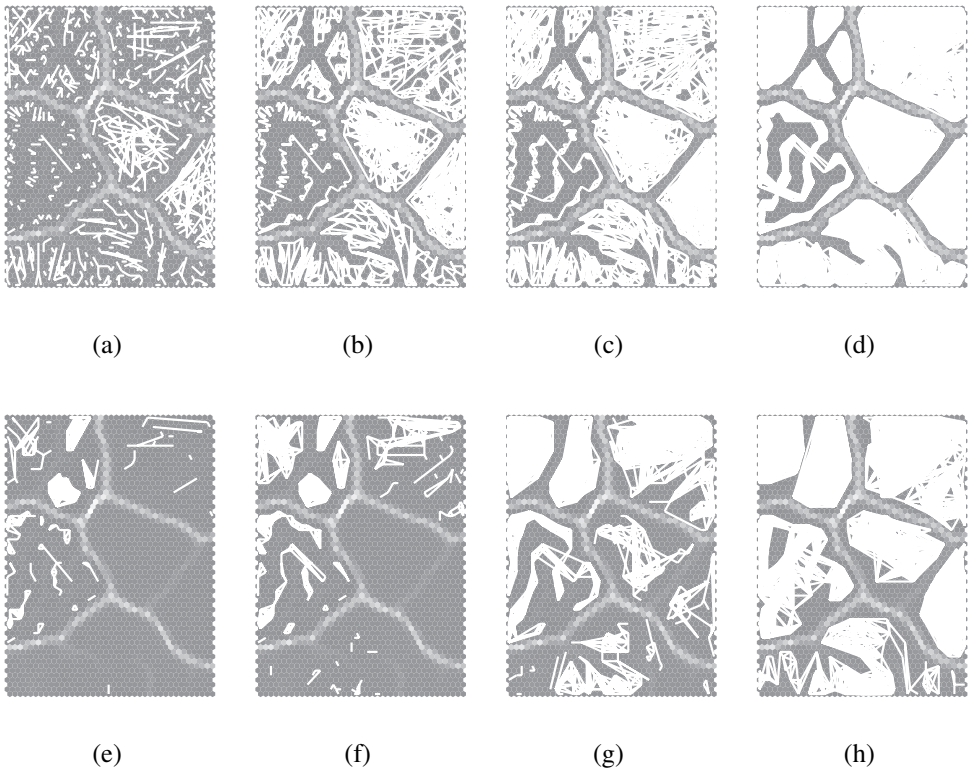


Figure 3.17: Multi-challenge data set, $\{40 \times 60\}$ SOM, (a)–(d) nearest neighbors method: (a) $k = 1$, (b) $k = 3$, (c) $k = 5$, (d) $k = 10$; (e)–(h) radius method: (e) $r = 0.3$, (f) $r = 0.5$, (g) $r = 1.0$, (h) $r = 1.5$

This break can be observed in Figure 3.7(b), where the hit histogram shows two U-shaped structures, where the endpoints should connect in order to complete the circles. The fifth subset on the bottom of the figure represents a sample along curve. This curve consists of line segments that are joined at the endpoints. The nearest neighbors method finds this structure.

To sum up, in the projection of these subsets, using nearest neighbors, no topology violations except for the intertwined rings are apparent, as the lines do not reach across large regions. It reveals the three subclusters of the cluster-within-clusters subset, but is not able to join them at reasonable levels on k. Further increasing the number of nearest neighbors would result in an overloading of the figure with lines. Linear and circular structures can easily be identified by the nearest neighbors method. Another finding is that the boundaries between the subsets have never been crossed.

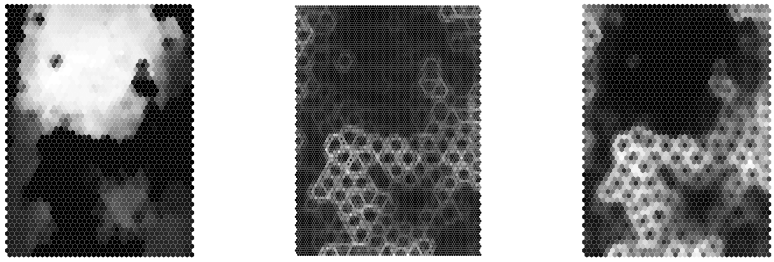
The results with the radius method are shown in Figures 3.17(e)–(h) for 0.3, 0.5, 1.0, and 1.5. The most obvious difference to the nearest neighbors method is that the lines are not evenly distributed among the subsets in the case of the radius method. Although each subset is individually normalized before the subsets are placed in the same final feature space, the density is not equal across the whole map. This is due to the curse of dimensionality and affects the density and distribution of the data across the map, and can be seen in the P-Matrix visualization in Figure 3.7(g). As the radius method is a technique for identifying differences in density, the dense areas such as the cluster-of-clusters subset are occupied by lines at low threshold levels.

Looking at the subsets individually, applying the radius method to the cluster-of-clusters data set shows the behavior of first joining each of the 3 small compact clusters, then the larger cluster, and finally joining the small clusters to form the second big cluster. With the second subset in the upper right, it is now easier to identify the two Gaussians when compared to the nearest neighbors method. The third subset on the right is connected only at high threshold levels, then showing the two clusters. The connections for the intertwined rings subset are slightly different from the nearest neighbors method as the two rings tend to be joined where they are close in feature space, and the lines do not just follow the circular structure. Finally, the piecewise linear structure at the bottom of the map is not identified as clearly as with the nearest neighbors method. This is due to the increased density at the inner line segments and the increased level of noise on the outer line segments.

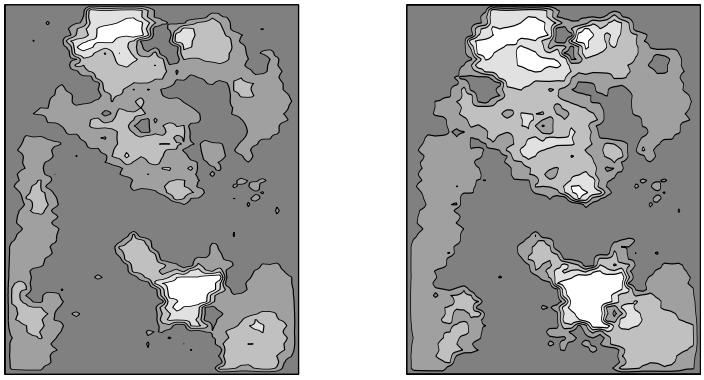
To summarize the empirical findings from the artificial examples, the nearest neighbors method is better suited for the Fully-connected data set, while the radius method achieved better results for the Equidistant clusters data. It has been demonstrated that the nearest neighbors method is suitable at uncovering the connectivity structure of a data set, while the radius method is suitable for identifying clusters. The limitations of the Graph visualization are approached in case of a very high number of topology violations, as demonstrated with the Fully-connected data with 8 vertices. Regarding the results from the Multi-challenge data set, the findings from the Equidistant and Fully-connected data sets can be reaffirmed, as the Gaussian clusters are identified better by the radius method, and the nearest neighbors method performs better with the linear and circular structures, identifying connectivity.

3.4.2 Experiments on benchmark data sets

In this section, experimental results on benchmark data sets are presented. The Ionosphere data set is shown in Figure 3.18, where standard density visualization methods are given for a $\{40 \times 60\}$ map. Figures 3.19(b)–(d) show the near-



(a) (b) (c)



(d) (e)

Figure 3.18: Large $\{40 \times 60\}$ Ionosphere SOM density visualizations: (a) P-Matrix, (b) U-Matrix, (c) U*-Matrix, (d) SDH with $s = 100$, (e) SDH with $s = 200$

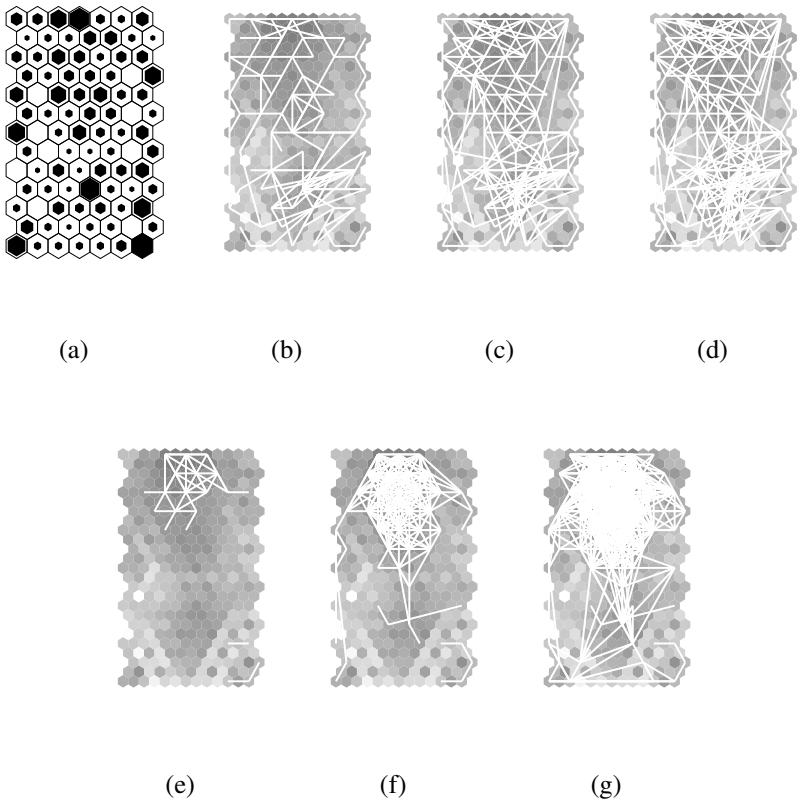
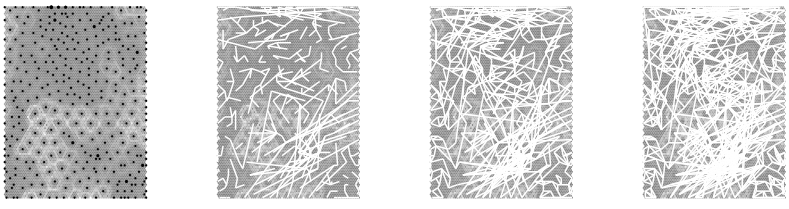


Figure 3.19: $\{7 \times 13\}$ Ionosphere SOM: (a) hit histogram; (b)–(d) Nearest neighbors method: (b) $k = 1$, (c) $k = 2$, (d) $k = 3$; (e)–(g) Radius method: (e) $r = 1.0$, (f) $r = 2.0$, (g) $r = 3.0$

est neighbors approach for three different numbers of neighbors, and 3.19(e)–(g) show the radius method at different threshold levels for a $\{7 \times 13\}$ SOM. The distribution of the data without any graph visualization can be seen in the hit histogram in Figure 3.19(a). The samples are distributed almost evenly. The graph based visualizations are plotted on top of the U-Matrix, which hints at a homogeneous region in the upper half of the map.

The first level of the radius visualization in Figure 3.19(e) shows the visualization of the threshold $r = 1.0$. At this level, only the most dense areas show lines at all, which are mostly connections between neighboring map units. It is thus likely that the samples mapped to the upper part of the map are also dense in feature space and the samples projected to the lower part are not mutually connected, since no lines are shown. The observation of different densities can also be



(a) (b) (c) (d)

(e) (f) (g)

Figure 3.20: Large $\{40 \times 60\}$ Ionosphere SOM: (a) hit histogram; (b)–(d) Nearest neighbors method: (b) $k = 1$, (c) $k = 2$, (d) $k = 3$; (e)–(g) Radius method: (e) $r = 1.0$, (f) $r = 2.0$, (g) $r = 3.0$

made with the P-Matrix visualization in Figures 3.8(a), which clearly shows that the upper part's density values dominate. When the radius for the Graph method is increased, as shown in Figures 3.19(f),(g), the graphs get increasingly stronger connected. At these levels, longer lines become visible that connect areas that are as far as half the map diagonal apart, which hints at topology violations, similar to the visualization of the topographic error, as described in Section 2.3.4. The nearest neighbors method, as depicted in Figures 3.19(b)–(d), does not yield good results. Again, as with the artificial samples in the previous section, the distribution of the Ionosphere data set is mostly interesting because of its clusters and not because of the shape of the data set, i.e. the connectivity of its data points. The radius method is more suited at unveiling this cluster information than the nearest neighbors method.

Figure 3.20 shows the Graph visualization for a larger SOM trained on the Ionosphere data with a topology of $\{40 \times 60\}$. Here, the number of data samples is smaller than the number of map units. Again, the radius method highlights the higher density in the upper part of the map. Additionally, the small clusters in the

lower part of the map are set into relation with connections at higher threshold parameter levels. The nearest neighbors method, however, still does not provide any usable information.

The Graph method applied to the Iris data set is displayed in Figure 3.21. Both the small and the large version of the map trained on the Iris data show that the upper third part, which corresponds to the Setosa species, is densely occupied, and the lower part is slightly less dense. There are no connections between these two region, indicating a strong separation between these areas. The lower parts of the smaller version of the maps, especially Figure 3.21(d), show that there is a systematic topology violation in the diagonal direction from the lower left to the upper right.

3.5 Systematic analysis and guidelines

In this section, the findings of the previous sections are analyzed to the point of creating guidelines for utilizing the Graph method as a data mining tool in combination with other SOM visualizations. Three aspects of data analysis are presented that can be investigated with the Graph method: Data density, connectivity, and topology violations. The steps for analyzing a data set for the given criterion are shown, and the use of Graph methods as a complementary means is shown.

3.5.1 Investigating density and clustering structure

In this section, the density and clustering structure of a data set is analyzed. The most common visulization method, the U-Matrix, is the first step in analyzing the clustering structure. It is shown in Figure 3.22(a). In this case, it reveals that there are five regions and large gaps between them. Together with the hit histogram in Figure 3.22(b), the gaps can be identified as interpolating units that do not represent any points in the original data set. The hit histogram further reveals that the data points are not evenly distributed across the map, as region "3" is more sparsely populated than the others.

Next, the density of the SOM is investigated with the P-Matrix, as shown in Figure 3.22(d). It can be observed, that some of the five regions are more dense than others, especially subset "1". The SDH in Figure 3.22(c) also confirms this finding.

Using the radius method and iteratively increasing the threshold r as shown in Figures 3.22(e)–(i), the clustering structure becomes apparent in an intuitive way. Those areas that are filled with lines first are the most dense parts of the map. After some steps, the map starts being filled with lines and eventually reaches the least dense areas. As the radius is increased, the other regions are occupied with lines

according to their density. The three small subclusters are joined individually at first in Figure 3.22(e); next, the big cluster to the left in Figure 3.22(f); then, the three subclusters are joined in Figure 3.22(g).

Another aspect concerning clusters is how they relate to each other. This is especially of concern if clusters that are not necessarily very close, but also not very distant, are projected to distant regions of the map. An example of such a constellation is shown for the Ionosphere SOM in Figures 3.23(a)–(d), which show the U- and P-Matrices, and Ward's linkage at different levels. A series of Graph visualization using the radius method is given in Figures 3.23(e)–(g). While what can be learned from the Graph method is similar to what can be observed from hierarchical clustering, the Graph method also shows how dense the clusters are and not just whether there are clusters or not. Regions that consist of fully-connected graph strucures are very dense clusters. Also, the Graph method provides insights into the inter-cluster relations, as the number of lines between clusters tells about their similarity. The dense area in the upper half of the map is connected to the seemingly unrelated cluster in the lower right part of the map, which is something that cannot be seen in the visualizations of Ward's linkage.

The guidelines for using the Graph method to investigate density and clustering structure are summarized here:

- Generally, the radius method is better suited than the nearest neighbor method to unveil density and clustering structure.
- The radius method is best used by starting with a small threshold r and then increasing it. In this way, the most dense structures will appear first, and then the less dense areas will be connected.
- Clustering methods and the U-Matrix visualization usually yield very deep insights into the clusters that are present in the data set. But clusters are not necessarily projected to adjacent regions on the map. By increasing the radius beyond a level where basic clusters are revealed, the relations of the clusters to each other and their hierarchy can be revealed. The results can be compared to hierarchical clustering methods like Ward's linkage.
- While the P-Matrix gives a first impression about the density of points around each map node, and the U-Matrix, hit histogram and clustering visualizations tell about the clustering structure of the map, the Graph method can be used to combine these pieces of informations, using the radius method and plotting a series with increasing radii.

3.5.2 Investigating connectivity and topology violations

In this section, the topology aspect of the SOM projection is investigated, both in terms of topology violations and connectivity. Topology violations occur when data points that are close in feature space are not projected next to each other in output space due to the mismatch between the dimensionality of feature and output space. Connectivity refers to low-dimensional manifolds that may be hidden in the data cloud, such as points along a curve that is folded into the feature space.

One of the most straight-forward methods to check for topology violations is the Topographic error, which is shown in Figure 3.24(b). The violations are depicted as lines connecting the two best matching units of a sample in case these two units are not adjacent. As the lines shown with the Topographic error are very short, only few violations are expected. The longer lines are mostly located in subsets "3" and "5", which are of higher dimension than the rest. Topology violations are more likely to happen in higher dimensional data sets, as the loss of dimensionality is higher when projected to a two-dimensional map.

Figures 3.24(a)–(c) show the 3 components of the SOM Distortion as described in Section 2.3.4. The quantization error in Figure 3.24(b) amounts to only 1.4% of the total error, so its influence is neglegible. The quantization error is highest where the higher-dimensional subsets are located. The neighborhood bias and neighborhood variance terms, which account for 17.7% and 80.9%, respectively, are very similar and are shown in Figures 3.24(c),(d). The areas close to the transitions between the subsets are responsible for most of the distortion in this map. This is not surprising, due to the large gaps between the subsets which overshadow the more fine-grained distortions of the map. As a result, the SOM Distortion does not yield any new information on topology violations. The Topographic Function cannot be computed due to the requirement of roughly 60 times the number of data samples per map unit, which cannot be fulfilled with this data set and map size.

The Topographic Product is shown in Figure 3.24(e). It depicts low (dark) values where the dimensionality of the output space is too low to accomodate the data from the feature space. For the hight (light) values, the output space is too high-dimensional for the data in feature space. The figure, which is computed solely from the codebook, disregarding the data samples, shows that the interpolating units are part of the too low-dimensional parts of the output space. The units close to these borders have the highest values. The problem with this visualization is that it is not clear what the interpolating units have to do with topology violations at all, so the figure does not help to reveal the actual violations. Furthermore, it is not clear why the highest dimensional subset, the 10-dimensional overlapping Gaussians in subset "3", has higher values when compared to lower-dimensional, otherwise similar subsets "1" and "2".

The nearest neighbors method is shown in Figures 3.24(f)–(i) at various levels of k. Beginning at $k = 3$, the clustering structure becomes visible for the Gaussian clusters ("1", "2", "3"). Other than the radius method, the nearest neighbors method joins clusters regardless of density, i.e. the dense clusters in "1" are not joined before the relatively sparse clusters in "3". Furthermore, the topology violation in subset "4" are visible as relatively long lines crossing the regions occupied by the other ring. At $k = 5$, the connectivity of the manifolds formed by the data points are revealed: The circles in subset "4" are visible as well as the curve in subset "5". No other visualization method is able to reveal the structure of the manifold.

- Like the radius method, the nearest neighbors method is best used as a series of visualizations at increasing parameter values.
- Universally usable methods for revealing topology violations are scarce. The Topographic Error is a simple technique of gaining a first impression of such violations. It can be complemented with the more sophisticated nearest neighbor method.
- When there are lines connecting regions that are far apart in the neighborhood method visualization, this is caused by topology violations.
- The nearest neighbors method does not reveal density. The radius method shows clusters first that are more compact, while the nearest neighbors method shows clusters without ranking them.
- If the data cloud resembles a low-dimensional manifold, the nearest neighbors method is able to reveal this. It is the only technique that is able to do so.

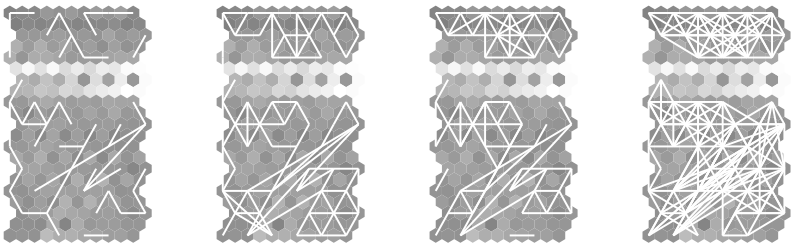
3.6 Summary

In this chapter, the Graph visualization method for Self-Organizing Maps has been introduced. It is a technique for showing the data density and several aspects of topology violations. Other than many visualization methods such as the U-Matrix, the data samples are required for computing it. The Graph method requires the computation of a proximity structure that can be computed either in a way based on nearest neighbors or on the spherical distance around the data samples.

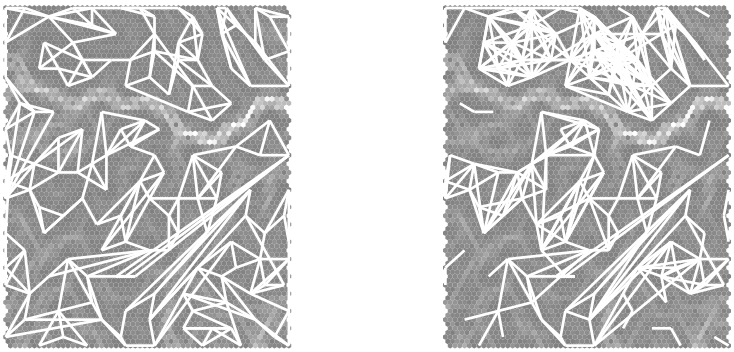
The radius method is best used in combination with other clustering and density visualization methods. By drawing a series of radius method visualizations at increasing threshold levels, the clustering structure can be shown along with the clusters' relations to each other.

The nearest neighbors method is used to show topology violations and to unveil points along low-dimensional manifolds hidden in the data set. It is best used together with the Topographic error. No other technique is able to find structure such as samples aligned along a curve in the data.

Another advantage of both Graph methods is that they can be combined with any other visualization method that uses color coding on the map.



(a) (b) (c) (d)



(e) (f)

Figure 3.21: Graph visualization for maps trained on Iris data set: (a)–(d) small $\{6 \times 11\}$ map: (a) nearest neighbors with $k = 1$, (b) $k = 3$, (c) radius method with $r = 0.5$, (d) $r = 0.8$, (e)–(f) large $\{30 \times 40\}$ Iris map: (e) nearest neighbors with $k = 3$, (f) radius method with $r = 0.5$

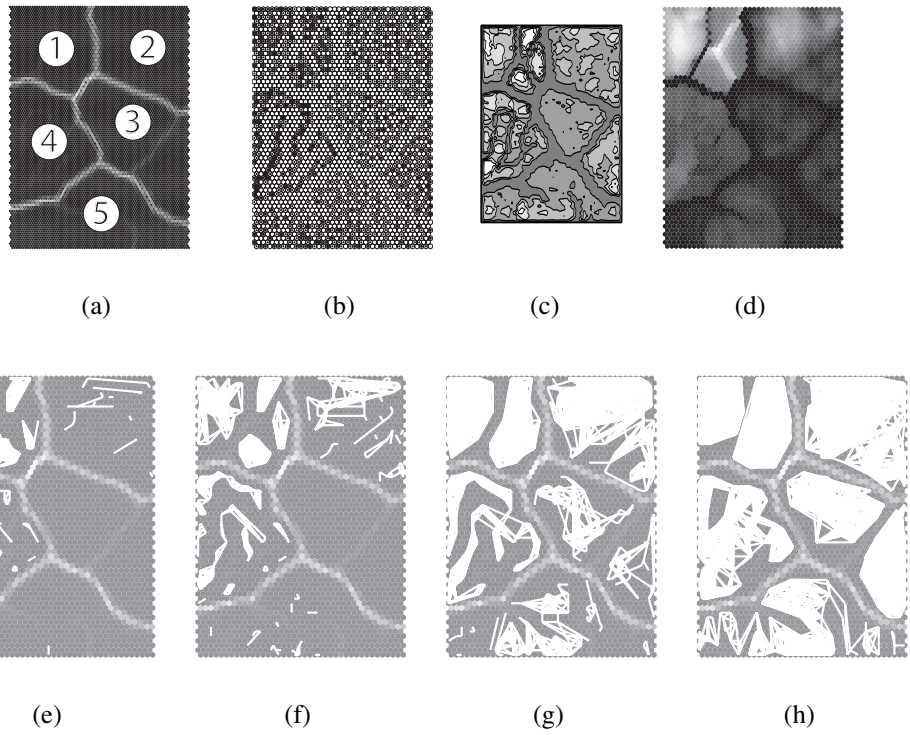
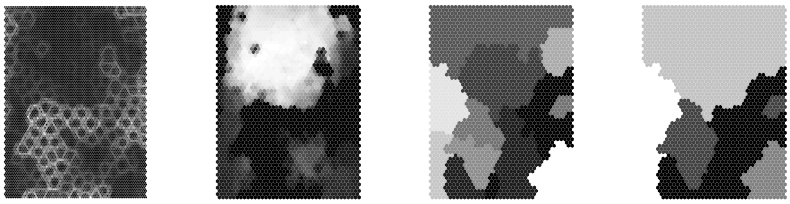


Figure 3.22: Density and clustering of the Multi-challenge data set, $\{40 \times 60\}$ SOM: (a) U-Matrix, (b) Hit histogram, (c) SDH with $s = 15$, (d) P-Matrix, radius method: (e) $r = 0.3$, (f) $r = 0.5$, (g) $r = 1.0$, (h) $r = 1.5$



(a) (b) (c) (d)

(e) (f) (g)

Figure 3.23: Density and clustering of the large $\{40 \times 60\}$ Ionosphere SOM: (a) U-Matrix, (b) P-Matrix; (c)–(d)Ward's linkage: (c) 10 clusters, (d) 5 clusters; (e)–(g) Radius method: (e) $r = 1.0$, (f) $r = 2.0$, (g) $r = 3.0$

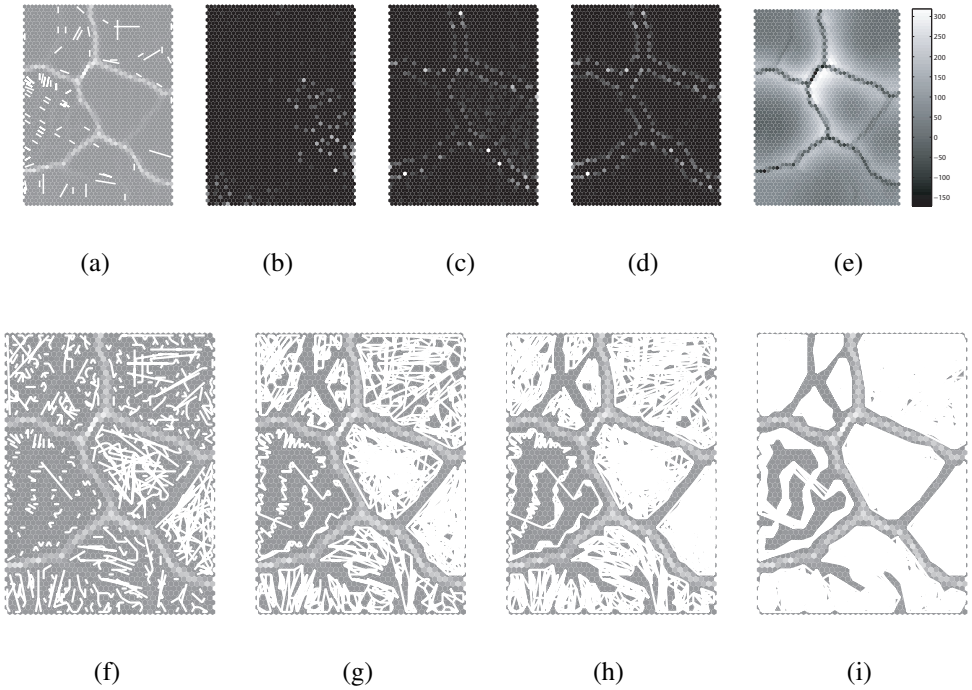


Figure 3.24: Connectivity and topology violation of the Multi-challenge data set, $\{40 \times 60\}$ SOM: (a) Topographic error, (b)–(d) SOM Distortion: (b) Quantization error, (c) neighborhood bias, (d) neighborhood variance; (e) topographic product; (f)–(i) nearest neighbors method: (f) $k = 1$, (g) $k = 3$, (h) $k = 5$, (i) $k = 10$

Chapter 4

Gradient Fields

4.1 Introduction

Many SOM visualizaton methods suffer from the association with plots of functions that many persons with engineering backgrounds have, and with their unfamiliarity with colored maps that are the basis for many of the SOM visualization methods. However, trying to assign a meaning on either the vertical or the horizontal axis does not make sense. Also, as only distances and not absolute positions are of interest, the map can actually be rotated, mirrored and flipped arbitrarily without altering its meaning. In this chapter, a visualization method is described that imposes an analogy especially for persons with engineering backgrounds who are familiar with vector fields. It comes in two variants which are called the Gradient Field and the Borderline methods. The Gradient Field is displayed as arrows that point to the most likely cluster center for each map unit. The Borderline method is an alternative representation of the Gradient Field that displays lines of varying lengths that show how clusters are separated. One property of this technique is that it can be adjusted by interactively setting a kernel smoothing parameter to show the granularity of the clustering, similar to the choice of the number of clusters in clustering algorithms, revealing the structure at various levels of detail. Several publications describe and analyze this method [77, 68, 76, 70, 69, 71].

In Section 4.2 the Gradient Field method is explained, which is a method for visualizing the clustering structure based on vector fields. The gradient is plotted on top of the SOM lattice with one arrow per map unit. The length and direction of each arrow indicate where the cluster centers are located. The entirety of the arrows forms a smooth vector field especially intended for use by professionals with engineering backgrounds, exploiting their familiarity with gradient and flow visualizations. Two extensions to the basic Gradient Field representation are introduced in Section 4.3. The Borderline visualization is derived from

the Gradient Field and provides an alternative view that emphasizes the cluster boundaries, which is shown in Section 4.3.1. In Section 4.3.2, the Gradient Fields are further extended to contrast contributing factors of the clustering structure. In Section 4.4 experiments are provided for the regular version of the visualization that show a single Gradient Field on top of the SOM, and in Section 4.5, experiments are shown for the extended version where multiple Gradient Fields are displayed. Section 4.6 elaborates systematic guidelines for the use of Gradient Fields and their variants. Section 4.7 summarizes this chapter.

4.2 Gradient Field Visualization

In this section the algorithm for obtaining a vector field that shows homogeneous areas is described. Each of its arrows $\mathbf{a}_i$ is computed based on the prototype vectors, the map topology, and the choice of the neighborhood kernel. The $\mathbf{a}_i$ will be plotted on top of their corresponding map units ξ_i. The vectors $\mathbf{a}_i$ have a u and v component, denoted as $\mathbf{a}_i^{\mathrm{u}}$ and $\mathbf{a}_i^{\mathrm{v}}$, corresponding to the horizontal and vertical coordinates, respectively. The algorithm outlined in the next paragraphs consists of two main steps which are repeated for each map unit: In the first one, weighted distances to all other prototype vectors are computed and separated along both the u and v axes in positive and negative directions. In the second step, these contributions are aggregated for each coordinate and normalized in order to avoid border effects.

From this point, the algorithm is outlined to compute the coordinates of $\mathbf{a}_i$, for map unit ξ_i and its associated prototype vector $\mathbf{m}_i$. Some of the formulas involve pair wise comparisons to other units and models vectors, which will be denoted with index j. These computations have to be performed for all $1 \leq j \leq M$ with $j \neq i$. First the vector connecting the positions of ξ_i and ξ_j in output space needs to be obtained, which is defined as

$$\chi_{ij} = \vec{\xi_i \xi_j} = \xi_j - \xi_i \tag{4.1}$$

The angle α of this vector χ_{ij} is

$$\alpha_{ij} = \arctan\left(\frac{\chi_{ij}^{\mathrm{v}}}{\chi_{ij}^{\mathrm{u}}}\right) \tag{4.2}$$

Figure 4.2 shows an illustration of how the output space is annotated. Using this notation, it is now possible to apply the neighborhood kernel to the distance between units ξ_i and ξ_j, which is the length of χ_{ij}, and to split up this weight into u and v directions:

$$\omega_{ij}^{\mathrm{u}} = \cos\alpha_{ij} \cdot h_\sigma(||\chi_{ij}||), \qquad \omega_{ij}^{\mathrm{v}} = \sin\alpha_{ij} \cdot h_\sigma(||\chi_{ij}||) \tag{4.3}$$

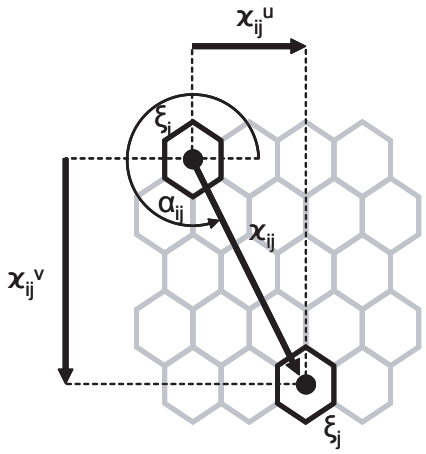


Figure 4.1: Notation of output space

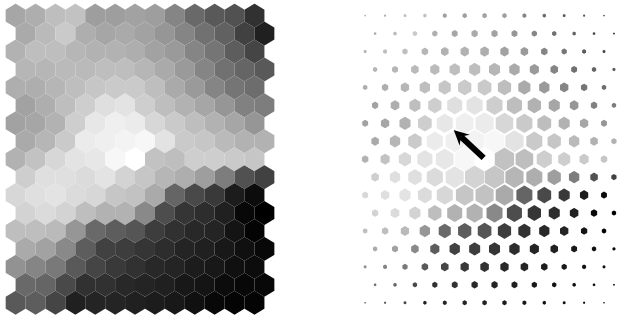
The value of ω_{ij}^{u} will be close to zero when either the distance between ξ_i and ξ_j is high, resulting in a very low kernel value, or in case $\xi_i^{\mathrm{u}} = \xi_j^{\mathrm{u}}$, i.e. ξ_i is directly above or below ξ_j with no horizontal offset. The value of σ also influences the kernel function and thus the value of ω since high σ tend to weight far-away map units higher than with low σ values. Note that ω_{ij}^{u} will be negative in case ξ_j is to the left of ξ_i.

In the following, formulas will only explicitly be provided for the u coordinate, as v follows analogously. In the next step, the distances between the associated prototype vectors $\mathbf{m}_i$ and $\mathbf{m}_j$ are taken into account, weighting these distances by ω, and assigning them to either the positive or negative side of u:

$$\delta_{ij}^{\mathrm{u}+} = \begin{cases} d_I(\mathbf{m}_i, \mathbf{m}_j) \cdot \omega_{ij}^{\mathrm{u}} & \text{if } \omega_{ij}^{\mathrm{u}} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.4}$$

$$\delta_{ij}^{\mathrm{u}-} = \begin{cases} d_I(\mathbf{m}_i, \mathbf{m}_j) \cdot (-\omega_{ij}^{\mathrm{u}}) & \text{if } \omega_{ij}^{\mathrm{u}} < 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.5}$$

Dividing the weighted distance contribution of the prototype vectors along the positive and negative directions will provide a means to find the direction where the dissimilarity of the current vector $\mathbf{m}_i$ is relatively low, and where the arrow $\mathbf{a}_i$ will ultimately point to. If, for example, ξ_j is next to ξ_i on the right side, and the distances between the prototype vectors are high, then $\delta_{ij}^{\mathrm{u}+}$ will be high, which will contribute significantly to $\mathbf{a}_i$ pointing to the left, away from ξ_j. Figure 4.2 gives a schematic overview of the weighting process to the unit in the center of the map ξ_i: First, the pair wise feature space distances $d_I(\mathbf{m}_i, \mathbf{m}_j)$ are shown in Figure 4.2(a), where the i^{th} node is the one in the middle of the map, and nodes



(a) (b)

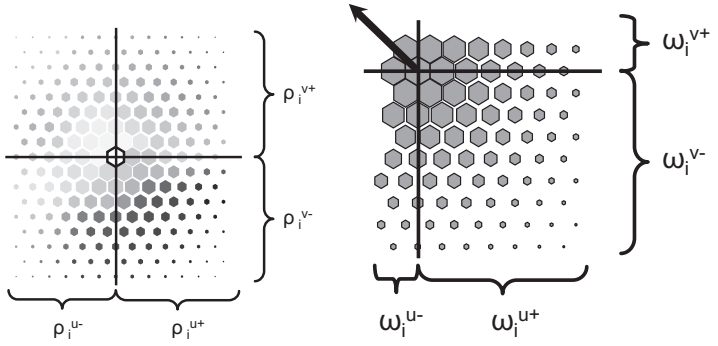
Figure 4.2: (a) Distances to prototype of map unit in the center; dark values denote high distances, (b) map unit size scaled according to Gaussian neighborhood kernel with $\sigma = 4$

are colored according to the distances in feature space, where dark colors denote high distances; then, the sizes of the hexagons are scaled according to the kernel value of their output space distance to the center node $h_\sigma(||\chi_{ij}||)$ in Figure 4.2(b). The arrow in this figure points in the direction where the least weighted distances are. Repeating this calculation for all the ξ_j, the $\delta_{ij}^{\mathrm{u}+}$ and $\delta_{ij}^{\mathrm{u}-}$ values can finally be aggregated:

$$\rho_i^{\mathrm{u}+} = \sum_{j=1\ldots M, j\neq i} \delta_{ij}^{\mathrm{u}+}, \qquad \rho_i^{\mathrm{u}-} = \sum_{j=1\ldots M, j\neq i} \delta_{ij}^{\mathrm{u}-} \tag{4.6}$$

This calculation step is illustrated in Figure 4.3(a). Each hexagon stands for a δ, which is calculated from the distance to the node in the center and the distance in feature space. The distance to the node in output space is shown as the size of the hexagon, while the color of the node denotes the distance in feature space. The values for ρ are thus computed as adding the δ values for each coordinate in positive and negative directions.

Once $\rho_i^{\mathrm{u}+}$ and $\rho_i^{\mathrm{u}-}$ is known, it can be told whether and to which extent one of the directions outweighs the other. For $\rho_i^{\mathrm{u}+} > \rho_i^{\mathrm{u}-}$, the accumulated distances from the right side are bigger than the ones on the left, so the arrow will point to the left. Close to the map borders, the distance contributions in the direction of the border will always be lower than in the direction inside of the map since there are simply no prototype vectors for which a distance could be computed. The resulting arrows are thus biased to always point outside of the map. To avoid this, a normalization has to be performed that sums the ω_{ij}^{u} values in positive and



(a) (b)

Figure 4.3: Schematic illustration of weight aggregation and arrow normalization: (a) Weight aggregation, calculation of ρ , (b) arrow normalization, calculation of ω

negative directions, i.e. counts the weights of the nodes on either side of the unit being analyzed:

$$\omega_i^{\mathrm{u}+} = \sum_{j=1\ldots M, j\neq i} \begin{cases} \omega_{ij}^{\mathrm{u}} & \text{if } \omega_{ij}^{\mathrm{u}} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.7}$$

$$\omega_i^{\mathrm{u}-} = \sum_{j=1\ldots M, j\neq i} \begin{cases} -\omega_{ij}^{\mathrm{u}} & \text{if } \omega_{ij}^{\mathrm{u}} < 0 \\ 0 & \text{otherwise} \end{cases} \tag{4.8}$$

This normalization scheme is illustrated in Figure 4.3(b). Other than Figure 4.3(b), only the output space distances are of interest, and not the node colors that indicate the feature space distance, thus all hexagons are grey. For a node that is located close to the map border, the weights ρ from the direction away from the border will almost certainly be dominating, resulting in arrows that will always point outside of the map. This is not a desired effect. To counter this, the sums of the output space nodes that are present to each side of the current unit are used to weigh the ρs of the other direction. Thus, the components of $\mathbf{a}_i$ can finally be determined as

$$a_i^{\mathrm{u}} = \frac{\rho_i^{\mathrm{u}-} \cdot \omega_i^{\mathrm{u}+} - \rho_i^{\mathrm{u}+} \cdot \omega_i^{\mathrm{u}-}}{\rho_i^{\mathrm{u}+} + \rho_i^{\mathrm{u}-}} \tag{4.9}$$

where the accumulated input space differences $\rho_i^{\mathrm{u}+}$ and $\rho_i^{\mathrm{u}-}$ are weighted by the opposing accumulated $\omega_i^{\mathrm{u}-}$ and $\omega_i^{\mathrm{u}+}$, respectively. In case the node ξ_i does not lie close to the edge of the u axis, $\omega_i^{\mathrm{u}+}$ and $\omega_i^{\mathrm{u}-}$ will be equal, and the effect of

this normalization will vanish. The most important of the computational steps are performed in Equations 4.6 and 4.9. The roles of ρ_i^+ and ρ_i^- of either u or v coordinates deserve special attention and will be briefly discussed:

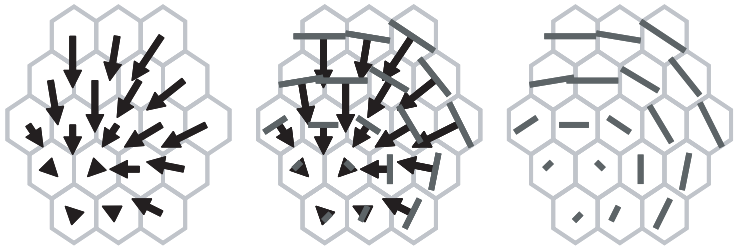
- $\rho_i^+ \approx \rho_i^-$: In this case, distances are balanced and the component of a will be small. This happens in two cases: If the values of ρ_i^+ and ρ_i^- are small, the prototype vectors $\mathbf{m}_j$ of the surrounding map units of ξ_i are very similar to $\mathbf{m}_i$ and likely to be in the center of a cluster. In this case, the arrows of neighboring units are pointing to it. In case both ρ_i^+ and ρ_i^- are large, ξ_i is likely to be right in between two clusters to neither of which it belongs. Such units are called interpolating units and are easily recognized as arrows of neighboring units are pointing away from them.

- $\rho_i^+ > \rho_i^-$: The distances in positive direction outweigh distances in negative direction. $\mathbf{m}_i$ is more similar to its neighbors in negative direction and $\mathbf{a}_i$ will reflect this by pointing there.

- The length of $\mathbf{a}_i$ is determined by the difference between ρ_i^+ and ρ_i^-. Large differences result in long arrows and are due to a high similarity to a cluster centroid.

This method differs from the U-Matrix in the way that it can be represented as a field of arrows, and by the smoothing that is performed by the kernel to override small cluster boundaries that may be artifacts that come from the choice of the map size. The choice of the kernel width σ plays an important role in what the visualization actually shows, since a small value of σ weights the direct neighbors of the map unit ξ_i much stronger than the other units, while a large value takes the surrounding units into account, weighting them nearly equally, and thus smoothing over wide areas. The effect of choosing σ lies in whether the visualization is a fine-grained vector field, where the neighboring arrows vary sharply, or a coarse representation that show the global clustering structure. One of the main advantages of this method is that it allows interactively setting and tuning σ to different levels, so different kinds of clustering structures can be observed. The value of σ has to be seen in relation to the size of the map. For usual settings, experiments have shown that setting σ to around one sixth of the number of units along the shorter side of the map usually results in a balanced visualization that is neither too coarse nor too granular, which is thus the recommended parameterization. Examples and further discussion of σ are provided in Section 4.4.1.

The computational complexity of calculating the Gradient Field is $O(M^2)$ since it requires the pair wise distances between the map nodes and prototype vectors. In case the kernel is cut off after a certain threshold, like $h^{\mathrm{G'}}$, h^{b}, h^{ip}, or h^{l}, which is plausible since distant map nodes influence each other by negligible



(a) (b) (c)

Figure 4.4: From Gradient Field to Borderline visualization: (a) Gradient Field visualization, (b) Gradient Field and Borderline, (c) Borderline visualization

amounts, the complexity reduces to $O(M)$. The computational cost of the U-Matrix also scales linearly with the number of map units, thus computation of the Gradient Field is not more expensive in terms of complexity than the U-Matrix. The experiments in Section 4.4, which include SOMs with up to 2000 nodes, have been computed in several seconds even for the $O(M^2)$ case, running under an implementation in Matlab code, thus offering itself for interactive visualization with different sets of σ.

4.3 Variations and extensions to the Gradient Field visualization

4.3.1 Borderlines representation

Another representation that emphasizes cluster boundaries over vector fields pointing towards cluster centers can be easily derived. By depicting the orthogonal of each $\mathbf{a}_i$ as a line from both sides of the center instead of an arrow, the resulting visualization shows likely cluster boundaries. A schematic illustration is shown in Figure 4.4. The length of the arrows is preserved such that long lines hint at a strong separation between clusters. This representation is called the Borderline visualization. Further examples are shown in Section 4.4.

4.3.2 Extension to groups of component planes

In this section, an extension to the Gradient Field technique is described that focuses on correlations between variables on SOMs. As data mining methods usu-

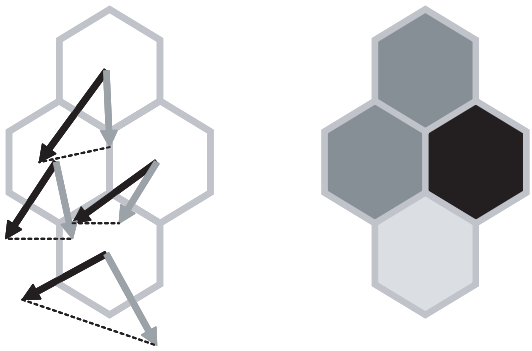


Figure 4.5: Groups of component planes: On the left, 2 Gradient Fields are shown, along with the distance between the arrows; to the right, the contrast plot shows the length of the arrows as color code

ally assume that the observed data samples follow an unknown statistical distribution, local correlations and dependencies between variables are often an interesting property to assess. The clustering structure is assumed to be induced by certain groups of variables. The variables of these distinct groups are either correlated to a certain degree or statistically dependent in a non-linear way. This assumption implies that the clustering structure can be decomposed into these groups of variables. The basic idea is to plot two or more groups of variables simultaneously with the Gradient Field method. This combined visualization shows the contributing factors of the clustering.

The reliance on selected variables, i.e. component planes, rather than the complete codebook vectors and their distances requires the introduction of some additional formal definitions. A projection of the the codebook's i^{th} variable is called the i^{th} component plane $\mathbf{m}_{(i)}$, which lies in a one-dimensional subspace of the feature space. Component planes can be conveniently visualized on the map lattice. Groups of component planes are defined as the combination of several variables. The set that consists of all the indices of these variables is denoted as $\mathfrak{S} = \{1, \ldots, M\}$. Further, it is of interest to compare g subsets $\mathfrak{S}_{1,\ldots,g} \subseteq \mathfrak{S}$. The sets must be disjoint, i.e. $\mathfrak{S}_i \cap \mathfrak{S}_j = \emptyset, \quad i, j \leq g, \forall i \neq j$ must hold. The union $\bigcup_{i=1}^{g} \mathfrak{S}_i$ does not necessarily have to contain all the variable indices if only a subset of the variables is of interest. By $\mathbf{M}^{(\mathfrak{S}_i)} = \bigtimes_{k \in \mathfrak{S}_i} \mathbf{m}_{(k)}$, where $\times$ refers to the Cartesian Product, the reduced codebook is denoted, which consists of the component planes $\mathfrak{S}_i$. $\mathbf{M}^{(\mathfrak{S}_i)}$ is a "sub-SOM" with the same number of map units, for which e.g. the U-Matrix and Gradient Field visualizations can be computed.

Interesting subsets $\mathfrak{S}_{1,\ldots,g}$ can be chosen either based on correlations between component planes [111, 35] or by grouping variables that are known to be se-

mantically similar, for example variables that have a common source. The former choice can be performed by either investigating the correlation matrix or by visual inspection of the component planes, combining the ones that are similar. Determining the correlation of component planes can be performed by

$$d_{comp}(i,j) = \text{abs}(\gamma(\mathbf{m}_{(i)}, \mathbf{m}_{(j)})) \tag{4.10}$$

where $\gamma(\mathbf{m}_{(i)}, \mathbf{m}_{(j)})$ is a suitable measure of correlation such as the Pearson correlation coefficient on the variables $\mathbf{m}_{(i)}$ and $\mathbf{m}_{(j)}$. For highly similar component planes, the absolute value is close to 1. If there is no linear correlation, its value is close to zero. Grouping components together is then performed by partitioning the set of component planes. The more common choice is grouping variables that are semantically similar, since this exploits a-priori knowledge about the data. Next, it is investigated whether the groups provide contrasting clustering structures.

Once the relevant variable groups have been selected, the Gradient Field method can be applied to all g sets and visualized simultaneously. In order to adjust the length of the arrows to negate the effect of different numbers of variables between the sets, a normalization is performed:

$$\hat{\mathbf{a}}_i^{(\mathfrak{S}_1)} = \mathbf{a}_i^{(\mathfrak{S}_1)} \frac{|\mathfrak{S}_1|}{\sum_{i=1}^{g} |\mathfrak{S}_i|} \tag{4.11}$$

where $|\cdot|$ denotes the cardinality of a set.

In Figure 4.3.2, an example is shown of dual component planes. It also shows the contrast plot, which shows the distance between two vectors, depicted as color codes. In Section 4.5.2, examples are provided that explain the relation to variable dependence and applications on a real-world data set.

4.4 Experiments of single Gradient Field and Borderline visualizations

In this section, the usefulness of the previously described techniques is investigated with artificial, benchmark, and real-world data sets. The Multi-challenge data set has already been introduced in the previous chapter. The Phonetic data set is a benchmark data set, which describes 20 distinct phonemes from continuous Finish speech, which are measured in 20 variables. It consists of 1962 data samples. The Fracture Optimization data set comes from the domain of petroleum engineering [123]. It has been collected from 199 gas wells in 10 dimensions. The examples shown here outline the properties and usage of the Gradient Field and Borderline methods, discussing the effects of the parameter σ and of the map size on the visualization.

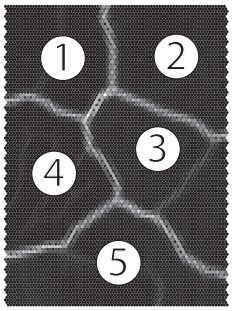


Figure 4.6: Multi-challenge data, $\{40 \times 60\}$ SOM: U-Matrix with lables for subsets: 1 - cluster-of-clusters, 2 - overlapping Gaussians, 3 - high-dimensional Gaussians, 4 - intertwined rings, 5 - curve consisting of line segments

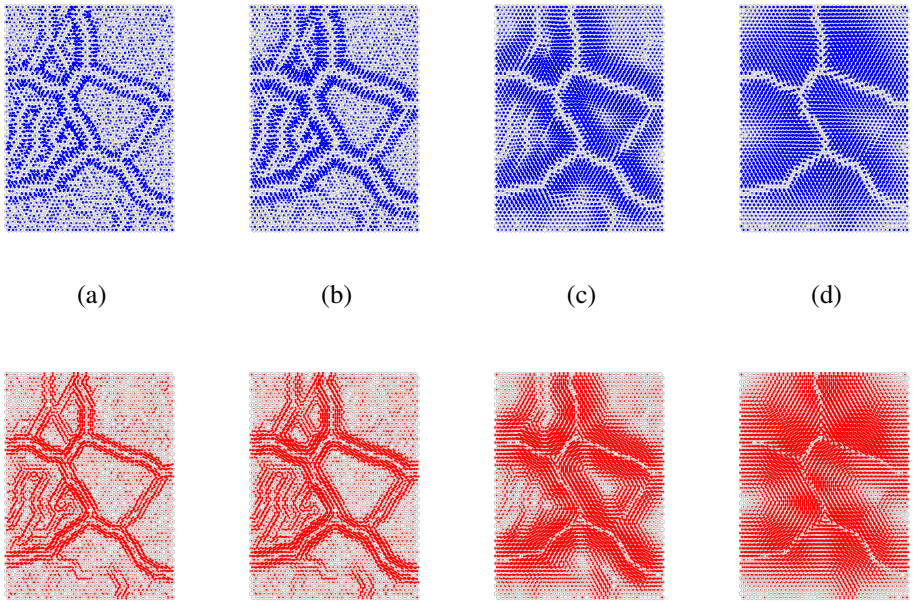
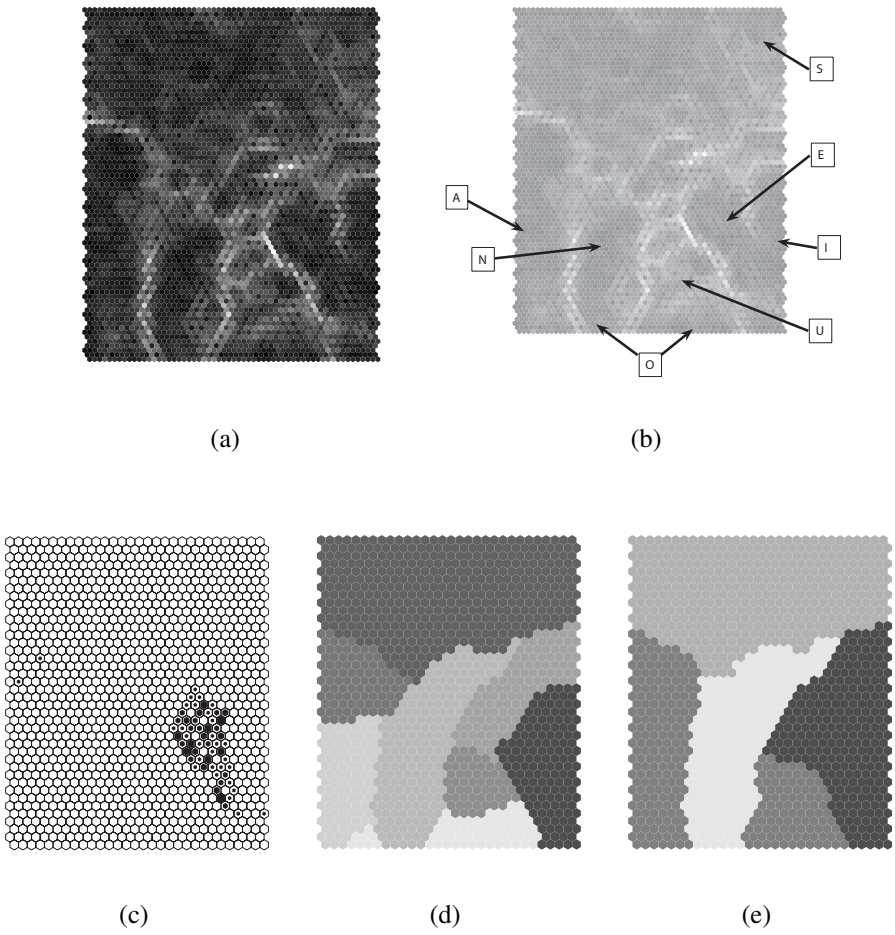


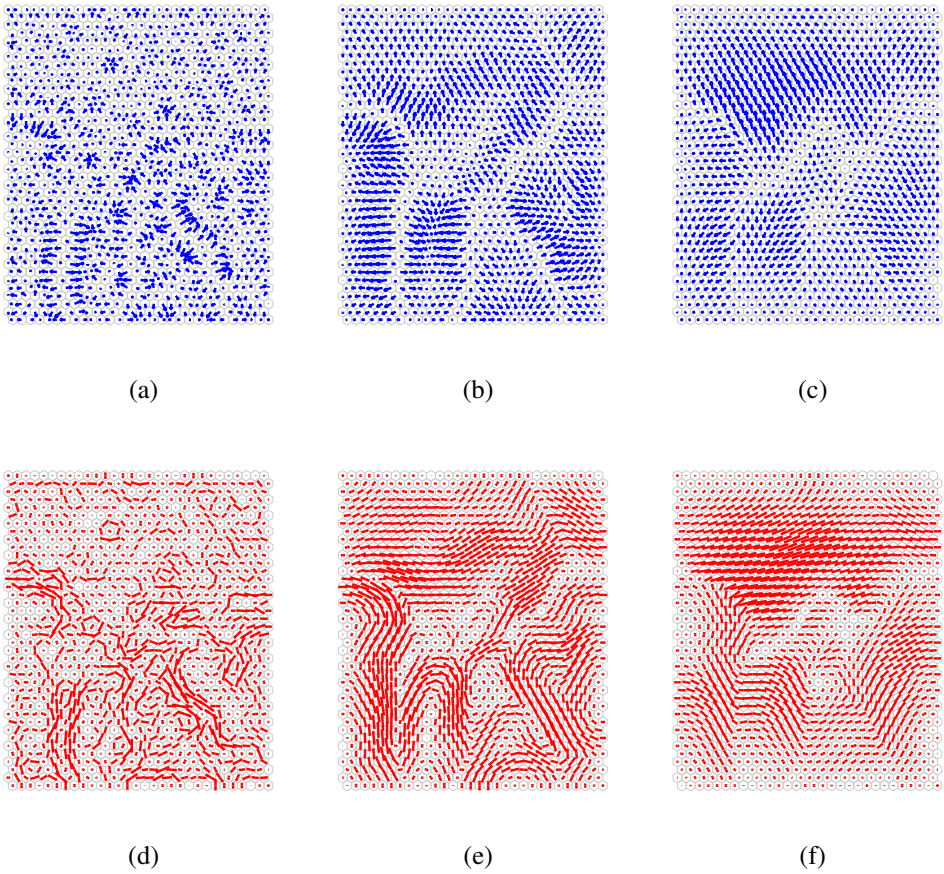
Figure 4.7: Multi-challenge data, $\{40 \times 60\}$ SOM, (a)–(d) Gradient Field visualization: (a) $\sigma = 0.5$, (b) $\sigma = 1$, (c) $\sigma = 3$, (d) $\sigma = 10$; (e)–(h) Borderline visualization: (e) $\sigma = 0.5$, (f) $\sigma = 1$, (g) $\sigma = 3$, (h) $\sigma = 10$



(a) (b)

(c) (d) (e)

Figure 4.8: Overview of Phonetic data set, $\{30 \times 40\}$ SOM: (a) U-Matrix, (b) phoneme labels on top of U-Matrix, (c) hit histogram of data with label "E", (d) k-means clustering with 8 clusters, (e) k-means clustering with 4 clusters



(a) (b) (c)

(d) (e) (f)

Figure 4.9: Phonetic data, $\{30 \times 40\}$ SOM, (a)–(c) Gradient Field visualization: (a) $\sigma = 1$, (b) $\sigma = 5$,(c) $\sigma = 15$; (d)–(f) Borderline visualization: (d) $\sigma = 1$, (e) $\sigma = 5$,(f) $\sigma = 15$

4.4.1 Effects of the neighborhood radius

At first, the effects of the kernel width σ are shown with a SOM trained on the Multi-challenge data set with a Gaussian kernel h^{G}. Figure 4.6 recalls the placement of the subsets onto the $\{40 \times 60\}$ map. In Figure 4.7, the visualizations of the Gradient Field and Borderline methods are shown. Different levels of σ are depicted: 0.5, 1, 3, and 10. The first two are very low values, and cause the calculation of the vectors to emphasize the nodes in their direct neighborhood. The dividing boundaries that lie in between the subsets are visible along with the finer gaps in the cluster-of-clusters subset. As the threshold σ is increased, a smoothing over an extended neighborhood occurs, with the effect of eliminating the finer gaps in subsets "1" and "3", while preserving and strengthening the dominating gaps between the subsets. As the radius is increased from 1 to 3, the boundaries between the smaller clusters in the cluster-of-clusters subset vanish, and the subset only shows two big clusters. By further increasing σ to 10, the separation between these clusters also disappears, and only the 5 subsets can be identified as clusters. The Gradient Field and Borderline methods are thus able to show a hierarchy of the clustering structure when the threshold parameter σ is varied.

The same effect is investigated with the Phonetic data set. In this case, many of the phoneme classes coincide with the homogeneous areas on the map. The U-Matrix of this $\{30 \times 40\}$ SOM is shown in Figure 4.8(a). Figure 4.8(b) highlights some regions that are primarily occupied by one vowel or consonant. As an example for a phoneme that is highly clustered, Figure 4.8(c) shows the hit histogram for data samples of phoneme "E". In Figure 4.9, the results for both Gradient Field and Borderline methods for $\sigma = 1, 5, 15$ are shown. The σ values "1", "5", and"15" have been chosen to represent low, medium, and high radii, respectively, in relation to the map size of $\{30 \times 40\}$. Low values of this parameter lead to very granular visualizations, where only direct neighbors are taken into account for the computation of each arrow and thus only local gradients can be observed, as visualized in Figures 4.9(a),(d). By increasing σ, the clustering structure revealed shifts gradually from local towards global. Figures 4.9(b),(e) provide a far better overview on the clustering structure, and individual regions can be distinguished. In Figures 4.9(c),(f), the global structure is shown for $\sigma = 15$. It reveals that the upper third, which is occupied by phonemes that do not correspond to a letter from the alphabet, is most strongly separated from the rest of the map, which has not been indicated by any of the former, more local representations.

The relation to a clustering algorithm is shown in Figures 4.8(d),(e), which show the results of k-means of the codebook vectors with 8 and 4 clusters. The map topology is omitted for the clustering process, so clusters can consist of non-adjacent nodes. σ is loosely related to the number of clusters, as high values of σ show few boundaries and are comparable to clustering with few cluster centers,

while a low σ results in many local cluster boundaries and is thus comparable to clustering with many cluster centers. When compared to Figure 4.9, it can be seen that the $k = 4$ clustering reveals similar information as the Gradient Field method with a high σ, while $k = 8$ is comparable to the ones with a lower smoothing parameter.

The choice of σ has to be performed interactively and varies for the type of kernel used, but a good starting point for the Gaussian kernel h^{G} is $1/6^{\mathrm{th}}$ of the shorter side of the map. σ always has to be seen in relation to the size of the map since it is defined in absolute terms over the number of units over which the smoothing is performed. While the choice of σ strongly influences the type of information on the structure revealed, the results are insensitive to the choice of the neighborhood kernel function. No significant differences between different neighborhood kernels could be noticed. The only exception is the Bubble kernel h^{b}, which is not continuous in a mathematical sense and does not allow a smooth convergence across the neighborhood range, and is also hardly used for SOM training. Thus, a cut-off kernel variant may be employed, resulting in linear complexity for calculating the Gradient Field.

4.4.2 Smoothing sparse maps

The next example shows the smoothing effect on sparse maps which are sometimes preferred if the SOM is used for visualization only and vector quantization is not of interest. These SOMs are trained with large numbers of units [101]. A $\{30 \times 40\}$ SOM is used as an example of an oversized SOM trained on the Iris data set with a Gaussian kernel h^{G}. Since the number of the map nodes (1200) is 8 times higher than the number of data samples (150), the U-Matrix visualization shown in Figure 4.10(a) shows artifacts that come from the fact that most samples are mapped to nodes that they occupy solely. The remaining units are merely interpolating units. The U-Matrix implies that these transitions are actually cluster boundaries, while the significant boundary between the upper and lower parts are overshadowed. The density is depicted with the P-Matrix visualization in Figure 4.10(b). Most of the map shows roughly the same density except for the corners and the part in the center that extends to the right side of the map. Figure 4.10(c) shows the U*-Matrix. It looks quite different from the U-Matrix and shows that there is a big boundary in the upper third, and a smaller one in the lower right-hand corner, while the artifacts around the samples across the rest of the map vanish. In Figure 4.10(d), a Gradient Field with $\sigma = 8$ is visualized, the radius being roughly one sixth of the map axes. The smoothing effect overrides the insignificant cluster boundaries and focuses on a more global clustering structure, which consists of two big clusters on the upper and lower part of the map and a transition region slightly right of the center. It is thus possible to adapt the

visualization accordingly for sparse SOMs.

4.5 Experiments of groups of component planes visualizations

In this section, the extension to groups of component planes is demonstrated. The use of this extension lies in contrasting and determining the influence of groups of variables that are believed to contribute in different ways to the overall shape, clustering structure, or correlation of the whole data set. The data sets used include several artificial data sets that are selected in order to demonstrate scenarios where typical patterns of statistical dependence or independence are present between the data. It is shown how the SOM and multiple Gradient Field visualizations are able to reveal such patterns. The method is also shown on the Fracture Optimization data, the variables of which are measured from 3 distinct sources.

4.5.1 Statistical dependencies between groups of variables

Next, the effect of the Dual Gradient Field method on 4 artificial data sets is examined where the aim is to find out whether one variable is statistically dependent on the other two. The data sets consist of $10,000$ samples. The first 3 examples are 3-dimensional, and the last one is 20-dimensional. In the 3-dimensional examples, the first two variables x_1 and x_2 are uniformly distributed between 0 and 1, and are statistically independent. The set of indices forming this group is denoted as $\mathfrak{S}_{\text{uniform}} = \{1, 2\}$. The second group consists of the third probably dependent variable $\mathfrak{S}_{\text{prob\_dep}} = \{3\}$ for which it is desired to know whether it can be explained by the former two variables.

In the first example, a third variable x_3 is considered that is independent of the former two and is also uniformly distributed. Scatterplots for this data set are provided in Figure 4.11(e), which show the variables x_1, x_2 and x_3 in rows and columns, and pair-wise scatterplots where they intersect; the bar charts show the distribution of each single variable. The scatterplots clearly show that there is no correlation between any of the variables. Figure 4.11(a)–(c) shows the component planes after training a $\{30 \times 30\}$ SOM with a cut-off Gaussian kernel $h^{\text{G'}}$ on this data. The Dual Gradient Fields in Figure 4.11(f) show that most arrows do not have common directions. To emphasize this, the length of the vector connecting the two arrows $\|\mathbf{a}_i^{\mathfrak{S}_{\text{uniform}}} - \mathbf{a}_i^{\mathfrak{S}_{\text{prob\_dep}}}\|_O$ is calculated, as can be seen in Figure 4.11(d). This value is high (light values) if the black and grey arrows point in different directions and low (dark values) if the black and grey arrows are similar. The figure has numerous light nodes, and thus indicates that there are no dependencies

between the variable groups.

In the second setup, the third coordinate of each sample is defined as $x_3 = \frac{x_1+x_2}{2}$. Scatterplots can be seen in Figure 4.12(e). Pearson's correlation coefficient between x_1 and x_3 is 0.7, and between x_2 and x_3 is -0.7, indicating strong linear dependence. The component planes plots for the $\{30 \times 30\}$ SOM on this data are shown in Figures 4.12(a)–(c). It can be seen that the projection results in linear ascent along diagonal lines, which are orthogonal for x_1 and x_2, stressing their independence. This has not happened in the previous example. Here, the data set is a 2-dimensional subspace embedded in the 3-dimensional feature space, and thus equal in dimension to the map lattice. The dependent component x_3 interferes with the other axes. When the Dual Gradient Field method is applied for groups $\mathfrak{S}_{\text{uniform}}$ and $\mathfrak{S}_{\text{prob\_dep}}$, the result can be seen in Figure 4.12(f). Aside from some deviations introduced by the SOM's border effect, it shows that the cluster structure of this map is caused by the same factors, since the black and grey arrows are very similar both in angle and length. This is an expected result, since the third variable is predictable, and it will not introduce a different clustering structure than the one already present from the previous coordinates. Figure 4.12(d), where again the differences of the black and grey arrows are depicted, shows that the arrows are very similar for all parts of the map.

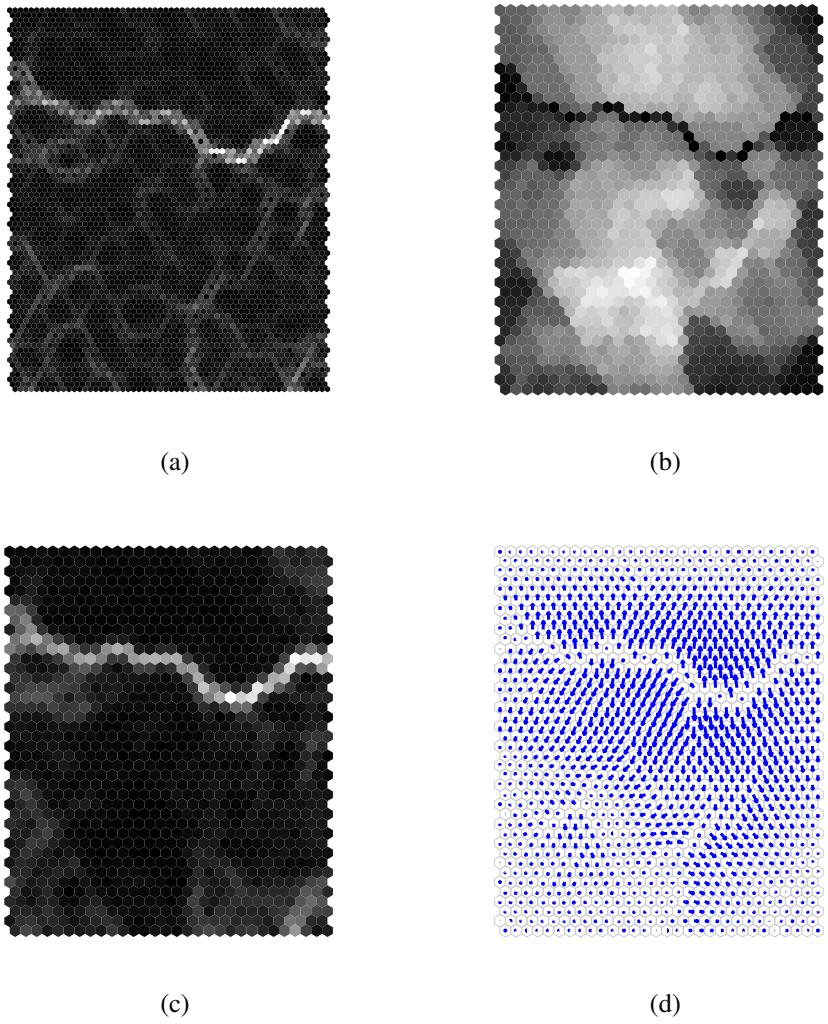
In the third case, the dependent variable is given as $x_3 = \text{abs}(x_1 + x_2 - 1)$ which is then multiplied by a factor to normalize its variance to the other variables. Although there exists a deterministic relationship between the variables, the correlation is zero for all pairs of variables, since there is no global linear relationship. x_3 is only piece-wise linearly dependent. Scatterplots are shown in Figure 4.13(e), which reveal that there is some sort of dependency between x_1 and x_3, and between x_2 and x_3. Component planes for the SOM trained on this data are visualized in Figures 4.13(a)–(c). The component plane for x_3 shows that the peak vales are on two edges of the map, the SOM thus has adjusted properly to this 2-dimensional manifold. By applying the Gradient Fields, the arrows in the regions with a linear relationship are almost identical. The smoothing performed to obtain the arrows only weights gradients within a certain radius, and the linear relationship can be observed within this radius. In the transition region, however, where x_3 approaches zero, no linear relationship is found, resulting in high differences between the arrows. Again, in Figure 4.13(d) the lengths of the difference vector between the arrow is depicted. This deviation shows where the linear relationship is not given, but recognizing that there is a linear relationship in most other areas of the map. When compared to Figures 4.11(d) and 4.12(d), where the same is performed for the independent and the linearly dependent case, it can now be shown in Figure 4.13(d) where the visualization recognizes piece-wise linear relationships. While most statistical coefficients fail to quantify the deterministic dependence of x_3 in this case, the method can be used in order to identify

piecewise linear portions of the data. Also, this implies that if cluster structures of variable groups overlap in certain regions of the map, this can be learnt via piece-wise dependencies between the variables.

The last example examines a more complex high-dimensional data set in 20 variables. Again, the data are split into two groups, and the dependence of the second group $\mathfrak{S}_{\text{prob\_dep}} = \{11, \ldots, 20\}$ is investigated. The first group $\mathfrak{S}_{\text{uniform}} = \{1, \ldots, 10\}$ is further divided into two subgroups of 5 dimensions. The variables in these subgroups are equally distributed with zero mean, and are constructed in a way that they are highly correlated within each subgroup with correlation coefficient of 0.9. Pairs of variables from different subgroups are not correlated. The second group of 10 variables is dependent on the first one and is constructed by an XOR-like function

$$x_k = \text{sign}(x_i) \cdot \text{sign}(x_j) \tag{4.12}$$

with $11 \leq k \leq 20$ the variable from the second group to be computed, $1 \leq i \leq 5$ a variable of the first subgroup, and $6 \leq j \leq 10$ a variable from the second subgroup. The results for the Dual Gradient Flow can be seen in Figure 4.14. Figure 4.14(a) shows an example component plane of $\mathfrak{S}_{\text{prob\_dep}}$ computed by Equation 4.12. In Figure 4.14(c), the results of the Dual Gradient Flow method are depicted. The arrows are highly divergent in the regions where the boundaries are: The grey arrows, denoting $\mathbf{M}^{(\mathfrak{S}_{\text{prob\_dep}})}$, point away from these boundaries towards their 4 cluster centers, while the black arrows are almost uniform over the map, with a small disturbance in the middle of the map that was probably introduced during training. The difference is visualized in Figure 4.14(b), which shows the deviation of the arrows. In the dark areas, the statistical relationship between the two groups is evident, while the light areas correspond to transitions. This is another example of a non-linear dependency that cannot be captured by a linear correlation coefficient, which is zero for pairs of variables from $\mathfrak{S}_{\text{prob\_dep}}$ and $\mathfrak{S}_{\text{uniform}}$.



(a) (b)

(c) (d)

Figure 4.10: Iris data set, $\{30 \times 40\}$ SOM: (a) U-Matrix, (b) P-Matrix, (c) U*-Matrix, (d) Gradient Field with $\sigma = 8$

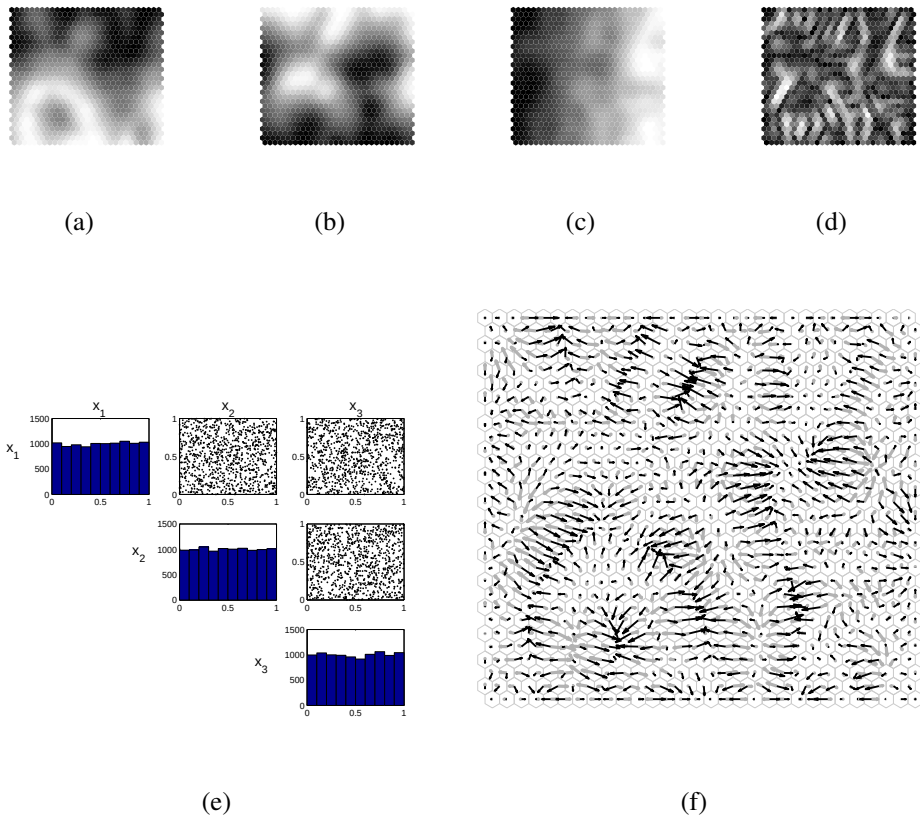


Figure 4.11: Artificial Data SOM (no relationship): Component planes: (a) x_1, (b) x_2, (c) x_3; (d) length of difference vector (contrast plot), (e) scatterplots and distribution of x_1, x_2, x_3, (f) groups of component planes $\mathbf{M}^{(\mathfrak{S}_{\text{uniform}})}$ vs $\mathbf{M}^{(\mathfrak{S}_{\text{prob\_dep}})}$

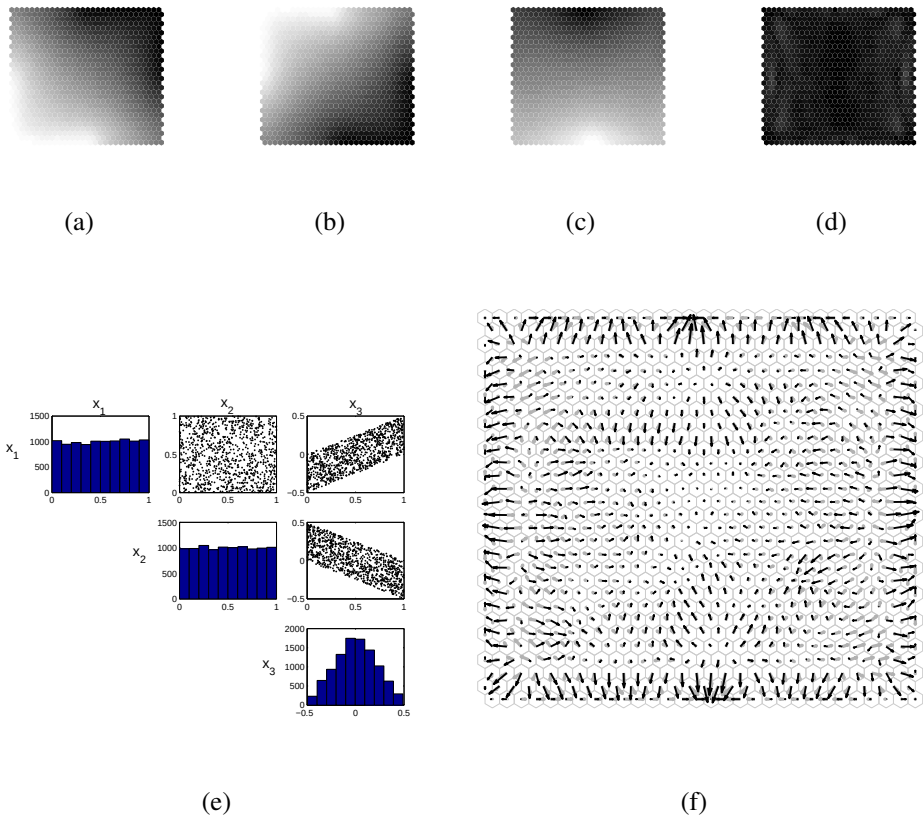


Figure 4.12: Artificial Data SOM (linear relationship): Component planes: (a) x_1, (b) x_2, (c) x_3; (d) length of difference vector (contrast plot), (e) scatterplots and distribution of x_1, x_2, x_3, (f) groups of component planes $\mathbf{M}^{(\mathfrak{S}_{\text{uniform}})}$ vs $\mathbf{M}^{(\mathfrak{S}_{\text{prob\_dep}})}$

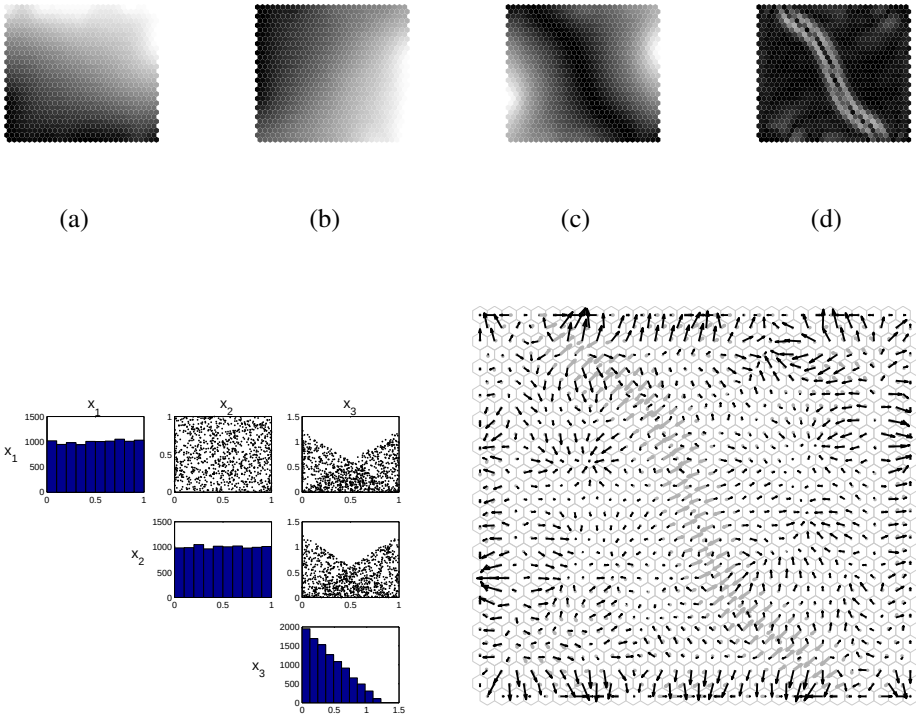
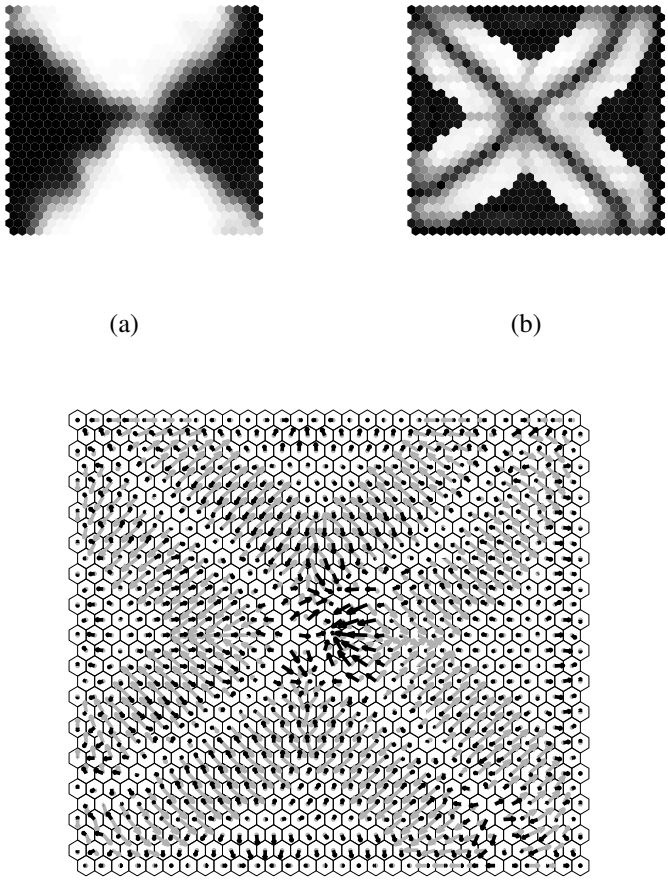


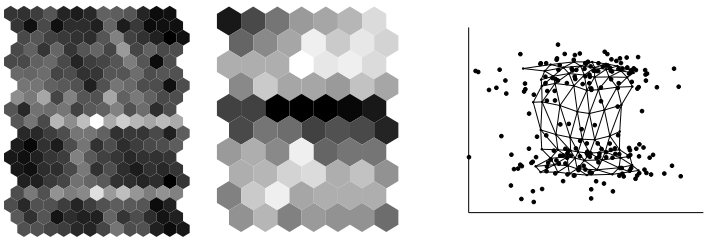
Figure 4.13: Artificial Data SOM (non-linear relationship): Component planes: (a) x_1, (b) x_2, (c) x_3; (d) length of difference vector (contrast plot), (e) scatterplots and distribution of x_1, x_2, x_3, (f) groups of component planes $\mathbf{M}^{(\mathfrak{S}_{\text{uniform}})}$ vs $\mathbf{M}^{(\mathfrak{S}_{\text{prob\_dep}})}$



(a) (b)

(c)

Figure 4.14: Artificial Data SOM (XOR-like relationship): (a) Component plane x_{11}, (b) length of difference vector (contrast plot), (c) Groups of component planes $\mathbf{M}^{(\mathfrak{S}_{\text{uniform}})}$ vs $\mathbf{M}^{(\mathfrak{S}_{\text{prob\_dep}})}$



(a) (b) (c)

Figure 4.15: Fracture Optimization, $\{7 \times 10\}$ SOM: (a) U-Matrix, (b) P-Matrix, (c) PCA with projection of map units

#	Variable Name	Type	Description	Group
1	Proppant in Formation	param	Amount of propp. pressed into formation (lbm)	G2
2	Average Pressure	geo	Pressure in Reservoir (psi)	–
3	Average Rate	param	Rate of pumping into formation (bbl/m)	G2
4	Pad Fluid Vol. Pumped	param	Total volume required to pump proppant (bbl)	G2
5	Total Volume Pumped	param	Total of fluids pumped (bbl)	G2
6	Produced Gas	out	Quantity of gas obtained (MSCF)	–
7	NetPay	geo	Depth of gas reservoir (ft)	G1
8	Formation	geo	One of two ground formation types	G1
9	Proppant Type	param	One of two proppant types	–
10	Stimulation Costs	out	Total cost of operation ($)	–

Table 4.1: Description Fracture Optimization data set variables

4.5.2 Dual Gradient Fields on petroleum engineering data

The combination of Gradient Fields to show how groups of component planes influence the overall clustering structure with a $\{7 \times 10\}$ SOM trained on the Fracture Optimization data is discussed. The Fracture Optimization data set used for this experiment consists of 199 samples in 10 variable dimensions, where each sample corresponds to a well for gas drilling. The variables are described in more detail in Section 4.5.2, as they are categorized into several groups and these groups are then analyzed. To present an overview of this data set, a medium sized map trained on this data set is discussed and shown in Figure 4.15. In Figure 4.15(a), the U-Matrix is given, which shows a gap that separates the upper from the lower part. The P-Matrix in Figure 4.15(b) shows that the map is populated uniformly

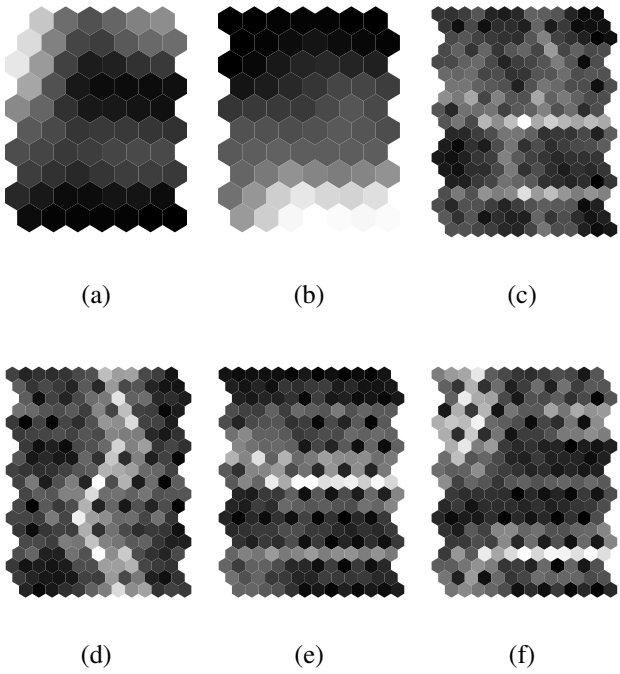
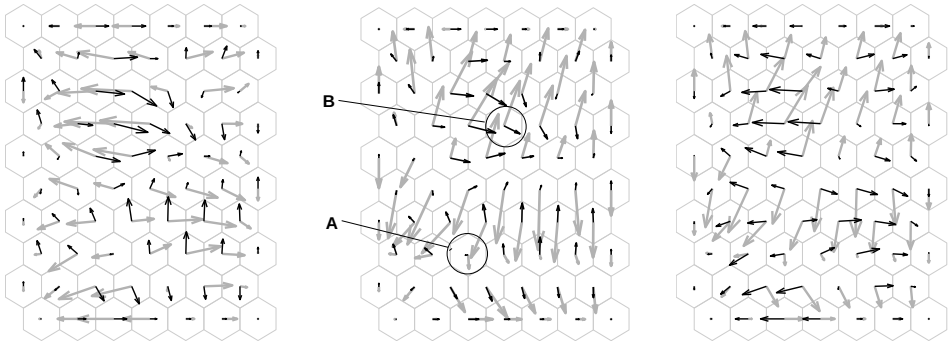


Figure 4.16: Fracture Optimization, $\{7 \times 10\}$ SOM: (a) Component plane "produced gas", (b) component plane "stimulation costs"; (c) U-Matrix, (d) U-Matrix of $\mathbf{M}^{(\mathfrak{S}_{\text{geo}})}$, (e) $\mathbf{M}^{(\mathfrak{S}_{\text{param}})}$, (f) $\mathbf{M}^{(\mathfrak{S}_{\text{output}})}$

except for the horizontal gap in the middle. The PCA plot of the data set and the map units in Figure 4.15(c), which explains 64% of the variance present in the data set, shows that two clusters are present. The variable dimensions come from 3 types of sources: (1) Geological factors that describe properties mainly determined by the choice of the geographic position; (2) Engineering parameters set during the gas pumping process; and (3) Output parameters that assess the success of the operation: "Stimulation Costs" and "Produced Gas". The variables of this data set are described in Table 4.5.2. The data is gathered during three steps, each corresponding to one of these groups. First, the position where to build the well after checking the geological data is selected; then, the engineering parameters are determined and proppant and other fluids are pumped into the earth; and finally, after the gas has been obtained, the output variables can be assessed. The index sets are denoted as $\mathfrak{S}_{\text{geo}}$ (3 dimensions), $\mathfrak{S}_{\text{param}}$ (5 dimensions), and $\mathfrak{S}_{\text{out}}$ (2 dimensions), respectively. A $\{7 \times 10\}$ SOM on this data set has been trained in order to find out how these groups of variables depend on each other and how they decompose the clustering structure. Note that the output variables are used



(a) (b) (c)

Figure 4.17: Fracture Optimization, $\{7 \times 10\}$ SOM, Dual Gradient Fields (the first group is indicated by black, and the latter by grey vectors): (a) $\mathbf{M}^{(\mathfrak{S}_{\text{out}})}$ vs $\mathbf{M}^{(\mathfrak{S}_{\text{geo}})}$, (b) $\mathbf{M}^{(\mathfrak{S}_{\text{out}})}$ vs $\mathbf{M}^{(\mathfrak{S}_{\text{param}})}$, (c) $\mathbf{M}^{(\mathfrak{S}_{\text{geo}})}$ vs $\mathbf{M}^{(\mathfrak{S}_{\text{param}})}$

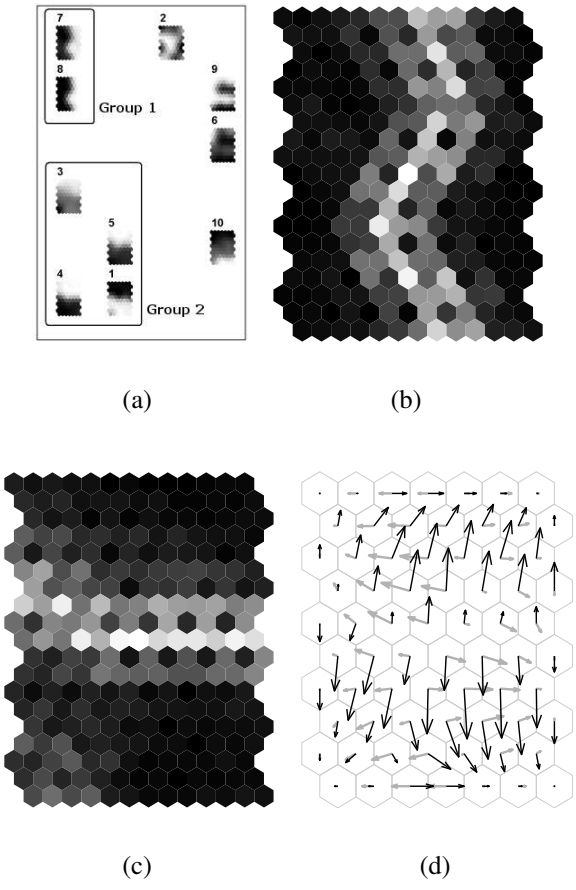
in the same way as the other ones for training since the SOM is an unsupervised learning method. What is intended to do is thus related to canonical correlation analysis rather than to regression. Data analysts are concerned with measuring the impact of the choice of the well's position or the fine-tuning of the engineering parameters on the output.

Figures 4.16(c)–(f) show the U-Matrices for the SOM and the sub-SOMs $\mathbf{M}^{(\mathfrak{S}_{\text{geo}})}$, $\mathbf{M}^{(\mathfrak{S}_{\text{param}})}$, and $\mathbf{M}^{(\mathfrak{S}_{\text{out}})}$. From these, an impression of the the cluster boundaries can be gained. The engineering parameters seem to divide the map horizontally, while the geological factors are responsible mainly for a vertical boundary. The output parameters are the most interesting ones, since the aim is to explain which regions of the map correspond to desirable outcomes, i.e. where wells with low costs and high produced gas are located. Figures 4.16(a),(b) show the component planes for these variables. It can be seen that the costs are high in the lower part of the map, and low in the upper regions. The gas production is high for wells that are mapped to the left border and slightly below the center of the map. Thus, the most desirable position for wells is the upper left corner with both low costs and high output. Figures 4.17(a)–(c) show the pair wise Dual Gradient Fields of the three groups. For the arrows pointing in different directions for most parts of the map, the underlying variable groups are likely to be statistically independent and explain different parts of the overall clustering structure, which will be discussed in the next example. Figure 4.17(c) shows $\mathfrak{S}_{\text{geo}}$ and $\mathfrak{S}_{\text{param}}$, where the arrows are orthogonal in most cases. This information can be exploited in order to improve the fracture optimization process. The horizontal position of the

sample projected onto the map is apparently determined by the geological factors since the black arrows are parallel to the horizontal axis. The vertical position corresponds to the engineering parameters. Thus, once the well is physically built, the geological factors cannot change anymore, and the horizontal position on the SOM describes a constraint for the effect of tuning the engineering parameters. It is desirable to shift a well towards the upper left corner to optimize output and costs. The lengths of the arrows correspond in how much a parameter has to be changed to achieve a change in the node that the sample is projected to. For example, Figure 4.17(b) shows $\mathfrak{S}_{\text{out}}$ (black) and $\mathfrak{S}_{\text{param}}$ (grey). Suppose a well is mapped to position "A", where the black arrow is short, while the grey arrow is long, thus moving one node up would require changing the engineering parameter by a large amount, while resulting only in small differences in output. For position "B", the arrows are approximately orthogonal, thus changing the parameters would only have marginal effects on the output since the gradients do not indicate that there is a change in output vertically.

As opposed to grouping the variables according to an external characteristic such as the classification of the source of the measurement, the variables can be investigated for their similarity. This can be done by examining their correlation or by clustering of component planes, as described in Section 2.3.3. Using the interactive approach of reorganizing component planes according to their correlation, a clustering is depicted in Figure 4.18(a) on the $\{7 \times 10\}$ SOM trained on the Fracture Optimization data set. By manual selection, two groups of variables have been singled out, the first one consisting of 4 variables in the lower left corner of the reorganized plane, the second one with 2 components in the upper left part. The index sets are $\mathfrak{S}_{\text{G1}} = \{7, 8\}$ and $\mathfrak{S}_{\text{G1}} = \{1, 3, 4, 5\}$; which variables these indices refer to is shown in column "Group" of Table 4.5.2. The U-Matrix can be applied to show the decomposition of local dissimilarities of map units on any of the reduced SOMs, as depicted in Figures 4.18(b)–(c).

Figure 4.18(d) shows the results for a Gradient Field visualization with kernel width $\sigma = 2$. It can be seen that the variables $\mathfrak{S}_{\text{G2}}$ are responsible for the horizontal division of the map, while $\mathfrak{S}_{\text{G1}}$ split the map vertically. This was also visible in the U-Matrices in Figures 4.18(b)–(c), but here it is combined in a single plot. Orthogonal angles between arrows from different groups, as in Figure 4.18(d), indicate that the groups are almost independent, which is not surprising as the groups have been selected based on low correlation. Using this method, however, helps to understand how the map is built based on the correlation structure. The vertical position of a data sample on the map is explained mostly based on the values of the variables of $\mathfrak{S}_{\text{G2}}$, while the horizontal position is determined by the values of $\mathfrak{S}_{\text{G1}}$.



(a) (b)

(c) (d)

Figure 4.18: Fracture Optimization, $\{7 \times 10\}$ SOM: (a) Clustering of component planes, (b) U-Matrix of G1, (c) U-Matrix of G2, (d) Gradient Field S_{G1} vs S_{G2} with $\sigma = 2$

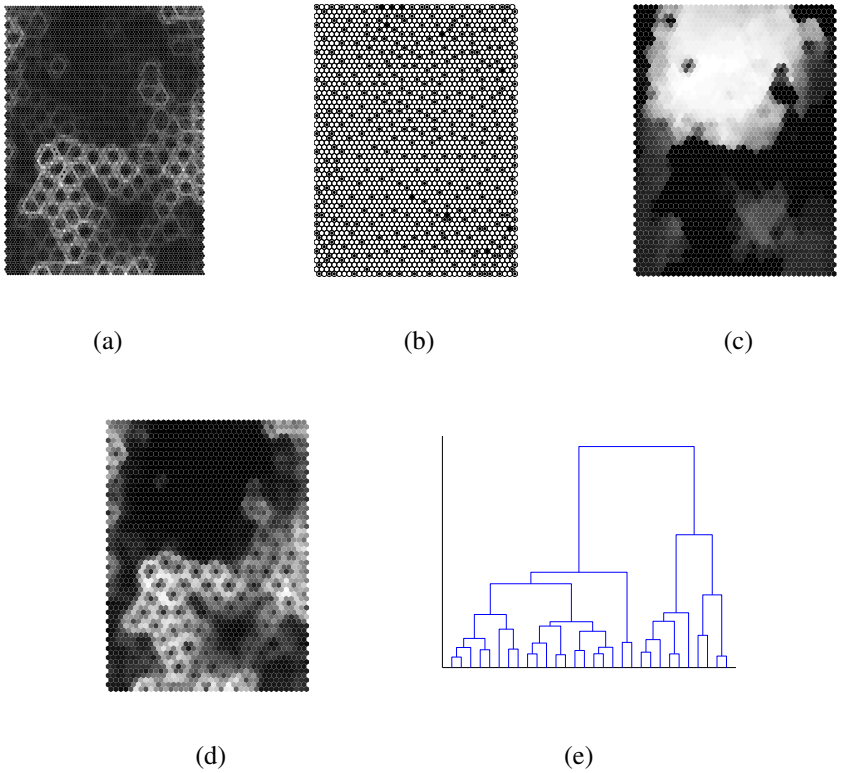


Figure 4.19: Ionosphere data set, $\{40 \times 60\}$ SOM: (a) U-Matrix, (b) hit histogram, (c) P-Matrix, (d) U*-Matrix, (e) Dendrogram of Ward's clustering on map codebook

4.6 Analysis

In this section, systematic guidelines are developed for applying Gradient Fields for both the investigation of overall clustering structure and inspection of groups of similar variables. The Gradient Fields are shown in context with other SOM visualization techniques.

4.6.1 Analysis of clustering structure

The subject of investigation in this section will be a $\{40 \times 60\}$ SOM trained on the Ionosphere data set. The basic visualization techniques are shown in Figure 4.19(a)–(d), where the U-Matrix, hit histogram, P-Matrix, and U*-Matrix are depicted. From the U-Matrix, the upper half of the map appears to be very close in input space, while the lower part is strongly fragmented. The hit histogram shows

the map is sparsely, but evenly populated by data samples. The P-Matrix shows that the upper part is indeed very dense, while the lower part is not. The U*-Matrix shows essentially the same as the U-Matrix, as the region where density is high also has low U-Matrix values, and thus no smooting occurs. According to these visulization techniques, the upper part is very dense and most likely a cluater, while not too much can be learned about the structure in the lower part.

Several visualization methods have been presented in this thesis that can identify clustering or density structure at different levels of detail, adjustable by a parameter. These are hierarchical clustering methods, where the number of clusters can be adjusted directly; the SDH visualization, where a parameter determines how much smoothing is applied to a hit histogram-like visualization; the Graph method, where the threshold radius adjusts the level of density to be displayed. All of these are intended to show the clustering or density of the map. The Borderline method is compared to these three methods at comparable levels in Figure 4.20. The rows refer to visualizations at comparable levels of granularity. The first row provides the most local view, while the last one shows the visualizations at a global view. The first column shows Ward's hierarchical clustering, the dendrogram of which is shown in Figure 4.19(e). The second column shows SDH, the third the Graph based method introduced in Chapter 3, and the last column shows the Borderline method. Figure 4.21 shows the Gradient Fields for the same radius values. These figures are larger than the ones for the Borderline method, as the arrow heads need to be recognizable. The Borderline method is better suited for large maps when depicted on a small space.

In the first row in Figures 4.20(a)–(d), a very coarse view is presented, which shows many clusters and many boundaries. Here, the local structure is investigated. Ward's clustering with 8 clusters shows that the upper part is still a cluster while the lower part of the map is fragmented. The cluster with the darkest color is split into two regions of the map, one part is to the left of the center, the other at the right border, hinting at topology violations. The SDH at $s = 25$ shows a highly fragmented view, where there are small islands across the map, with the exception of the center and lower left part of the map, which is free from any islands. The Graph based method shows the density of the upper part most impressively, as there are almost no lines in the lower part. The U-Matrix that is shown beneath the lines shows the interpolating units in the center and lower left parts of the map. The Borderlines method at $\sigma = 2$ shows several separations and boundaries, mainly in the lower part of the map, but no overal conclusions can be deducted.

The next two rows in Figures 4.20(e)–(h) and 4.20(i)–(l) show the same visualization methods with parameter values that result in more global views of the map, and could be described as "zooming out". In Ward's clustering, the clusters in the center and lower right are joined and form a big cluster, with the cluster

close to the left border and the one in the upper half of the map forming the other big clusters. For SDH, the picture is similar, as a big island appears in the upper part, a smaller one at the lower left border, and a last one that is fairly separated from the rest in the lower right part of the map. The Graph method at higher radii shows that the density in the upper part still dominates, while small clusters emerge at the left border and the lower right part. The Borderlines visualization shows that the boundary that ranges from the lower left to the center, where the upper part is separated from the lower part. The upper part shows longer border lines, while the lower right part appears like a plateau. This phenomenon is due to the fact that the arrows in the upper, dense part point away from the boundary in the middle, while the arrows in the lower part do not have a cluster center where they all point to, as they are equally repelled from all sides.

Finally, the most global view is shown in the last row in Figures 4.20(m)–(p). The hierarchical clustering shows that the remaining clusters contain the big one in the upper part of the map, combined with the one on the lower left border of the map. SDH shows a similar picture, but it is not so clear where the left part actually belongs. The Graph method shows that the lower right part is connected to the upper and left parts to some degree, leaving the interpolating units empty. The Borderline method finally shows the horizontal separation in the middle, and to a lesser degree a separation in the lower left part.

Comparing the results from the different methods, several observations can be made. Herarchical clustering does not recommend any setting for the right level of clusters. Given the number of clusters, the algorithm will always present a result regardless of that number of clusters is supported or not. Getting the number of clusters right can be achieved by inspecting the dendrogram, as shown in Figure 4.19(e), and choosing a level where the distance to the next join is large both in directions up and down. Another option is to evaluate the clustering with an index such as the DB-Index described in Section 2.3.4. However, the DB-Index, by the way it is calculated, favors several clustering methods over others, i.e. Ward's linkage over single linkage, and thus requires deep understanding of the way it behaves and what it penalizes. When visualized on the map, it does not tell anything about whether the boundaries actually indicate a strict separation or represent a separation between to units that are actually very similar, as the Gradient Fields and Borderline methods do.

SDH is based on the distribution of the data samples, as opposed to codebook clustering. Therefore, regions are visually depicted as islands if their codebook vectors are close to many data samples in feature space. The resulting island visualization shows the clusters, and also where there are no clusters, i.e. interpolating units or regions, for example, the separating canal between the main islands in Figures 4.20(n),(n).

The Graph method is also based on the data samples and the codebook. Above

all, it shows the density structure, and how the clusters relate to each other even if not adjacent. This latter feature is unique to this method. Ward's clustering identifies disjoint regions on the map as one cluster, which can be seen in Figure 4.20(a) as the cluster of the darkest color. These regions are not identified as adjacent at all by the Graph method, such that the two methods provide contradicting results. As the Graph method is based on the data samples, and not only on the proximity of the codebook, the Graph method is more credible in this respect. Ward's linkage tends to identify equally large clusters, even if the data does not support this, in this case regions of interpolating units, which are treated equally to units within dense clusters.

The Borderlines method shows roughly the same cluster structures as the other methods. It differs in that the gradual transitions are most visible here even when not varying the parameter, especially when compared to crisp clustering such as Ward's linkage. The Borderlines method shows the strength of individual boundaries. For example, in Figure 4.20(p), the region along the right side ranging to the upper part of the map shows a very smooth transition, even though these areas have been identified as clusters previously. It is similar to the U-Matrix in this respect, which does this at the most granular level. The boundaries are very similar at low radius levels, c.f. Figures 4.20(d) and 4.19(a). When compared to SDH, the Borderlines methods are more stable in that clusters do not disappear and reappear at different parameter settings, such as the island on the left border in Figures 4.20(j), (n). While the overall picture of the Borderlines changes strongly when the parameter is modified, cluster boundaries either become stronger or disappear, but the clustering structure is mostly left unchanged.

The Graph method is different to the Borderlines method in that it solely visualizes data density. The Gradient Field and Borderlines do not emphasize density, and are not based on data samples, but on the codebook only. Density is implicitly visualized if it is represented by the codebook, and is depicted as large regions full of long arrows, pointing inwards, or long border lines. What the Gradient Field and Borderlines methods are not able to visualize is regions with non-adjacent but similar prototype vectors. The Gradient Field and Borderlines methods are most helpful for parameter settings that is neither too local nor too global. A good choice is one sixth of the shorter side of the map, closest to Figure 4.20(l) in this case. It provides more information than the U-Matrix, but the view is not too global in that the transitions between clusters become too blurred.

To sum up, the major characteristics of the Gradient Field method can be stated as follows:

- The level of granularity of the Gradient Field is comparable to a hierarchical clustering. A difference is that hierarchical clustering results in crisp clusters and strict boundaries while the Gradient Field does not.

- The boundaries of Gradient Field visualizations at low radius levels resemble the local boundaries visible with the U-Matrix.
- The Gradient Field is best used in combination with other density and clustering visualizations, such as U-Matrix, hit histogram, P-Matrix, U*-Matrix, hierarchical clustering, and Graph based methods. It is best applied if the radius is varied interactively in order to learn about the clustering strucure at desired levels of detail.
- The Gradient Flow method is comparable to the U-Matrix in that it compares units to its neighbors, where neighbors includes a wider area depending on the radius. However, it cannot be used to identify separated clusters or similar data samples projected to different parts of the map due to topology violations, which the Graph method, for example, can be used for.
- A good choice for the starting point of the radius σ is one sixth of the shorter side of the map.
- Increasing the radius results in zooming out to a more global view, resulting in less visible clusters. By decreasing the radius, the view is becoming more granular, showing finer cluster boundaries.
- The Borderline method is a viable alternative to the Gradient Fields that essentially shows the same thing. For larger maps where the individual arrows are not easily recognized, the Borderlines method is even better suited to show boundaries.

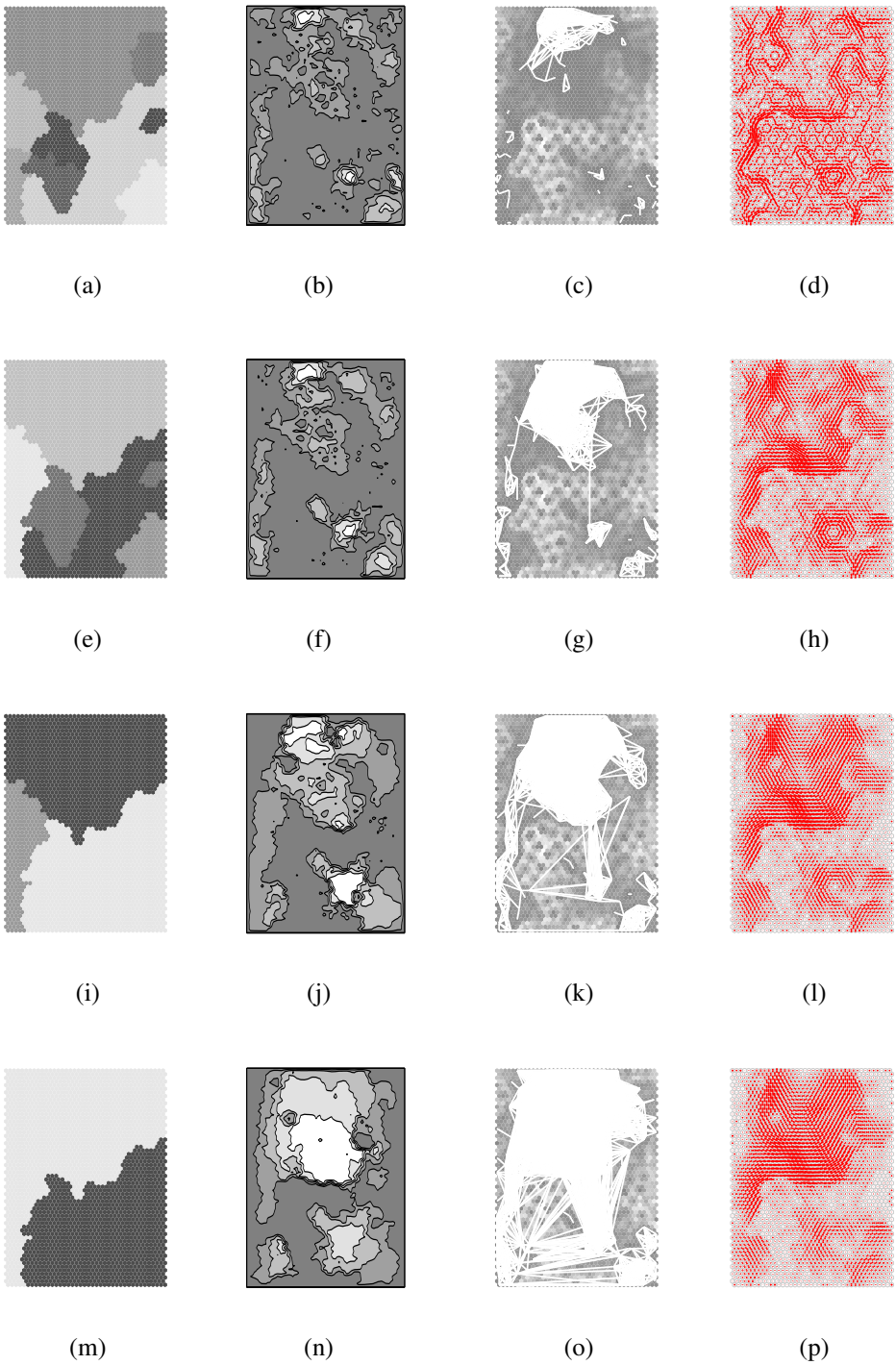
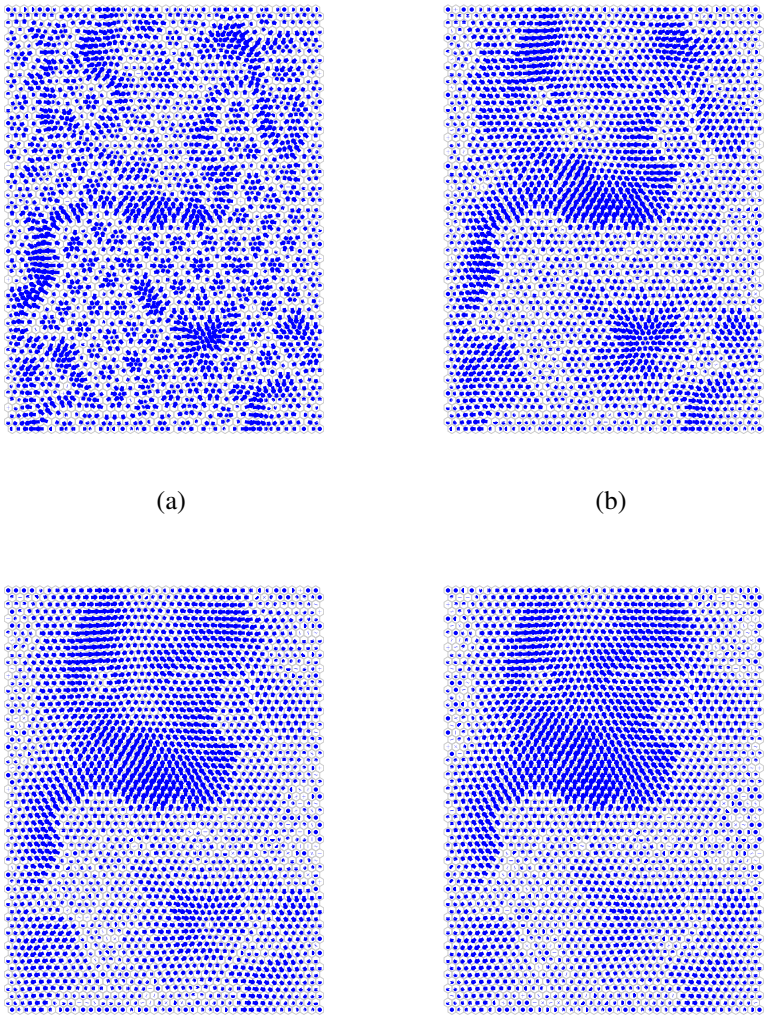


Figure 4.20: Ionosphere data set, $\{40 \times 60\}$ SOM: Comparison of Ward's clustering, SDH, Graph based method (on top of U-Matrix), and Borderline visualizations; (a)–(d) Granular view, high number of clusters, (e)–(h) less granular view, less clusters, (i)–(l) almost global view, few clusters, (m)–(p) global view, low number of clusters; parameters for Ward (a), (e), (i), (m): 8, 5, 3, 2 clusters; SDH (b), (f), (j), (n): 25, 50, 200, 1000; Graph (c), (g), (k), (o): 1.0, 1.75, 2.5, 3.5; Borderline (d), (h), (l), (p): 2, 4, 6, 8



(a) (b)

(c) (d)

Figure 4.21: Ionosphere data set, $\{40 \times 60\}$ SOM, Gradient Fields: (a) $\sigma = 2$, (b) $\sigma = 4$, (c) $\sigma = 6$, (d) $\sigma = 8$

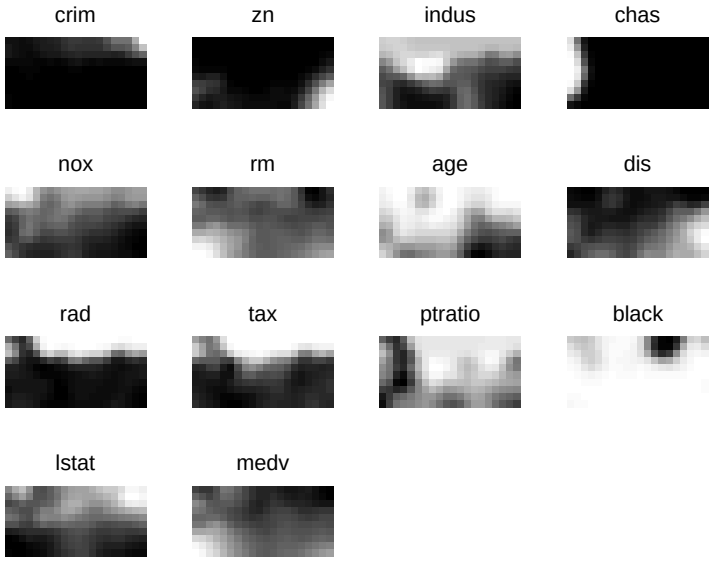
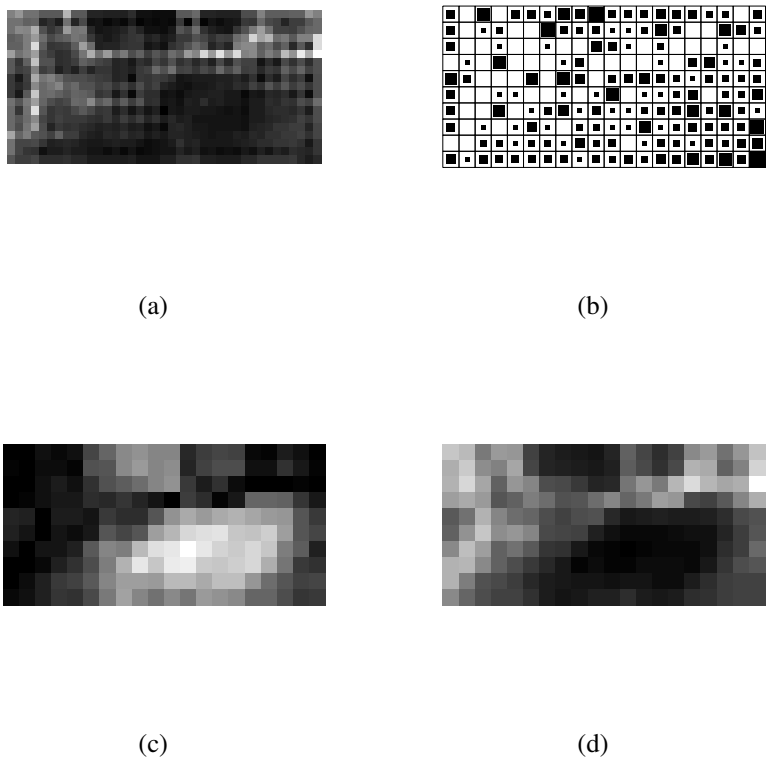


Figure 4.22: Boston housing data set, rectangular $\{10 \times 20\}$ SOM: Component planes

4.6.2 Analysis of groups of component planes

In this section, the Boston housing data will be used to investigate the similarity of groups of variables, and compared to the correlation visualization methods described in Section 2.3.3. The component planes of the $\{10 \times 20\}$ SOM are shown in Figure 4.22, and the most common visualizations are shown in Figure 4.23. This data set consists of 506 samples in 14 variables. The U-Matrix shows that there is a cluster in the center ranging to the lower right corner of the map. On the left and upper sides of the map there are cluster boundaries, but only two small clusters in the upper part of the map. The hit histogram shows the data is fairly evenly distributed across the map, with a small gap between the center region and the left and upper parts, where the U-Matrix values are high. The P-Matrix supports the finding from the U-Matrix that there is one big cluster in the center and lower right, with two minor clusters on the top. The U*-Matrix does not provide any additional information to the U-Matrix, as this is not a sparse map.

In this section, the component planes of the codebook are regarded as a valid proxy for the actual data, i.e. the codebook is deemed to be sufficiently similar to the data set such that findings made by inspecting the codebook can be transferred



(a) (b)

(c) (d)

Figure 4.23: Boston housing data set, rectangular $\{10 \times 20\}$ SOM: (a) U-Matrix, (b) hit histogram, (c) P-Matrix, (d) U*-Matrix

to the data set. All the methods compared here disregard the data set. Conceptually, the component planes are in the center of interest. Groups of component planes are a method for contrasting clusters of variables and investigating their influence on the map's layout. To get an impression of how the component planes are correlated, component plane reordering and clustering is performed. Then, the groups of component planes visualization is contrasted with the metro visualization, which is related as it also shows the gradient, or direction of change, of similar component planes.

The component plane reordering approach has been demonstrated in Section 2.3.3. It rearranges the component planes by their similarity based on the codebook in order to create clusters of variables that are most likely correlated. In Figure 4.24, the reordered component planes are shown, grouping variables such as "rad" and "tax", "dis" and "age", and "medv" and "rm". "crim" is placed in

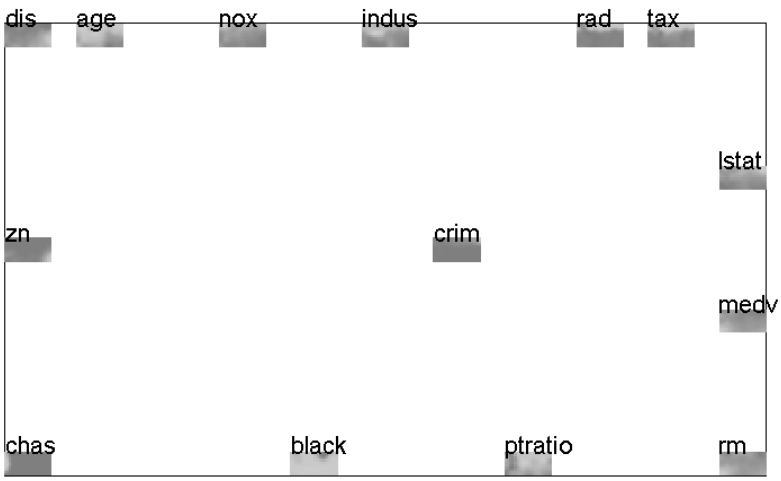


Figure 4.24: Boston housing, rectangular 10×20 SOM: Reordered component planes

the middle such that it could be correlated to any other variable.

Similarly, the component planes can be clustered directly by a clustering algorithm. The dendrogram of a Ward's linkage of the component planes is shown in Figure 4.25. Two big clusters can be distinguished, and the variables grouped at the lower levels is very similar to the reordering approach. By selecting clusters of components, their influence on the map structure can be investigated with the groups of component planes method, which does not by itself prefer any grouping. As described in Section 4.5.2, the groups to be contrasted have to be pre-selected. However, the results from the metro visualization technique can be compared to the groups of component planes visualization.

The metro visualization [61], as depicted in Figure 4.26, is a visualization method that represents component planes as piece-wise linear curves along their gradient. These curves are aggregated according to a metric that is based on how far apart these lines are. Based on this metric, the most similar lines, i.e. component planes, are joined similar to a hierarchical clustering, and a new curve is calculated from the lines that have been grouped together. The lines for single

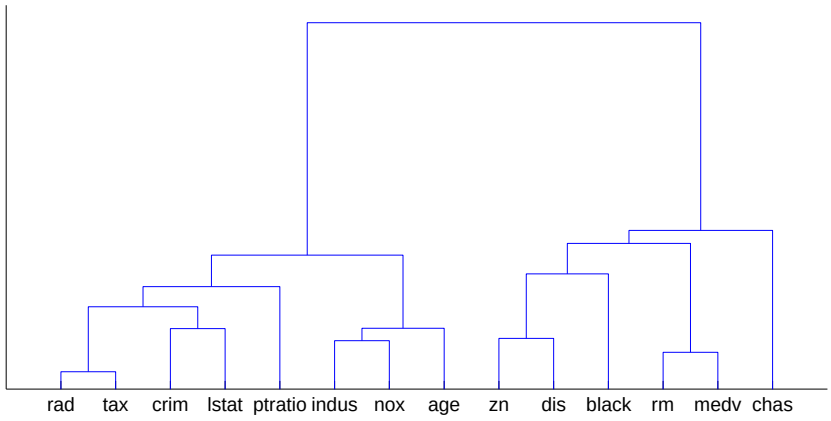


Figure 4.25: Boston housing data set, rectangular 10×20 SOM: Dendrogram of Ward's clustering performed on the feature dimensions of the codebook

component planes run along the gradient of this component plane, from its lowest to its highest point. The bundled lines show an aggregated gradient, i.e. running from a point where a correlated set of component planes has similarly low values to a region where all values are high; in case of negative correlation, the starting points may be high for one set of variables, and low for the other one, leading to low and high regions, respectively. In the example of the Boston housing SOM, 5 groups are formed, 4 of which consist of two component planes each, and the last one consisting of 6 component planes. The first group that will be investigated is the one containing "crim" and "lstat". The component planes in Figure 4.22 reveal that "crim" has its highest values in the upper right corner, and low values almost everywhere else; "lstat" looks exactly the other way around, with low values in the upper right, and high values everywhere else. There is thus a high negative correlation between these variables. The dendrogram in Figure 4.25 also supports this, as these variables are joined at a low level. The component plane reordering in Figure 4.24 shows that the two variables are not too distant, but also not directly next to each other. The metro visualization shows the line running from the center of the map to the upper right corner. It is therefore connecting the center of the low "crim" and high "lstat" values, which is also the center of the map, to the center of the high 'crim" and low "lstat" values, the upper right corner. The second group that is investigated consists of "zn" and "dis", which both have high values in the lower right corner, and lower values everywhere else. The metro line also shows this gradient, but running along the lower border horizontally as "dis" has high values there.

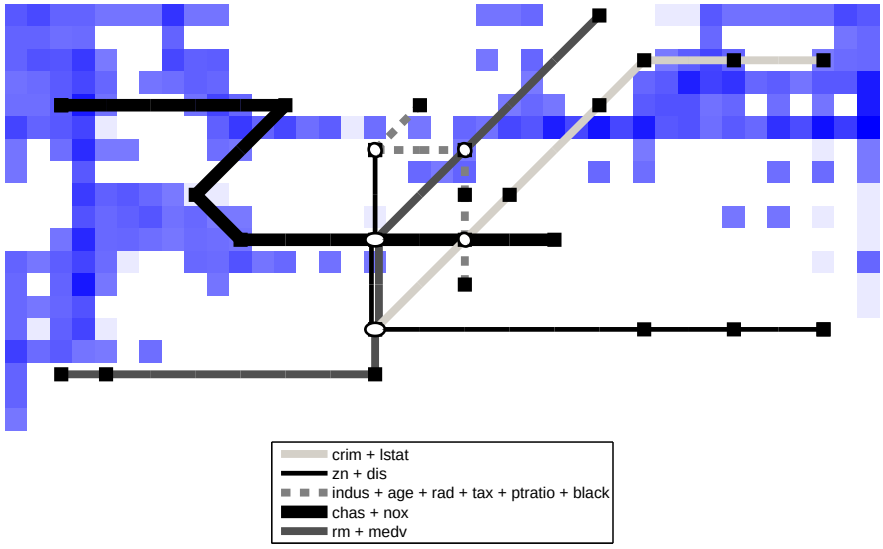


Figure 4.26: Boston housing data set, rectangular 10×20 SOM: Metro visualization on top of U-Matrix

These results can now be compared with the groups of component planes visualization in Figure 4.27(a) with a radius $\sigma = 2$. The black arrows denote the group consisting of "crim" and "lstat". It can be seen that the arrows follow the same direction as the metro line, from the upper right to the center, continuing to the lower left. The arrows also fill the areas the metro lines do not, e.g. in the lower right corner, in this case away from the high values in the upper right towards the low values in the lower right. The arrows in the left side of the map, where there is no metro line as it ends in the center, point away from the center, but are much less pronounced than the ones in the right part of the map. This is due to the lower differences in the left side of the map, where "crim" values are low and "lstat" values are high with little local differences. The group containing "zn" and "dis" is denoted by the gray vectors. The vectors in the right side point to the left, and the ones in the lower part are longer indicating greater differences. Eventually, the arrows point towards the upper part, when approaching the center of the map. This also resembles the direction of the gradient lines of the metro visualization.

In order to actually contrast the two groups, the difference of the vectors can be calculated, as shown in the contrast plot in Figure 4.27(b). It shows regions where the gradients of the two groups differ. The peak difference is slightly above and to the right of the center of the map. According to the U-Matrix, there is a

gap here. Long arrows resemble large local differences in a group of component planes. If these arrows are highly different, as measured in the contrast plot, the two groups in question contribute to the clustering structure in opposing ways. For example, if the SOM would consist only of the first group of component planes "crim" and "lstat", it would most likely have one small cluster in the upper right corner, and a larger one containing the rest of the map. A SOM of the second group would have a cluster in the lower right corner reaching to the center, and another one containing the rest of the map. The contrast plot shows where these two clusterings intersect.

A second set of variables is investigated, which are the two remaining metro lines with two component planes each. The first group consists of "chas" and "nox", the second of "rm" and "medv". Joining the first group does not receive strong support from the dendrogram, as "chas" and "nox" are joined only at the last step of the hierarchical clustering. However, visual inspection of the component planes reveals that they are similar in that they both have peak values on the upper part of the left border. "rm" and "medv" have high values on the lower left corner, reaching to the center and the lower right part of the map, and only the upper part of the map is occupied by low values. The groups of component planes visualization is shown in Figure 4.28, along with the contrast plot. Again, the arrows follow the same directions as the metro lines. Where there are no metro lines, the gradients are generally not so pronounced, resulting in shorter arrows. The contrast plot shows that the cluster structure differs mainly on the left side of the map, where the first group containing "chas" and "nox" has high differences, with a horizontal gradient, while the second group has very short arrows, as this region is very homogeneous for these two variables.

In order to summarize this section, the following observations can been made:

- Both the metro line visualization and groups of component planes show the gradients of clusters of variables. This is an advantage over correlation measures, which can compare only pairs of single variables, and not groups of them.

- Groups of component planes are a technique for contrasting sets of variables. These groups can be either based on semantic meaning or on some pre-calculation of component plane correlation. There is no built-in mechanism that provides certain sets of components.

- The metro visualization provides a strongly simplified breakdown of the subgroups of component planes, while the gradient field approach provides more details on the contrast between the groups, especially in regions where there are no metro lines.

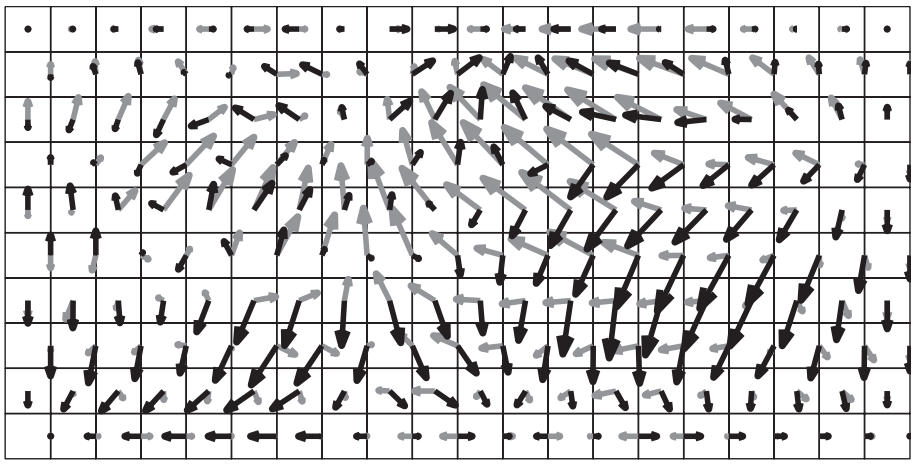
- The radius is not varied much in this approach, it is usually set to a sixth of the shorter side of the side.
- The contrast plot shows where the two groups that are compared differ in their gradients. It explains the influence of the groups on the clustering structure, showing peaks where the cluster boundaries of each group taken individually intersect.

4.7 Summary

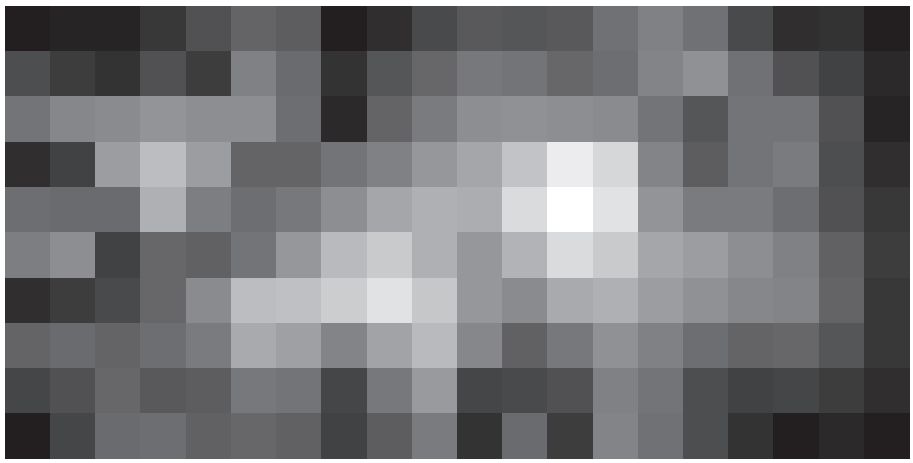
In this chapter, a visualization technique for Self-Organizing Maps with vector fields has been described that are especially aimed at professionals with engineering backgrounds. The method can be displayed either as a flow diagram where arrows point in the direction of most likely cluster centers, or as an equivalent that emphasizes at showing cluster boundaries. The former method is referred to as Gradient Field, the latter as Borderline method. The arrows in the Gradient Field method are longer close to cluster boundaries, and shorter close to the cluster centers. The lines of the Borderline are exactly as long as for the Gradient Field, only rotated by 90 degrees. A parameter is provided that determines how much smoothing is applied to the resulting visualization.

The Gradient Field and Borderline methods have been compared to clustering and density visualizations and its differences, strengths and weaknesses have been elaborated. The main difference to cluster visualizations is that the visualization is not crisp but shows the nuances of where there are stronger and weaker boundaries between clusters. Varying the radius can be used to finetune the visualization in order to show local or global clustering structures. A good starting point for the radius is one sixth of the shorter side of the map, such that the visualization is neither too local nor too global. The Gradient Flow method is similar to the U-Matrix at low values of σ. However, it is not able to tell about clusters that are separated on the map, i.e. similar data projected to different parts of the map due to topology violations, which the Graph visualization method can identify.

Furthermore, an extension to this method has been presented with the goal of simultaneously plotting multiple groups of variables to show the decomposition of the clustering structure in contributing factors. This method has also been shown to be able to be used to detect linear and non-linear dependencies between variables, given the groups of the component planes are known. Another extension, the contrast plot, can be used to show where the groups that are investigated differ most strongly, indicating at how and where they contribute to the clustering structure, and especially to the cluster boundaries.

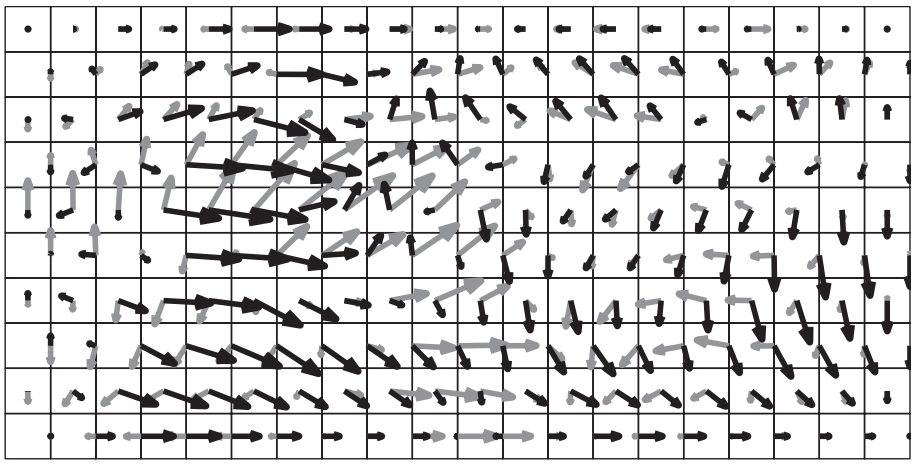


(a)

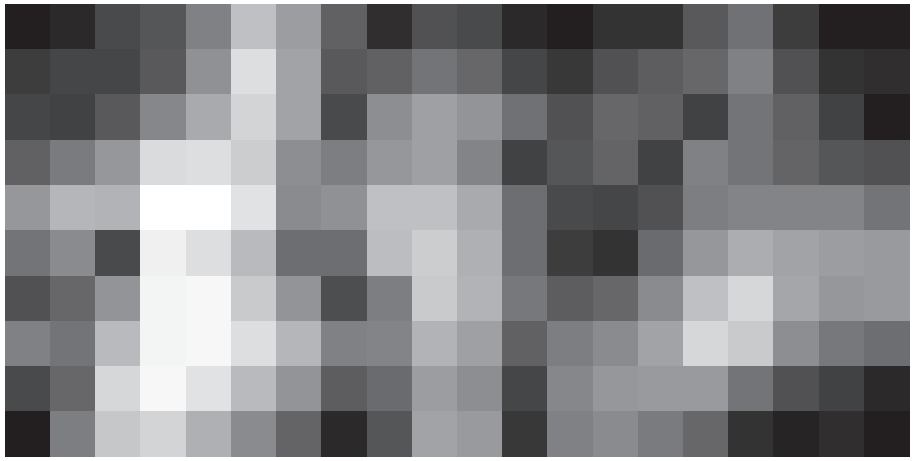


(b)

Figure 4.27: Boston housing data set, rectangular 10×20 SOM: (a) groups of component planes and (b) contrast plot: “crim” and “lstat” (black) vs. “zn” and “dis” (gray)



(a)



(b)

Figure 4.28: Boston housing data set, rectangular 10×20 SOM: (a) groups of component planes and (b) contrast plot: "chas" and "nox" (black) vs. "rm" and "medv" (gray)

Chapter 5

Decision Manifolds

5.1 Introduction

While the previous chapters have been concerned with the SOM as a visualization tool seen from the perspective of unsupervised learning and clustering, where the shape and density of the data set plays a central role, this chapter applies the SOM algorithm to a supervised setting, where each data sample has a label, which is known during training time, but hidden once training is completed. While the method that is introduced in this chapter has a different goal than the visualization methods, it is also centered around the ideas of identifying density and shape, but aims at explaining the transitions between data of different classes rather than the data set as a whole.

The method that is described in this chapter is a neural classifier algorithm for binary (two-class) problems [52]. It aims at approximating the decision boundaries locally by linear separating hyperplanes. This estimation of decision boundaries is performed where the data sample is sufficiently dense by placement of a representative that describes the decision boundary in its vicinity. The classifier, which is called Decision Manifold [73], subjects the shape of the decision boundary to topological constraints. An efficient training algorithm is provided that iteratively updates the local classifiers by moving their positions along the decision boundaries in a way similar to how Self-Organizing Maps (SOMs) are trained. One of the major conceptual contributions of this chapter is that it transfers self-organization to the domain of supervised learning, which has been tried before with mediocre results, such as the Supervised SOM [50]. This chapter thus places special emphasis on the theoretical findings, but provides experimental results by comparing the classifier algorithm to prominent machine learning techniques on a number of artificial as well as benchmark data sets.

Further, a model selection scheme is presented to estimate the correct topol-

ogy. As shown in the experiments section, the Decision Manifolds classifier is comparable in performance to modern supervised learning algorithms. Apart from fast training, the advantage of this method lies in the exploitation of the classifier topology, which is used during training to align the classifiers to achieve an ordered representation of the decision boundary, and to avoid overfitting and local minima in the placement of the local classifiers. In the last stage of the training phase, when the positions of the individual local classifiers have converged, the topology is used to fine-tune classification performance by determining the optimal voting weights.

The remainder of the chapter is based upon the presentation of the algorithm in [72, 73] and is organized as follows: As this part of the thesis deals with supervised learning, this chapter has its own related work in Section 5.2. In Section 5.3, concepts that this technique is based upon are described, namely linear classifiers and decision boundaries in general. Further, a brief introduction to topological concepts is given that are the motivation for the Decision Manifold approach. Section 5.4 outlines the training algorithm, optimization of classification accuracy by adapting the voting weights of the classifiers in the committee, and a model selection framework for estimating the topology of the basic classification algorithm. Applications to artificial and benchmark supervised learning data sets are given in Section 5.5, as well as comparisons to state-of-the-art classifiers. Section 5.6 summarizes the Decision Manifolds algorithm.

5.2 Related work

The related work section is split into two parts, as Decision Manifolds are conceptually related to two approaches: The first one is the mostly unsupervised domain of non-linear projection methods. In the second part, classification methods are discussed. The Decision Manifolds algorithm is inspired by topology preservation from the first part, while it competes with supervised learning techniques from the second part.

5.2.1 Non-linear dimensionality reduction methods

Decision Manifolds are inspired by topology preservation, hence the first part of the related work is concerned with comparable unsupervised methods usually applied in visualization settings, or for pre-processing and feeding the lower-dimensional outputs to a supervised algorithm. Some of the approaches described in this section take this a step further and perform the projection in a way that explicitly prepares the classification.

There are several methods for learning manifolds. These consider the shape of the data cloud when calculating the projection from feature to output space. Among the most important of these algorithms are Isomap [95], Self-Organizing Maps, and Locally Linear Embedding [89]. The specific features of Self-Organizing Maps have been discussed in more detail in Section 2.2.

Isomap works by initially computing a graph structure that replaces the original distance function in feature space. Each data point is a vertex in this graph, and egdes exist only for data points which are close in feature space, and are weighted by their original distance. Closeness in input space can be determined either by a k-nearest neighbors scheme or by a threshold distance. The distance between two points is then defined as the shortest path connecting the vertices of two data points. The distance thus depends on the manifold, such that its path has to follow the distribution of the data cloud. This distance matrix is subsequently subjected to a Multi-Dimensional Scaling [99] algorithm that performs the actual dimensionality reduction.

Several approaches have been made to extend the standard Isomap to a supervised setting. WeightedIso [117] performs this by increasing the distance measured through the graph structure by introducing a penalty for data samples with different labels. This is taken a step further by the Supervised Isomap [24], where the distance penalty is computed in a more sophisticated way by subjecting the Euclidean distance to one of two transformation functions, based on whether the distance is measured between samples with different labels or not. After the samples are projected, a generalized regression network [119] is performed to complete the model for non-learning set data points. Data point labels are predicted by projection and subsequent labeling through a k-nearest neighbors algorithm.

An approach that is similar to Isomap is Locally Linear Embedding (LLE). While the former finds a global geodesic mapping solution, LLE is, as its name implies, more local in its focus. It is performed by first finding the best least-squares approximation by a linear combination of a data sample by its nearest neighbors. The coefficients of this approximations are stored in a matrix, which is then used to perform a reconstruction of the data samples in the low-dimensional output space. A supervised LLE variant exists [82], which also tweaks the distance measure by introducing a penalty for distances between samples of differing classes.

The advantages of both LLE and Isomap lie in convergence, globally optimal solutions, and low numbers of parameters. LLE is also computationally very efficient, while Isomap in its original form has complexity issues, which have been solved with more recent developments [30, 120, 92]. The problem of global parameterization of the resulting manifolds has also been addressed in various ways, such as through the generalized regression network mentioned above, or by global coordination methods [85].

The fundamental difference between both Isomap and LLE and the Decision Manifolds method is that Supervised Isomap performs the classification after the projection into a subspace that is of a lower dimension than the original feature space has occurred. The topology-related part of these algorithms is thus utilized in the projection. Decision Manifolds perform the separation in the original feature space. The topology-related part, which is inspired by how SOMs arrange the prototype vectors, is restricted to positioning the separating hyperplanes which ultimately perform the classification.

The similarities between Self-Organizing Maps and Decision Manifolds lie generally in the algorithm, as only the principle of aligning vectors within some measure of neighborhood is borrowed. The algorithmic similarities will become obvious in Section 5.4.1. However, the original SOM and Decision Manifolds differ fundamentally as the SOM is purely an unsupervised visualization method, while Decision Manifolds are a pure classification technique.

5.2.2 Supervised learning methods

Supervised learning algorithms [34], such as Support Vector Machines [107], Random Forests [13], k-Nearest Neighbors, and Decision Trees [14], are the primary methods that the Decision Manifolds algorithm is competing with. Some of these algorithms have been described in Section 2.1.2. Each classifier estimates the separating manifold in some way, which lies between adjacent regions with opposing labels. This estimation can be either explicit, where the shape of the decision boundary can be extracted, like SVMs, or implicit, where it is not easily possible to find a representation of the decision boundary, like with k-Nearest Neighbors and Random Forests.

There has been a lot of effort in the past two decades to design and implement piecewise linear classifiers, starting with the work of Sklansky [93]. In this approach, a number of hyperplanes is created and the feature space is split into partitions. However, the algorithm has severe complexity problems, as the number of partitions grows exponentially with the number of hyperplanes, as opposed to linear complexity in terms of number of hyperplanes with Decision Manifolds.

A lazy learning algorithm that is conceptionally similar to the Decision Manifolds approach is proposed in [11], where for each pattern that is to be classified k training samples from its vicinity are selected and a classifier is trained on this data set. The results have been shown to be very good in terms of classification performance, however, computationally this approach is very expensive, as a new classifier has to be trained with every pattern to be classified. The Decision Manifolds algorithm is similar in terms of classification accuracy, but is not a lazy learner as it takes advantage of local representatives that cover an area of the feature space, thus speeding up the learning and classification process consid-

erably. In Section 5.4.1, the non-trivial task of training and positioning these local representatives is elaborated.

As Decision Manifolds consist of several local classifiers that perform the actual classification task by a voting scheme, ensembles of classifiers [16] are related to this technique. Mixtures of local experts [38] train a committee of multi-layer perceptrons in a competitive learning approach where each MLP covers a region of the feature space. The final classification is performed by a selector that assigns the samples to be classified to the local experts. Recently, ensembles of biased classifiers [44] have been introduced to increase classification performance by training several classifiers that minimize the error each for a different class.

SVMs deserve special attention in this context, as the shape of the decision boundary is directly influenced by the kernel function. In case of linear kernels, the boundary is simply a hyperplane, while in the case of e.g. a cubic kernel and a two-dimensional input space, the decision boundary is a cubic polynomial. The Decision Manifolds method differs in that it approximates this decision boundary with a number of hyperplanes, which are connected by a static topology. This topology is an input parameter in the same sense as the kernel function is an input parameter for SVMs. As the dimension of the discrete topology has to be estimated, a model selection technique is proposed in Section 5.4.3 that applies a linear dimensionality reduction method, but not for actual pre-processing of the data samples, but for coming up with rough estimates for candidate topologies.

5.3 Elementary Concepts

In this section, several concepts that the Decision Manifolds algorithm is based upon and which will serve as building blocks are discussed. Further, the relevant notations used in the later sections are defined. Section 5.3.1 describes general properties of linear classifiers that are used as the building blocks of the algorithm. As a motivation for Decision Manifolds, decision boundaries in general, their topological properties, and linear approximation are described in Section 5.3.2.

5.3.1 Linear Classifiers

In this section, the notation for linear classifiers which is required for the Decision Manifold method is described. What linear classifiers generally have in common is that their result are separating hyperplanes. These can be extracted and are used by Decision Manifolds. Data samples $\mathbf{x}_i \in \mathfrak{R}$ with binary labels $y \in \{-1, +1\}$ are presented to the classifier algorithm. The whole data set without the labels is written as matrix X, where rows correspond to patterns and columns to features, and a vector of labels $\mathbf{y}$. After training the classifier, the normal vector $\tilde{\mathbf{w}}(X, \mathbf{y})$

to the separating hyperplane pointing in the direction of class $+1$ is obtained. As the affine hyperplane is characterized by the pair of its normal vector and an arbitrary point that lies on this hyperplane, homogeneous coordinates are adopted that include the offset- or bias-term in an artificial coordinate in the first position of the vector to avoid the cumbersome notation of pairs of vectors and points. In the rest of this chapter, homogeneous coordinates are referred to by a tilde over the letter symbolizing vectors and matrices, which are obtained by insertion of "1" as a first coordinate. Non-homogeneous coordinates are written without the tilde, and refer to the vector without the bias term.

Using this notation, classification of datum x is performed by estimating label $\hat{y}$, formally

$$\hat{y}(\mathbf{x}) = \text{sign}(\tilde{\mathbf{x}} \cdot \tilde{\mathbf{w}}) \tag{5.1}$$

Rosenblatt's Perceptron [84] is a well-known example for an iterative classifier that starts with a randomly initialized hyperplane, which is then updated as the data vectors are presented during several epochs. The Perceptron algorithm has been shown to converge to an optimal solution with no misclassification in case the data set is linearly separable. In this chapter, however, the Moore-Penrose Pseudoinverse is used, where the separating hyperplane can be computed in a single step, thus not requiring multiple iterations. This way of calculating the normal vector is defined as

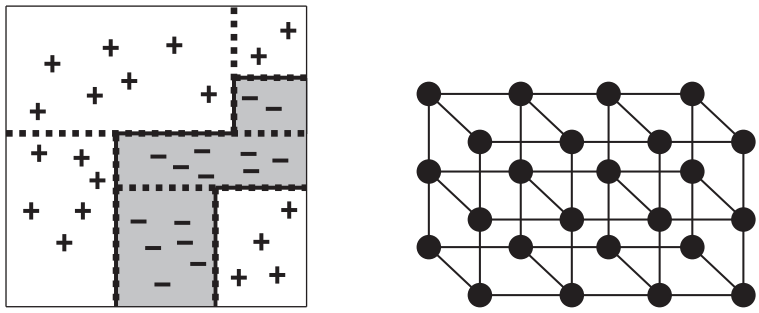
$$\tilde{\mathbf{w}}(X, \mathbf{y}) = (\tilde{X}^\top \tilde{X})^{-1} \tilde{X}^\top \mathbf{y} \tag{5.2}$$

This method solves the linear least squares problem. One drawback is that the results deteriorate in case the data distribution is skewed or contains outliers. However, it is deterministic and computationally cheap, so it will be used in most of the experiments.

Another choice for a classifier is the linear Support Vector Machine (SVM, [107]). This method is computationally rather expensive, and will thus only be used for performance experiments with Decision Manifolds in Section 5.5.3.

5.3.2 Decision Surfaces and Topological Considerations

Every classification algorithm explicitly or implicitly performs an estimation of a decision surface which partitions the feature space into disjoint regions that are assigned a label. Mathematically, a decision surface is a hypersurface (i.e. of dimension $D-1$, and of arbitrary shape). This decision boundary is assumed to consist of a finite number of topological manifolds, thus the possibly non-contiguous hypersurface can be decomposed into contiguous subsets. Bounded topological manifolds, i.e. ones that do not extend to infinity, can be categorized according to their dimensionality. For example, a line segment is topologically equivalent to



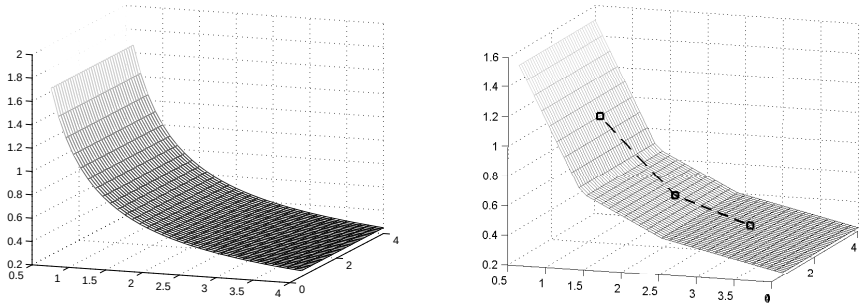
(a) (b)

Figure 5.1: (a) Example of a decision surface by decision trees, (b) graph of $\{4 \times 3 \times 2\}$ topology with connections for $A_{ij} = 1$

an arc of a circle since both are one-dimensional manifolds. Further, a topological manifold has a surface that is locally Euclidean and can thus be approximated by a patchwork of hyperplanes.

For example, Decision Trees are recursively constructed by splitting the feature space perpendicular to a coordinate axis. Figure 5.1(a) shows partitions for a decision tree, where dashed lines denote splitting decisions, and the solid line denotes the parts of the the actual decision hypersurface, which is one-dimensional in this case. In the rest of this chapter, it is assumed that the decision surface consists of a finite number of topological manifolds, thus the possibly non-contiguous hypersurface can be decomposed into contiguous subsets. In the Decision Tree example, there are two disjoint lines that delimit regions corresponding to different classes.

Figure 5.2(a) shows a possible decision boundary in three dimensions of the form $f(x, y) = 1/x$. Figure 5.2(b) shows a linear approximation of this surface by three hyperplanes aligned along a one-dimensional manifold, i.e. a curve. The approximation shows one important concept that can be exploited: As the function value is constant along the y coordinate, the patchwork can be aligned along a one-dimensional manifold. There is no need to introduce additional hyperplanes along the y-coordinate as this is already covered by the linearity of the hyperplanes that extend to infinity in the y direction. The two-dimensional decision manifold can thus be represented by a one-dimensional topology, along which the hyperplanes are patched together. If their number is increased, the manifold can be approximated at arbitrary precision. In Section 5.5, the experiments show that the required topology dimension is usually very much lower than the feature space dimension. Figure 5.2(b) also gives an outline of the goal of Decision Man-



(a) (b)

Figure 5.2: Illustration of decision surfaces: (a) decision hypersurface in 3-dimensional space, (b) approximation by 3 local hyperplanes aligned along a 1-dimensional manifold

ifolds: A set of points that represent the center of a classifier hyperplane (square markers) and their topologically correct order (dashed lines). Following this motivation, the algorithm that is described in the next section aims to estimate a local approximation of the decision hypersurface with the following characteristics:

- The decision surface is estimated reliably where data density is sufficiently high, otherwise extrapolation is used by extending the decision hyperplanes to infinity.
- The decision surface can be approximated locally by linear classifiers that are ordered along a topology that is as low-dimensional as possible.

The Decision Manifolds classifier consists of a set of local linear classifiers that are subjected to a given topology. In this chapter, rectangular topologies of various dimensions are considered. This is referred to the classifier's topology as the number of local linear classifiers along the axis in each dimension; the topology of the classifier in Figure 5.2(b) would be $\{3\}$. An example of a topology with $\{4 \times 3 \times 2\}$ local classifiers is shown in Figure 5.1(b) where each dot at the intersections of the grid's edges represents a local linear classifier. These are similar to the units of a Self-Organizing Map of according dimensionality, i.e. in this case a 3-dimensional SOM. However, the major difference to SOMs is that prototype vectors are placed where data density is high, and the Decision Manifolds method places hyperplanes between dense areas of samples with different labels.

Algorithm 1 Training Algorithm

randomly initialize $\mathbf{c}_j$ and $\mathbf{v}_j$
for $t = 1$ to T **do**
  for $j = 1$ to M **do**
    find set of samples $\mathfrak{S}_j$ assigned to classifier j
    compute weight γ_j
    if $\gamma_j > 0$ **then**
      compute separating hyperplane $\tilde{\mathbf{w}}_j$
      compute projection $\mathbf{c}'_j$ of center point $\mathbf{n}_j$ to $\tilde{\mathbf{w}}_j$
      compute normal $\mathbf{v}'_j$ from $\tilde{\mathbf{w}}_j$
    end if
  end for
  compute new $\mathbf{c}_j$ according to topology A, $\mathbf{c}'$, and $\sigma(t)$
  compute new $\mathbf{v}_j$ according to topology A, $\mathbf{v}'$, and $\sigma(t)$
end for
compute σ_{final}

5.4 Decision Manifolds

In this section, the Decision Manifolds classification algorithm for two-class decision problems is introduced. Section 5.4.1 introduces the training algorithm for this supervised learning method. Section 5.4.2 discusses how to classify unlabeled data with a trained Decision Manifold, along with optimization approaches for classification accuracy. In Section 5.4.3, a framework for estimating a suitable topology for the Decision Manifold is proposed.

5.4.1 Training of Decision Manifolds

For supervised learning, as opposed to the unsupervised setting described in Section 2.1.2, the training samples $\mathbf{x}_i$ are each associated with a class label $y_i \in \{-1, 1\}$. The Decision Manifold consists of M local classifiers, each specified by a pair of a representative point $\mathbf{c}_j \in \mathbb{R}^D$ and a classification vector $\mathbf{v}_j \in \mathbb{R}^D$ that is orthogonal to the decision hyperplane of the local classifier. Both $\mathbf{c}_j$ and $\mathbf{v}_j$ are initialized randomly. $\mathbf{c}_j$ determines the position of the classifier in feature space, while $\mathbf{v}_j$ is relevant for performing the classification. Samples to be classified are assigned to their closest representative and classified by the local hyperplane defined by $\mathbf{v}_j$. The goal of the training algorithm is to put the representatives $\mathbf{c}_j$ in the adequate positions and to let $\mathbf{v}_j$ point in the correct direction for classification. When training has finished, the Decision Manifold is described by the tuple $\{C, V, \sigma_{\text{final}}, A\}$, where A is the given topology, C and V are the sets of representa-

tives and classification vectors, respectively, and σ_{final} is the optimal classification width, which will be defined in Section 5.4.2. An example of a trained classifier can be seen in Figure 5.4(g). The thin lines delimit the areas where samples are assigned to the respective classifier, and thick lines and arrows represent the separating hyperplane. The dashed lines refer to the topology of the Decision Manifold. Note that the connected classifiers are actually next to each other. This ordering is induced by the imposed one-dimensional topology in this example, which will be explained in the course of this section. Further, an iterative algorithm is provided, that aims at placing the representatives in a way such that the predefined topology of the classifiers is preserved, i.e. that neighboring classifiers are responsible for neighboring areas of the data space. The adjacency matrix A that defines the topology, and thus the shape of the decision boundary, is assumed to be given throughout this section. Selection of a suitable topology is dealt with in Section 5.4.3, since training with a topology that does not match the shape of the decision boundary will result in suboptimal classification performance.

Training is performed for a predefined number of epochs T. The experiments have shown that this parameter is non-critical to the performance of the classifier, and $T = 5$ is assumed in the rest of this chapter. An outline of the training algorithm is given in Algorithm 1, and is referred to by the line numbers when explaining each step. In the first step of each epoch, the data samples are assigned to the closest local classifier representative (line 4), as in Equations 2.2 and 2.3. Again, the set of indices of samples mapped to the j^{th} local classifier is denoted as $\mathfrak{S}_j$, and the sets of samples and labels assigned to it as $X_{\mathfrak{S}_j}$ and $\mathbf{y}_{\mathfrak{S}_j}$, respectively. Note, that other than in the case of the unsupervised SOM, the representatives $\mathbf{c}_j$ are not true prototype vectors placed where data density is high, but rather in positions where there is a transition between two neighboring areas of different classes, which will be explained in detail in the next paragraphs. Further, the centers $\mathbf{n}_j$ of the partitions are of interest, which do not necessarily have to coincide with $\mathbf{c}_j$.

Once all the samples have been assigned, a linear classifier is trained for each partition to obtain a separating hyperplane. Since this can only be performed if samples of both classes are present, and the number of training samples will be of interest in a later step, the weighting factor (line 5) can be computed as follows:

$$\gamma_j = \begin{cases} |\mathfrak{S}_j| & \text{if } (-1 \in \mathbf{y}_{\mathfrak{S}_j}) \wedge (+1 \in \mathbf{y}_{\mathfrak{S}_j}) \\ 0 & \text{otherwise} \end{cases} \tag{5.3}$$

This means that classifier positions that are located in areas of high data density and contain data samples of both classes will receive higher weights. More complex estimates, based e.g. on the relative frequency of both classes, may be used but the simple estimate has shown to perform equally well. If $\gamma_j > 0$, a linear classifier is trained (line 7) on the subset $X_{\mathfrak{S}_j}$ from which the separating hyper-

plane $\tilde{\mathbf{w}}_j(X_{\mathfrak{S}_j}, \mathbf{y}_{\mathfrak{S}_j})$ can be extracted, as defined in Section 5.3.1. This hyperplane must pass through the convex hull of the training samples $X_{\mathfrak{S}_j}$ and thus partly lies within the Voronoi region of representative $\mathbf{c}_j$. The next step to be performed is updating the representative $\mathbf{c}_j$ such that it lies on the separating hyperplane and does not drastically change the Voronoi partition for consecutive epochs. As all the points that lie on the hyperplane are equivalent to describe it along with the normal vector, a new preliminary representative $\mathbf{c}'_j$ is computed. It lies on the hyperplane by projection of the data centroid $\mathbf{n}_j$ and stores the information for classification in the normalized vector $\mathbf{v}'_j$:

$$\mathbf{c}'_j = \pi_{\tilde{\mathbf{w}}_j}(\mathbf{n}_j) \tag{5.4}$$

$$\mathbf{v}'_j = \frac{\mathbf{w}_j}{\|\mathbf{w}_j\|} \tag{5.5}$$

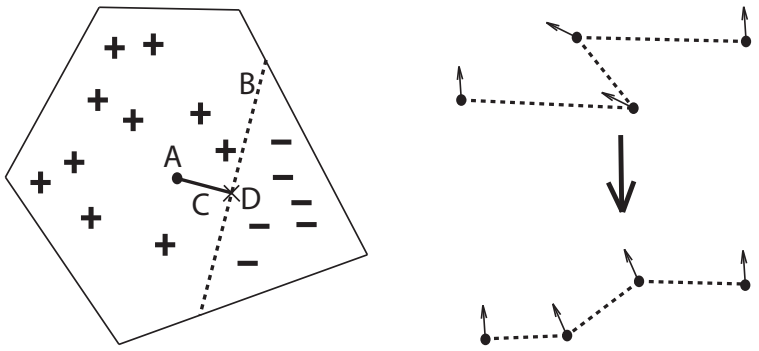
where $\pi_{\tilde{\mathbf{w}}}(\cdot)$ denotes orthogonal projection onto the hyperplane specified by $\tilde{\mathbf{w}}$. This projection ensures that the representative is placed on the decision boundary and is the point closest to the centroid of the Voronoi partition. A schematic overview of this update step (from partitioning in Voronoi sets to calculation of $\mathbf{c}'_j$ and $\mathbf{v}'_j$) is shown in Figure 5.3(a). For the set of points delimited by the thin lines that represent the borders of the Voronoi region, the separating hyperplane obtained by linear classification is shown as a dashed line ("B"). The centroid $\mathbf{n}_j$ ("A") is projected along the normal ("C") to its position $\mathbf{c}'_j$ ("D").

At this point, the topology that puts the representatives into relation comes into play. As the local classifiers should be ordered such that they resemble a continuous decision surface, a smoothing step is performed that takes care of ordering the local representatives to obey the topology induced by matrix A. This step is very similar to the update process used in the training of SOMs. The neighborhood kernel weighted by the number of samples in each Voronoi partition is used to calculate smoothed versions along the topology of both the classification vectors $\mathbf{v}_j$ and representatives $\mathbf{c}_j$. The idea behind this is to make them more similar to their topological neighbors in order to achieve a smooth representation of the decision boundary and to avoid overfitting. Using a formula similar to the Batch SOM training defined in Equation 2.14, the updated local classifiers (lines 12, 13) are

$$\mathbf{c}_j = \frac{\sum_{k=1}^{M} h_{\sigma(t)}(k,j) \cdot \gamma_k \cdot \mathbf{c}'_k}{\sum_{k=1}^{M} h_{\sigma(t)}(k,j) \cdot \gamma_k} \tag{5.6}$$

$$\mathbf{v}_j = \frac{\sum_{k=1}^{M} h_{\sigma(t)}(k,j) \cdot \gamma_k \cdot \mathbf{v}'_k}{\sum_{k=1}^{M} h_{\sigma(t)}(k,j) \cdot \gamma_k} \tag{5.7}$$

The new representatives $\mathbf{c}_j$ and classification vectors $\mathbf{v}_j$ are derived from the preliminary versions $\mathbf{c}'_j$ and $\mathbf{v}'_j$ defined in Equation 5.4 and subjected to the kernel



(a) (b)

Figure 5.3: (a) Training of a linear classifier; thin lines indicate borders of Voronoi set, "+" and "-" denote samples, "A" center point $\mathbf{n}$, "B" separating hyperplane, "C" normal $\mathbf{v}'$, "D" projected center: $\mathbf{c}'$, (b) smoothing over neighborhood topology: the upper part shows the local classifiers before smoothing step, the lower part after the smoothing step where the representatives and classification vectors are made more similar according to the topology

smoothing according to Equation 2.8. Further, $\mathbf{c}_j$ and $\mathbf{v}_j$ are weighted according to topological distance and by the number of data points γ_j they represent. Thus, a local classifier that represents many data points will pull its neighbor that represents relatively few samples in its direction. Also, an effect known in vector quantization as magnification factors [115] occurs that concentrates representatives in dense areas. During the first few epochs, a high value of σ ensures that the local classifiers are aligned according to the topology. As σ declines, the local classifiers specialize and reach their final positions. The influence through topological proximity thus vanishes. Figure 5.3(b) explains how this smoothing happens over a possibly overtrained situation, and shows how the representatives and classification vectors are realigned to be more similar to their neighbors. After this step, the current epoch is finished.

Figure 5.4(a)–(h) shows an example of training 5 local classifiers with a one-dimensional topology, referred to as $\{5\}$, on a simple non-linearly separable data set. It consists of 200 samples that are distributed along a sine-wave with Gaussian noise, where the class "+" is offset by a small vertical margin. In the figures, data points are represented as "+" and "o". The classification vectors $\mathbf{v}_j$ are shown as the arrows pointing to the direction of the "+" class, and the hyperplanes as thick, solid lines. The positions of the representatives $\mathbf{c}_j$ are denoted as small squares where hyperplane and classification vector intersect. The topology of the local

classifiers is visualized by dashed lines connecting adjacent local classifiers (i.e. where $A_{ij} = 1$). In Figure 5.4(a), the randomly initialized Decision Manifold is shown. After the first training epoch, depicted in Figure 5.4(b), the local classifiers are arranged in a more orderly fashion as a result of smoothing according to Equation 5.6, yet do not classify well after this iteration. Figures 5.4(c)–(e) show the consecutive stages of training as σ, the parameter controlling the mutual influence between topological neighbors, is decreased. The purpose of the earlier epochs is to roughly align the representatives along the topology, while the later epochs are for fine-tuning the individual areas of each classifier. The finished Decision Manifold is shown in Figures 5.4(f),(g) with and without the data points. It can be seen in Figure 5.4(h), which shows the Bayes decision boundary as a thick line along with the decisions over the whole feature space, that the optimal decision boundary has been approximated very reliably. Figures 5.4(i),(j) show a trained Decision Manifold of topology $\{1\}$ that consists of only a single classifier, and is thus equivalent to performing a simple linear classification. This topology is not capable of approximating the decision boundary sufficiently. Selecting the (unknown) correct topology in the first place is thus very important and will be dealt with in Section 5.4.3.

The complexity of the training algorithm can be calculated as the sum of the sample assignment to representatives $O(N \cdot M)$, training of the M linear classifiers $O(M \cdot O_l(N))$, where O_l denotes the classifier complexity, and the complexity of the SOM update step $O(M^2)$. For T epochs, this results in $O(T \cdot (N \cdot M + M \cdot O_l(N) + M^2))$. The training algorithm can be implemented very efficiently. In the current experiments running on a 1.60 GHz computer and a Matlab implementation of the algorithm, the training duration for a data set with $N = 400$ and $D = 50$, and training parameters $T = 5, M = 20$ is less than a second.

5.4.2 Classification with Decision Manifolds

When the local classifiers are in their final positions, the training algorithm enters its last phase where the classification performance is fine-tuned on the training data set. In its most simple form, classification is performed by assigning a data sample to its closest linear classifier in the same way as during training, and then classifying it according to its position relative to the hyperplane, and is defined analogous to Equation 5.1 in non-homogeneous coordinates:

$$\hat{y}(\mathbf{x}) = \operatorname{sign}\big((\mathbf{x} - \mathbf{c}_{I(\mathbf{x})})^\top \cdot \mathbf{v}_{I(\mathbf{x})}\big) \tag{5.8}$$

A more sophisticated approach takes advantage of the topology: Since neighboring representatives are expected to form a smooth decision boundary, voting of the local classifier ensemble according to the topological proximity of every

classifier to the representative closest to $\mathbf{x}$, i.e. $\mathbf{c}_{I(\mathbf{x})}$, can significantly increase accuracy [122]:

$$\hat{y}(\mathbf{x}, \sigma) = \text{sign}\big(\sum_{j=1}^{M} K_\sigma(I(x), j) \cdot \hat{y}(\mathbf{x}) \big) \tag{5.9}$$

where σ is the smoothing width, with higher values increasing the influence of distant classifiers in terms of the topology. Effectively, the datum $\mathbf{x}$ is classified by all the hyperplanes, and the final decision is performed as weighted voting where neighboring classifiers receive a higher weight. In the last step of the algorithm (line 15), the training set accuracy (the percentage of correctly classified samples) of the classifier is maximized with respect to σ. This is done by sampling several values of $0 < \sigma \ll M$ and by setting

$$\sigma_{\text{final}}(C, V, X, \mathbf{y}) = \arg\max_{\sigma} accuracy(C, V, X, \mathbf{y}, \sigma) \tag{5.10}$$

Next, five data sets representing typical non-linearly separable supervised learning problems are discussed, each consisting of 200 samples. They are shown in Figure 5.5 together with the trained Decision Manifold. Note that the dashed lines again do not determine the classification boundaries, but indicate the topological neighborhood of the local classifiers.

The first example, shown in Figure 5.5(a), consists of samples divided by a boundary along a quadratic polynomial. The Bayes decision boundary is non-linear in this case. 6 local linear separators are trained, resulting in the ordered approximation visualized in the figure.

In Figure 5.5(b), both classes are normally distributed with the same center, but class "+" has a higher variance. In this case, there is considerable overlap between the two classes in all regions. The Bayes optimal boundary runs along a circle where the class conditional probability density functions intersect. As Figure 5.5(b) shows, the Decision Manifolds algorithm is capable of finding this border. Within this circle, there will likely be some misclassifications of class "+", but class "o" is still more common there.

In a related example shown in Figure 5.5(c), class "o" is distributed uniformly on a circle, and class "+" is normally distributed around its center. A Bayes optimal classifier would assign label "o" to any point on the circle and "+" otherwise. The Decision Manifold consists of 5 local classifiers along a one-dimensional bounded topology. This is a mismatch to the topology of the actual decision boundary, which is also one-dimensional but circular. However, most of the classification boundary can still be captured. Due to the sparsity of samples outside the circle, the decision boundary does not identify these samples correctly as "+".

Figure 5.5(d) shows the results for a data set where the Bayes decision boundary is not contiguous. The decision hypersurface is split into two linear manifolds

Table 5.1: Eigenvalues of PCA, Dimensionality estimation for the topology connecting the local classifiers

i	λ_i	Topology for $d_{\max} = i$	without 0,1	Graph
1	0.2618	$\{10\}$	$\{10\}$	
2	0.2164	$\{5 \times 4\}$	$\{5 \times 4\}$	
3	0.1287	$\{4 \times 3 \times 2\}$	$\{4 \times 3 \times 2\}$	
4	0.1094	$\{3 \times 3 \times 1 \times 1\}$	$\{3 \times 3\}$	
5	0.0953	$\{3 \times 2 \times 1 \times 1 \times 1\}$	$\{3 \times 2\}$	
6	0.0853	$\{2 \times 2 \times 1 \times 1 \times 1 \times 0\}$	not valid	
7	0.0525	$\{2 \times 2 \times 1 \times 1 \times 1 \times 0 \times 0\}$	not valid	
8	0.0506	$\{2 \times 2 \times 1 \times 1 \times 0 \times 0 \times 0 \times 0\}$	not valid	

(parallel lines). After training of 6 local classifiers with a one-dimensional topology, the result shows that the Decision Manifolds method is capable of dealing with this problem. It could have been solved with only 2 linear classifiers, but it has been chosen as a demonstration of how the representatives are aligned in case of a topology breach. Further, it is demonstrated in this example that the Decision Manifolds' performance does not deteriorate in case of over-specification in terms of the number of classifiers.

In Figure 5.5(e), separation of the XOR-problem is demonstrated with four local classifiers. Here, the feature space is split along the diagonals, and the neighborhood relations are disregarded for classification, i.e. σ_{final} is close to zero and no voting is performed.

5.4.3 Topology Estimation and Model Selection

It has been mentioned above that selection of a suitable topology is critical for the performance of the Decision Manifolds algorithm, as the topology constrains the shape of the decision boundary. In this section, a scheme for topology estimation is presented, and a model selection approach that trains several Decision Manifolds with different topologies and selects the best one. Training over a number of topologies can be afforded due to the computational inexpensiveness of training a Decision Manifold. The topology connecting the representatives can have at most dimension $D - 1$. If it is assumed that each axis of the topology grid contains 5 local classifiers, and the dimension of the data set is 50, this would result in 5^{49} classifiers, which clearly cannot be handled. As the dimension of the topology is likely to be much lower, a reasonable estimation of the intrinsic dimensionality

of the data set has to be performed. This task has been addressed previously [5]. In the context of Decision Manifolds, a simple PCA-guided scheme is applied. From the data set X, the ordered set of eigenvalues $\lambda_1, \ldots, \lambda_D$ is extracted, which is normalized to add up to one. Note that PCA is not used for dimensionality reduction of the data set, just for estimation of the topology. For the approximate desired number m of classifiers distributed along all discrete axes, the topology is constructed as follows:

$$\mathfrak{d}_i = \left\lfloor \frac{m \cdot \lambda_i}{\sum_{j=1}^{d_{\max}} \lambda_j} \right\rfloor \quad , \tag{5.11}$$

where $\mathfrak{d}_i$ is the number of classifiers along the rectangular topology's i^{th} axis, and $\lfloor \cdot \rfloor$ denotes rounding down to the closest integer. The topology is then $\{\mathfrak{d}_1 \times \cdots \times \mathfrak{d}_{d_{\max}}\}$. $d_{\max}$ is the dimension of the topology, at most the data set dimension D. $\mathfrak{d}_i = 1$ can be omitted, since an axis that only holds one discrete coordinate does not provide any information, and if any $\mathfrak{d}_i = 0$, the topology is not valid. Further, the topology is constructed as a ratio of the up to D eigenvalues $\lambda_1 : \lambda_2 : \cdots : \lambda_{d_{\max}}$. For example, the Pima Indian Diabetes data set, which will be discussed in the next section, consists of 8 variables. Its ordered eigenvalues are $0.26, 0.21, 0.12, 0.10, 0.09, 0.08, 0.05, 0.05$; for $m = 10$ the candidate topologies are shown in Table 5.1. This procedure is repeated by iterating $m = 1 \ldots 10$, thereby a set of 17 distinct topologies is obtained, the smallest of which is $\{1\}$, the largest is $\{4 \times 3 \times 2\}$. 10 has been chosen as a reasonable upper limit for m as real-world problems rarely require large numbers of separating hyperplanes. For estimation of the performance of the Decision Manifold with each topology, X is split into training and test set. After each classifier has been trained, σ_{final} is estimated on the training set. The resulting Decision Manifolds are then evaluated on the test set, and the best model is selected.

For estimation of the generalization accuracy, 10-fold cross-validation [87] is performed. The remaining 90 % of the data set are divided into training and test set by the ratio 80 to 20. The topologies for the models are then estimated by the PCA approach described in the previous paragraphs and the models are trained, and σ_{final} is estimated. The resulting classifiers are then evaluated on the test set, and the best model is selected for validation, resulting in the final accuracy measurement. This Training-Test-Validation approach is summarized in Table 5.2 for one fold.

To summarize the methodological part of this chapter, the properties of the Decision Manifolds method are recapitulated:

- The algorithm is a stochastic supervised learning method for two-class problems.

Table 5.2: Model Selection

Training			Test	Validation
Classifier	Topology	Estimate σ	Test Set Accuracy	Validation Set Accuracy
1		0.27	87 %	
2		1.19	88 % Best Model	85 %
3		2.14	77 %	
4		0.66	82 %	
. . .				

- Computation of a Decision Manifold is computationally very efficient.
- The topology induced by adjacency matrix A defines the ordering and alignment of the local classifiers; it is also exploited for optimizing classification accuracy by a weighted voting scheme.
- As the topology of the decision hypersurface is not known in advance, a heuristic model selection is applied that trains several classifiers with different topologies.
- The classifier performs well in case of multiple non-contiguous decision surfaces and non-linear classification problems such as XOR.

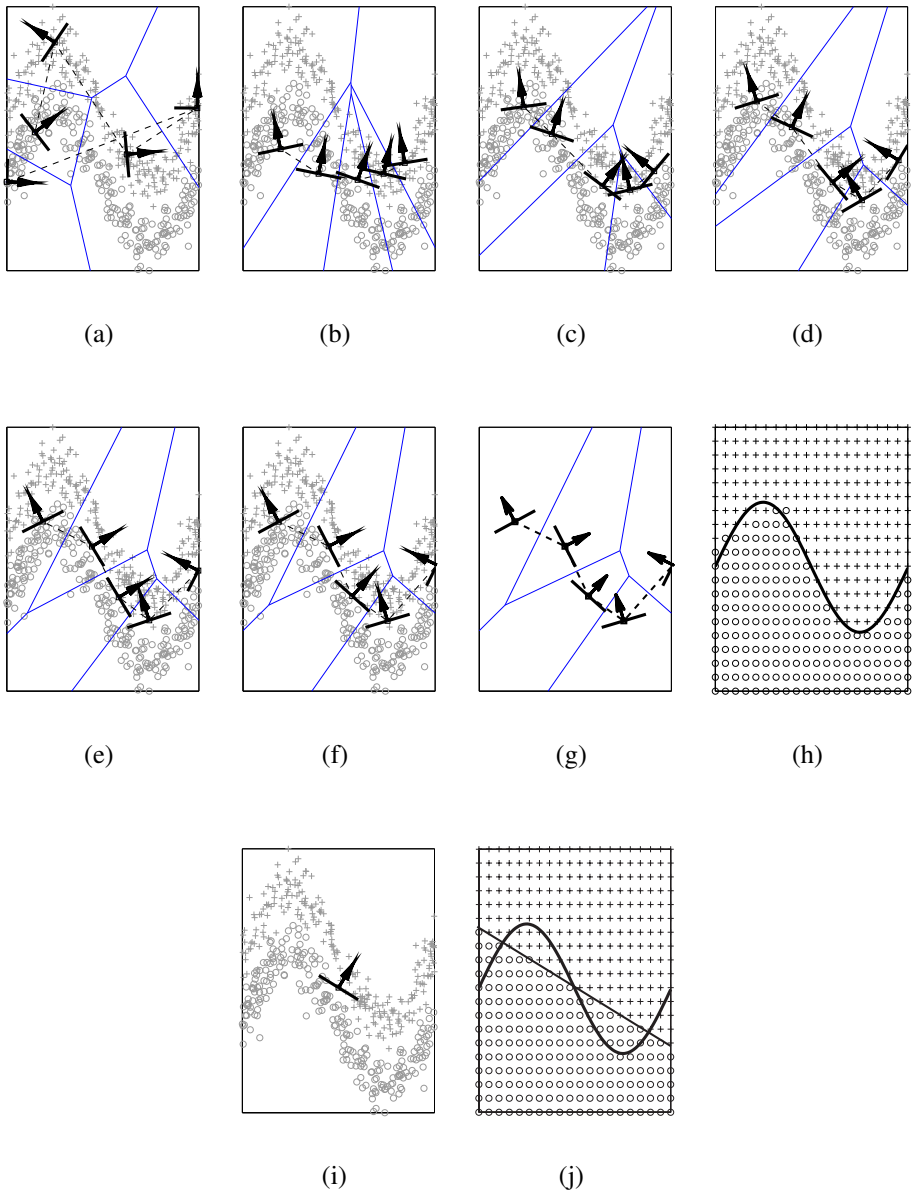


Figure 5.4: Training algorithm on non-linearly separable data set: (a) after initialization, (b)–(f) after 1^{st}–5^{th} epoch, (g) after training (without data samples), (h) classification results and Bayes decision boundary, (i) only one classifier, (j) classification with one classifier

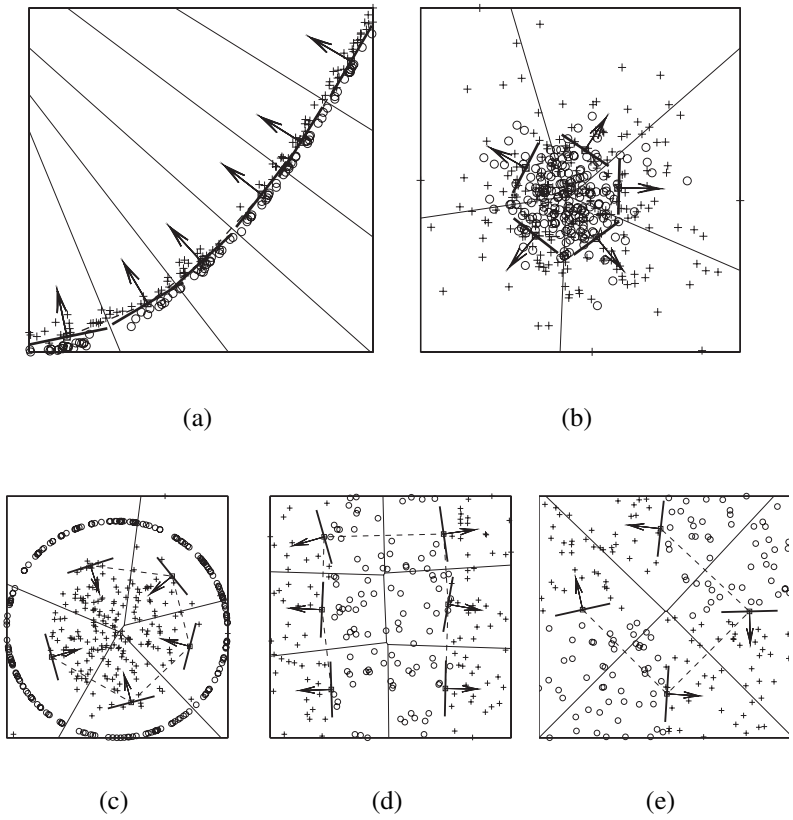


Figure 5.5: Decision Manifolds on artificial data sets: (a) quadratic decision surface, (b) 2 Gaussians with different variances, (c) ring and Gaussian, (d) 2 linear separations, (e) XOR

5.5 Experimental Results

The experiments that have been conducted are categorized into tests with artificial data sets, benchmark data sets, and different linear classifiers. The artificial data sets in Section 5.5.1 are designed to test a specific capability of the classifier. The strengths and weaknesses of the classifiers become obvious with these data sets. The benchmark data in Section 5.5.2 sets show the applicability of Decision Manifolds in real-world settings. The final experiment in Section 5.5.3 tests which linear classifier should be used.

The results of the Decision Manifolds are compared to Random Forests, linear, polynomial, and radial basis function SVMs, k-Nearest Neighbors, and Decision Trees, by averaging the out-of-bag error for 10-fold crossvalidation. The experiments of these classification algorithms have been performed with R, a computational statistics environment [1], using the libraries "nnet", "ipred", "RandomForests", "e1071", and "rpart". The parameters have been estimated according to the following model selection schemes by sampling the parameter in question in 15 steps: For polynomial SVMs, the degree of the kernel has been tuned in the range of 2–5, and for RBF kernels, the radius has been tuned in the range of 0.2 – 4. k-NN was performed with $1 \leq k \leq 29$, where k are only the odd numbers in order to avoid ties. For Random Forests, 500 trees have been trained and random splitting with 30% of the variables has been used. In this case, no model selection has been performed, as Random Forests are generally very robust with respect to the choice of parameters. For Decision Trees, different pruning thresholds have been tried, a parameter that influences the size of the tree grown. Decision Manifolds have been trained with $T = 5$ epochs, and with Moore-Penrose-Inverse as underlying linear classifier. All the experiments have been performed on a 1.60 GHz computer, and a Matlab implementation of the Decision Manifold algorithm has been used.

5.5.1 Artificial Data Sets

In this section, certain properties of Decision Manifolds are investigated and compared to other machine learning methods. The data sets are low-dimensional and are specifically designed to address a characteristic classification problem. Each data set is generated from a hidden probability density function. Three samples of different sizes are taken from each generating function in order to show the scalability and convergence of the classifiers. Tables 5.3 and 5.4 summarize the results.

The first class of problems are the chessboard family of data sets, depicted in

[1] R can be obtained at http://cran.r-project.org.

Figure 5.6. The XOR is the simplest of these, with 2 fields of either class. The other data sets are bigger, with 4×4 and 8×8 fields. The XOR problem can be easily solved by any classifier except for the linear SVM, and the other algorithms achieve error rates of 0 – 5%. Random Forests perform best for 500 and 1000 samples, with 0.6 and 0.0% error rates, respectively.

For the 4×4 chessboard, the error rates are significantly higher, even though a Bayes decision boundary with 0% error rate exists. As the number of XOR-like fields increases, the number of points may appear deceptively high: For this data set, there are $4 \times 4 = 16$ fields, so the number of points per field is only 12.5 on average for the 200 data points setting, making this data set much more difficult than the XOR. Decision Manifolds achieve errors 18.0, 10.2, and 9.0 %, for data set sizes of 200, 500, and 1000 points, respectively, and perform slightly worse than RBF SVM, k-NN, and Random Forest, while outperforming the other 3 classifiers. Decision Manifolds can solve this problem by aligning the local classifiers row by row between the fields of opposing classes. The setup of the local classifiers works with both one- and two-dimensional topologies. However, this category of problems is particularly challenging for Decision Manifolds, which perform best in case of continuous boundaries of arbitrary shape, or continuous boundaries with a low number of breaks. An example of how this problem can be solved with Decision Manifolds is presented in Figure 5.6(b), where a $\{5 \times 5\}$ topology is selected, resulting in an error as low as 2.5%. However, such a topology is rarely selected for such a low-dimensional data set, due to the fact that the whole concept of Decision Manifolds assumes that the data set's complexity is much lower than the original data space. Thus, the errors reported in Table 5.3 are significantly higher than the 2.5%.

For the 8×8 chessboard, the error rates are once again higher. There are $8 \times 8 = 64$ different fields, resulting in 3.125 points per field for the 200 samples data set, and 15.625 points per field for the 1000 samples data set. These are very low numbers and make prediction very difficult. Decision Manifolds generally perform bad on this data set. In order to correctly classify this example, a linear classifier would have to be placed between all adjacent fields. The number of such transitions is $n \cdot (n-1) \cdot 2$, thus for the classical XOR, 4 local classifiers are required, for the 4×4 chessboard, the number is 24, and for the 8×8 chessboard, 112 linear classifiers are required. The model selection scheme typically does not test such a high number of linear classifiers. Even if such a high number of local classifiers is provided, they are rarely placed in the correct positions. A Decision Manifold with a topology of $\{11 \times 11\}$ is shown in Figure 5.6(b), which is not trained under the model selection scheme. It achieves an error of 23.0 %, but there is no clearly perceivable structure of how the Decision Manifold folds onto the data set and the boundaries. A data set of this complexity obviously cannot be handled by the Decision Manifolds algorithm. Random Forests and k-

NN show the best performance as the number of samples is increased, with less than half of the error rates of all other classifiers at 1000 data samples. k-NN are at an advantage as the chessboard does not alter densities, but only increases the complexity of the boundaries.

The next class of problems are also non-overlapping, crisp data sets where the separating boundary has a special, non-linear shape. The data points are sampled from a uniform, quadratic area, and the label of the sample is derived from its position. Three data sets are generated in this category: The "Circle", "2 linear", and "sine" data sets. All of these data sets are two-dimensional and could be separated without any error if the Bayes decision boundary was known. The point of introducing these data sets here is to show the performance of non-linear separating boundaries.

The "Circle" data set has a circular boundary, with opposing labels inside and outside of the circle. When compared to the other algorithms, Decision Manifolds achieve good results, with 7.0, 4.4 and 2.9 % for the three data set sizes. Only polynomial and RBF kernel SVM perform better on all three sizes, and Random Forests and k-NN achieve similar results as Decision Manifolds. A trained Decision Manifold is shown in Figure 5.7(a) with 500 data points and 5 local linear classifiers. It can be seen that the 5 hyperplanes provide a good approximation of the circular boundary, but the approximation is not perfect: At the corners where two of the hyperplanes meet, there are several misclassifications.

The "2 linear" data set consists of two parallel linear boundaries, such that one class is in the middle, and the other one is split between the first class' left and right. This data set has been introduced previously, for example in Figure 5.7(b). All the classifiers except for the linear SVM perform very good on this data set. Decision Manifolds have error rates between 0.5 and 1.5 %, outperforming the SVMs and k-NN with all data set sizes. Random Forests and Decision Trees produce almost no errors with this data set, as the fact that the boundaries are parallel to the coordinate axes puts these information theoretic algorithms at an advantage. The data set is shown in Figure 5.7(b) with a trained Decision Manifold with a topology of $\{2\}$, which is sufficient to approximate the two boundaries.

The "sine" data set has a boundary that resembles a sine wave. An approximation is shown in Figure 5.7(c) with 4 linear classifiers. It can be seen that the boundary can be approximated to some degree, but there are still some misclassifications close to the high and low points of the sine wave. The Decision Manifold achieves error rates between 6.0 and 4.2 %, which is worse than Random Forests, RBF SVM, and k-N. Still, Decision Manifolds are better than Decision Trees, linear SVMs, and polynomial SVMs.

The next 4 artificial data sets are also two-dimensional but have overlapping class distributions, thus a Bayes optimal classifier will fail to correctly classify all the samples. The data sets in this category are the "polynomial", "2 Gauss I",

"2 Gauss II", and the "ring, Gauss" data sets. Examples for these data sets with trained Decision Manifolds are given in Figure 5.8.

The first in the overlapping category is the "polynomial" data set where the boundary resembles a quadratic function. The samples of the two classes are arranged above and below the decision boundaries, with a small amount of Gaussian noise that allows for some degree of overlap across the boundary. Here, Decision Manifolds outperform all the other algorithms, achieving error rates between 2.5 and 7.5 for 1000 and 200 points, respectively. An example with a $\{6\}$ topology is shown in Figure 5.8(a).

The "2 Gauss I" data set consists of two Gaussian bell-shaped curves with the same center and different covariance matrices, as shown in Figure 5.8(b). Due to the differences in the covariance matrix, one Gaussian has higher density around the center, but lower density further away from the center. The Bayes optimal decision boundary lies along a circle around the center. Decision Manifolds are outperformed by SVMs by a small margin at all three data set sizes. E.g. at 1000 points, Decision Manifolds achieve an error of 24.8% and RBF SVMs 23.3%.

The "2 Gauss II" data set consists of two Gaussians with different centers but equal covariance matrices. This data set is shown in Figure 5.8(c). The Bayes optimal decision boundary is simply a line that is orthogonal to the line connecting the cluster centers. The boundary is not parallel to a coordinate axis, making it harder for the information theory based classifiers (Random Forests, Decision Trees) to find them when the number of points is low. Decision Manifolds perform best on this data set for point numbers greater than 200.

The "Ring and Gaussian" data set has also been explained previously in Section 5.4.1. It consists of a Gaussian and samples of the other category distributed along a circle with the same center as the Gaussian. An example of a Decision Manifold with 5 local classifiers is shown in Figure 5.5(c). When compared with the other classifiers, Decision Manifolds perform better than the other ones on all three data set sizes.

In this category of low-dimensional overlapping problems, Decision Manifolds frequently outperform the other classifier algorithms. Decision Manifolds are good at finding complex, non-linear boundaries as long as they are continuous.

The last category of data sets cover typical machine learning problems related to intertwined, non-overlapping shapes. These are the the intertwined rings data set and the doublehelix data set, both of which are three-dimensional. In order to visualize the Decision Manifolds on data sets which have a dimension of more than two, a PCA projection is shown along with a SOM projection.

The "2 rings" data set consists of two intertwined rings. A PCA projection of a data set sampled from this distribution is shown in Figure 5.9(a), where the data samples and the classifier positions and their classification vectors are projected

onto the two-dimensional plane spanned by the most important eigenvectors. In Figure 5.9(c)–(d), the hit histograms of samples from both classes on a SOM trained on this data set is shown. The classifier positions are also mapped onto this map and are depicted as large triangles. In this figure, a $\{2 \times 2\}$ Decision Manifold is shown, which achieves an error of 2.0 %. The PCA projection shows a problem inherent in such a linear projection, as it is not perceivable where the arrows are actually pointing and where the local classifiers are actually placed. The SOM visualization is better at showing the positions of the local classifiers, but it is not possible to show the classification vectors. However, if the triangles are placed between samples from opposing classes, this is a hint that the classifier is placed correctly.

The second data set in this category is the "Doublehelix" data set. It consists of two curves along which the data points are generated. The curves resemble intertwined spirals that spin twice around their middle axis. A $\{5\}$ Decision Manifold on this data set is shown in Figure 5.10. While the PCA plot shows what the data set looks like, it does not show the shape of the Decision Manifold in an understandable way. The SOM visualization shows the distribution of the data samples across the map. It can be seen that the triangles are positioned close to the transition areas between the two classes.

Decision Manifolds achieve error rates between 0.1 and 4.0 % for both data sets and all three sizes. However, the density based classifiers RBF SVM and k-NN result in no misclassifications at all. This is due to the fact that there is no overlap between the classes, and that the shapes of the samples form either class are continuous. Linear and polynomial kernel SVMs are both worse than Decision Manifolds, and Random Forests and Decision Trees are better on the "2 rings" and similar on the "Doublehelix" data sets.

To summarize this section, Decision Manifolds have been shown to perform well in terms of execution time and classification performance. The chessboard family of data sets are examples where the Decision Manifold has problems finding the decision boundaries. Compared to density-based classifiers such as k-NN, Decision Manifolds perform worse in cases like the chessboard since it has to approximate each transition between different classes, while the complexity does not increase for density based classifiers. In the non-overlapping category of data sets, Decision Manifolds achieve good results, but are outperformed by Random Forests and RBF SVMs. Decision Manifolds perform much better in the overlapping class of problems, as long as there are not too many breaks in the shape of the decision boundary, even if the decision boundary assumes complex non-linear shapes. Decision Manifolds dominate most of the examples from this category, as they are primarily designed to approximate non-linear shapes of continuous decision boundaries. Finally, in the intertwined category of problems, Decision Manifolds achieve low error rates but are beaten by density-based classifiers.

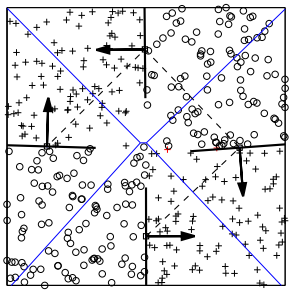
Table 5.3: Artificial data sets, part 1: Comparison of average 10-fold cross-validation error (in %) and training time

Data	Samples	Dec. Mf.	lin. SVM	pol. SVM	RBF SVM	R. F.	Dec. Tr.	k-NN
XOR	200	2.5 (3.0s)	41.0 (0.3s)	6.0 (1.4s)	**2.0** (3.6s)	2.5 (3.1s)	8.0 (0.2s)	2.5 (0.8s)
XOR	500	2.8 (10.7s)	44.0 (0.8s)	7.6 (3.5s)	2.6 (8.8s)	**0.6** (6.1s)	1.6 (0.3s)	4.8 (1.1s)
XOR	1000	0.9 (15s)	46.7 (2.0s)	3.0 (10.5s)	1.6 (21.7s)	**0.0** (13.8s)	1.4 (0.3s)	2.0 (2.1s)
Chess (4x4)	200	18.0 (4.7s)	46.5 (0.3s)	39.5 (1.6s)	**16.0** (4.8s)	20.0 (3.3s)	36.0 (0.2s)	17.5 (0.7s)
Chess (4x4)	500	10.2 (8.4s)	51.0 (0.7s)	40.8 (4.1s)	9.0 (23.9s)	**7.4** (9.9s)	11.2 (0.3s)	8.4 (1.2s)
Chess (4x4)	1000	9.0 (15.6s)	50.2 (1.7s)	44.9 (11.8s)	6.4 (55.3s)	**3.5** (14.7s)	7.5 (0.4s)	7.3 (2.1s)
Chess (8x8)	200	**44.5** (5.0s)	56.0 (0.3s)	49.5 (1.5s)	50.0 (6.1s)	49.0 (4.2s)	50.0 (0.3s)	50.5 (0.8s)
Chess (8x8)	500	42.8 (8.0s)	46.6 (0.9s)	45.0 (4.2s)	39.2 (29.9s)	28.2 (10.3s)	43.0 (0.3s)	**21.0** (1.2s)
Chess (8x8)	1000	38.2 (15.4s)	45.0 (2.4s)	46.5 (12.5s)	34.4 (92.5s)	17.4 (20.9s)	30.8 (0.5s)	**16.8** (2.3s)
Circle	200	7.0 (3.9s)	50.5 (0.3s)	7.0 (1.2s)	**5.5** (3.9s)	7.0 (2.8s)	9.5 (0.2s)	8.5 (0.6s)
Circle	500	4.4 (8.6s)	46.8 (0.7s)	2.4 (2.7s)	**0.8** (8.7s)	3.4 (6.1s)	7.2 (0.3s)	2.8 (1.0s)
Circle	1000	2.9 (15.6s)	46.1 (1.6s)	**1.3** (6.3s)	1.5 (21.4s)	2.9 (12.8s)	5.0 (0.3s)	1.7 (2.1s)
2 linear	200	0.8 (3.1s)	53.0 (0.3s)	4.0 (1.1s)	3.0 (3.5s)	0.5 (2.5s)	**0.0** (0.2s)	3.0 (0.6s)
2 linear	500	1.4 (8.6s)	51.0 (0.7s)	1.4 (2.9s)	1.4 (7.9s)	**0.4** (5.4s)	**0.4** (0.2s)	1.8 (1.0s)
2 linear	1000	0.5 (15.8s)	47.5 (2.0s)	1.0 (9.0s)	1.3 (21.4s)	0.5 (11.0s)	**0.4** (0.3s)	2.3 (2.0s)
Sine	200	6.0 (2.5s)	19.5 (0.3s)	22.5 (1.2s)	**1.5** (3.5s)	4.5 (2.8s)	9.5 (0.2s)	**1.5** (0.7s)
Sine	500	4.2 (8.0s)	16.4 (0.5s)	24.6 (3.0s)	2.0 (7.8s)	3.8 (6.1s)	8.0 (0.3s)	**1.8** (1.1s)
Sine	1000	4.2 (15.2s)	19.4 (1.4s)	24.5 (8.3s)	**2.3** (20.4s)	2.5 (13.1s)	5.3 (0.4s)	2.4 (2.1s)
Polynomial	200	**7.5** (3.0s)	46.5 (0.3s)	48.0 (1.2s)	15.5 (5.2s)	21.0 (3.4s)	29.5 (0.2s)	15.0 (0.7s)
Polynomial	500	**4.6** (8.9s)	42.4 (0.7s)	38.2 (3.6s)	5.2 (17.9s)	8.2 (8.6s)	15.8 (0.3s)	5.0 (1.1s)
Polynomial	1000	**2.5** (15.0s)	38.1 (1.9s)	42.0 (10.8s)	3.9 (55.2s)	4.2 (16.2s)	14.3 (0.4s)	**2.5** (2.1s)

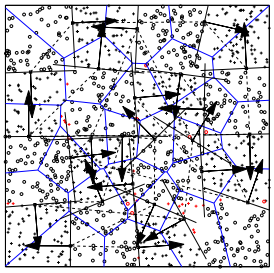
Table 5.4: Artificial data sets, part 2: Comparison of average 10-fold cross-validation error (in %) and training time

Data	Samples	Dec. Mf.	lin. SVM	pol. SVM	RBF SVM	R. F.	Dec. Tr.	k-NN
2 Gauss I[a]	200	28.5 (4.8s)	51.5 (0.3s)	**25.0** (1.9s)	25.5 (4.6s)	29.5 (3.3s)	33.5 (0.2s)	27.5 (0.7s)
2 Gauss I	500	26.8 (8.8s)	41.0 (0.7s)	26.8 (8.6s)	**24.2** (17.1s)	31.0 (8.8s)	26.4 (0.3s)	29.0 (1.0s)
2 Gauss I	1000	24.8 (15.0s)	44.4 (2.1s)	23.7 (28.0s)	**23.3** (59.1s)	27.8 (16.7s)	27.7 (0.4s)	27.4 (2.0s)
2 Gauss II[b]	200	6.0 (2.5s)	5.5 (0.2s)	7.0 (1.1s)	**4.5** (3.6s)	6.5 (2.6s)	6.0 (0.2s)	5.5 (0.8s)
2 Gauss II	500	**5.9** (8.4s)	7.0 (0.4s)	8.6 (2.6s)	6.8 (9.3s)	6.8 (6.3s)	8.4 (0.3s)	7.4 (1.1s)
2 Gauss II	1000	**5.8** (15.6s)	7.1 (0.8s)	8.4 (10.0s)	7.0 (28.8s)	8.5 (13.0s)	8.2 (0.4s)	7.5 (2.1s)
Ring, Gauss	200	**7.0** (4.8s)	45.5 (0.3s)	43.5 (1.4s)	16.5 (4.9s)	11.0 (3.3s)	13.5 (0.3s)	18.5 (0.7s)
Ring, Gauss	500	**6.4** (8.4s)	57.0 (0.7s)	27.0 (3.9s)	14.4 (18.3s)	10.8 (7.7s)	17.6 (0.3s)	14.4 (1.1s)
Ring, Gauss	1000	**5.2** (15.5s)	52.0 (2.0s)	31.5 (13.2s)	9.3 (59.3s)	5.3 (15.0s)	10.8 (0.4s)	8.0 (2.1s)
Int.tw. rings	200	3.0 (3.5s)	35.0 (0.3s)	10.5 (1.2s)	**0.0** (5.0s)	**0.0** (2.3s)	0.5 (0.2s)	**0.0** (0.8s)
Int.tw. rings	500	0.8 (8.1s)	34.0 (0.6s)	5.6 (2.3s)	**0.0** (13.0s)	**0.0** (5.7s)	0.4 (0.3s)	**0.0** (1.1s)
Int.tw. rings	1000	0.1 (15.5s)	34.5 (1.6s)	3.7 (5.2s)	**0.0** (27.9s)	**0.0** (10.4s)	**0.0** (0.3s)	**0.0** (2.4s)
Doublehelix	200	4.0 (4.5s)	45.5 (0.3s)	21.0 (1.3s)	**0.0** (6.0s)	6.0 (3.6s)	23.5 (0.2s)	**0.0** (0.8s)
Doublehelix	500	1.8 (8.3s)	57.0 (0.8s)	14.8 (3.1s)	**0.0** (19.8s)	1.6 (8.5s)	5.8 (0.3s)	**0.0** (1.2s)
Doublehelix	1000	1.1 (15.6s)	44.8 (2.6s)	14.7 (8.6s)	**0.0** (52.5s)	0.1 (15.8s)	2.5 (0.4s)	**0.0** (2.3s)

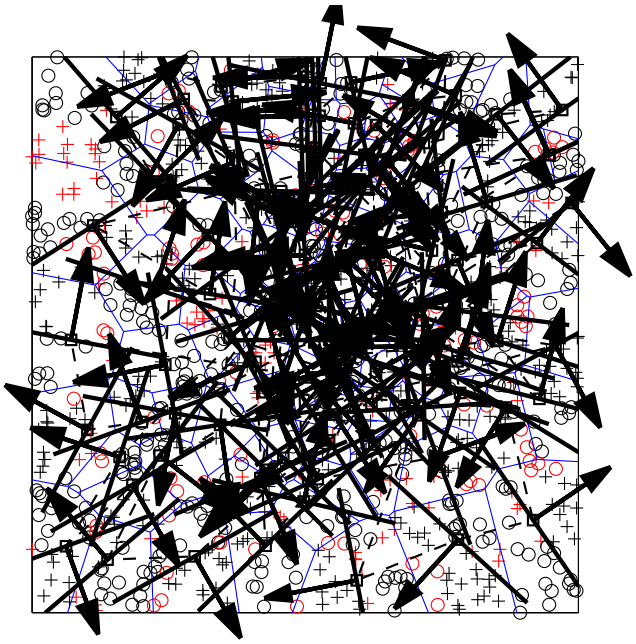
[a]The Gaussians of this data set share the same mean (center point), but have different covariance matrices
[b]The Gaussians of this data set have different means (center points), and have the same covariance matrix



(a)

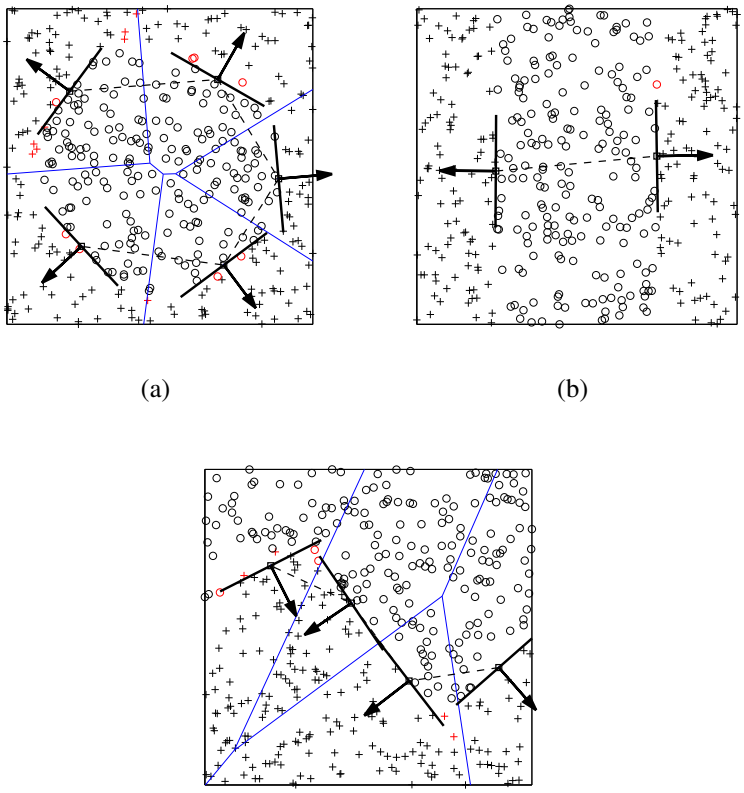


(b)



(c)

Figure 5.6: Artificial data sets from the chessboard family, and trained Decision Manifolds: (a) XOR = 2×2 chessboard, $\{2 \times 2\}$ Decision Manifold; (b) 4×4 chessboard, $\{5 \times 5\}$ Decision Manifold; (c) 8×8 chessboard, $\{11 \times 11\}$ Decision Manifold



(a) (b)

(c)

Figure 5.7: Artificial, non-overlapping, two-dimensional data sets with non-linear decision boundaries, and trained Decision Manifolds: (a) “Circle”, (b) “2 Linear”, (c) “Sine”

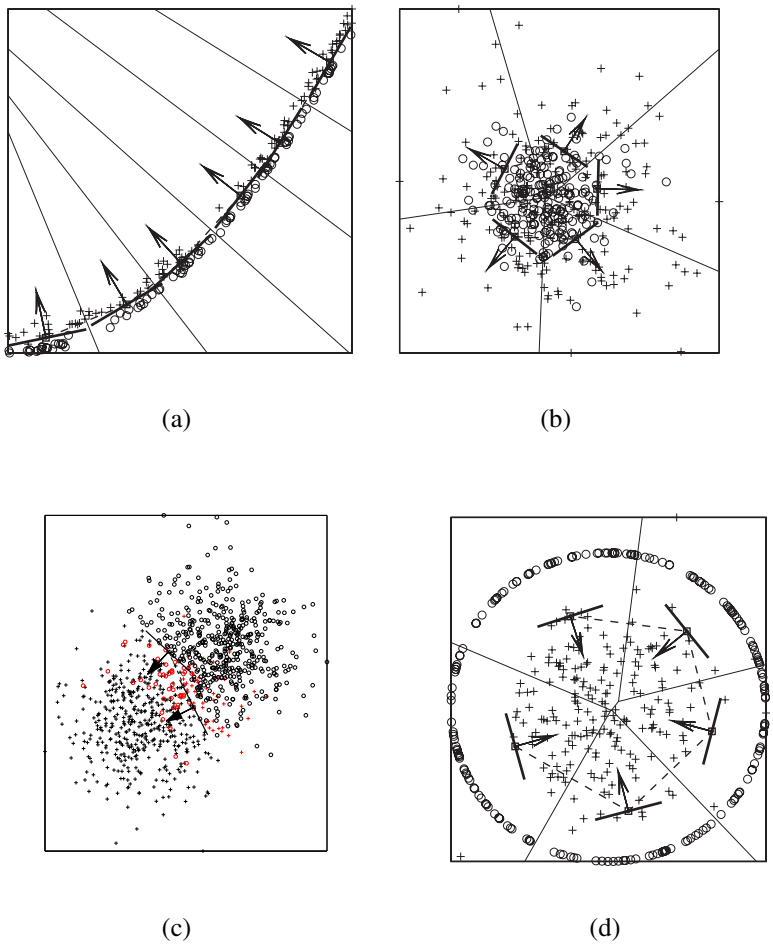
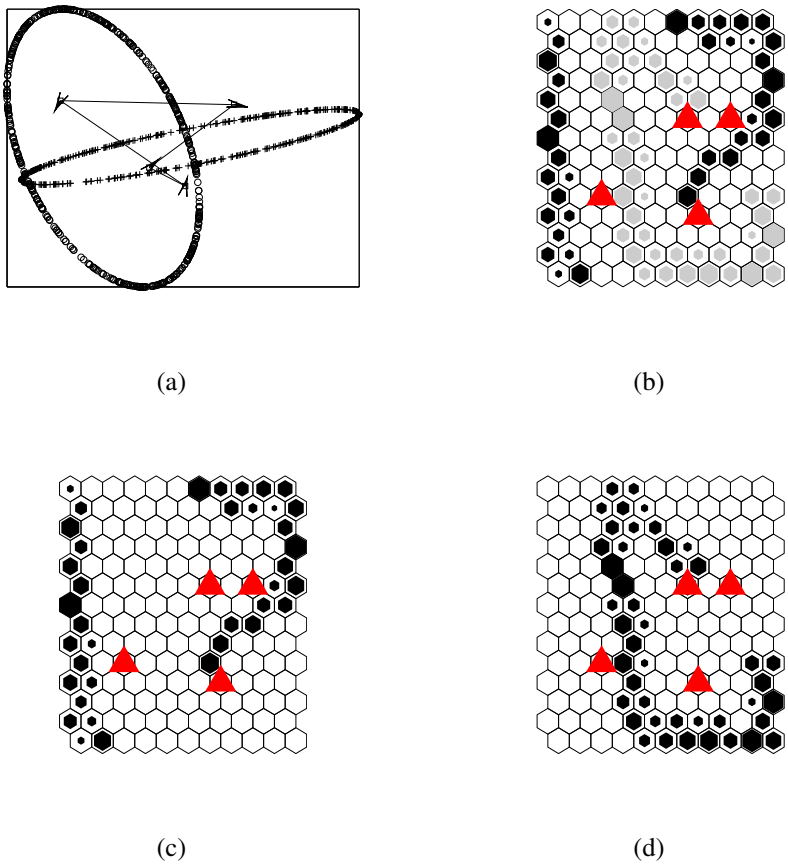


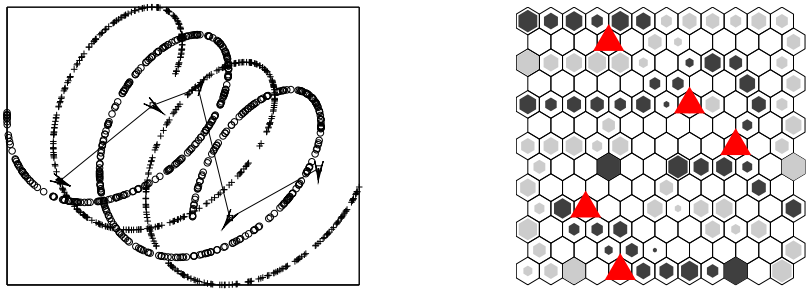
Figure 5.8: Decision Manifolds on overlapping artificial data sets: (a) quadratic decision surface ("polynomial"), (b) 2 Gaussians with different variances ("2 Gauss I"), (c) 2 Gaussians with different centers ("2 Gauss II"), (d) ring and Gaussian ("Ring, Gauss")



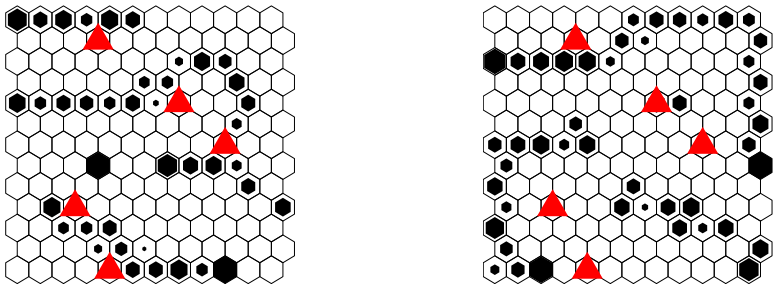
(a) (b)

(c) (d)

Figure 5.9: Projections of Decision Manifolds on "2 rings" (intertwined rings) data set: (a) PCA projection, (b)–(d) hit histogram of 2 classes onto $\{12 \times 13\}$ SOM: (b) both classes, (c),(d) only one class

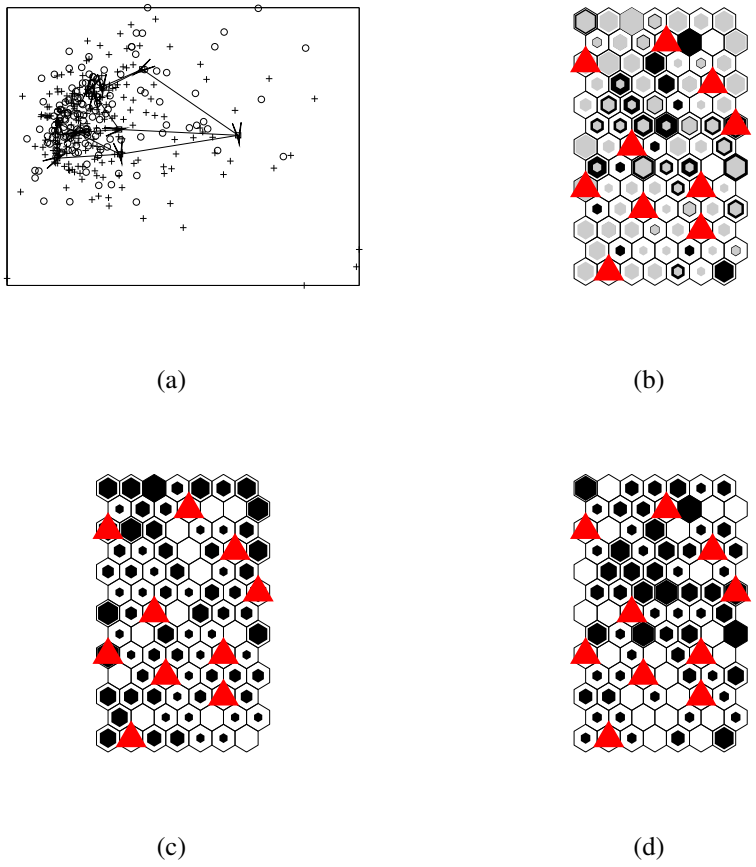


(a) (b)



(c) (d)

Figure 5.10: Projections of Decision Manifolds on "Doublehelix" data set: (a) PCA projection, (b)–(d) hit histogram of 2 classes onto $\{12 \times 13\}$ SOM: (b) both classes, (c),(d) only one class



(a) (b)

(c) (d)

Figure 5.11: Projections of Decision Manifolds on Bupa liver disorders data set: (a) PCA projection, (b)–(d) hit histogram of 2 classes onto $\{7 \times 13\}$ SOM: (b) both classes, (c),(d) only one class

5.5.2 Benchmark Data Sets

The benchmark experiments are performed on 7 binary and 3 multi-class supervised learning benchmark data sets taken from the UCI Machine Learning Repository, which are described in Appendix D: Bupa Liver Disorders, Pima Indian Diabetes, Spam, Ionosphere, Statlog Heart Disease, Sonar, and Statlog German Credit binary data sets, and Iris, Contraceptives, and Glass data sets. In case of categorical features, 1-to-N encoding has been applied, otherwise the data has been normalized with a zero-mean-unit-variance transformation.

The number of samples and dimensions are summarized in the first column of Table 5.5, which also summarizes the classification results. In column "Topol.",

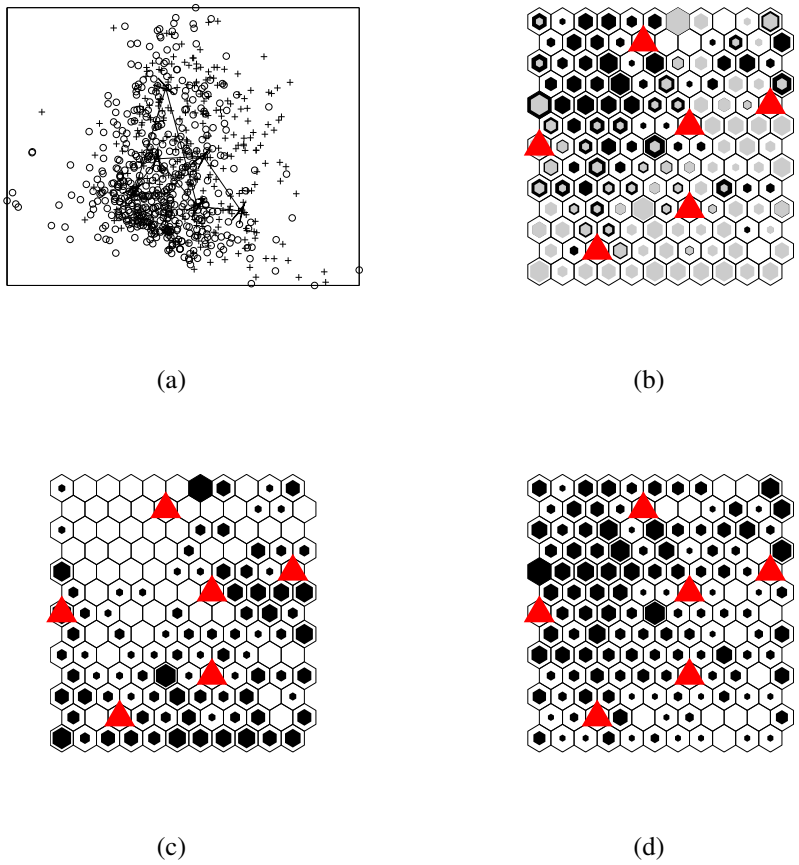


Figure 5.12: Projections of Decision Manifolds on Pima Indian Diabetes data set: (a) PCA projection, (b)–(d) hit histogram of 2 classes onto $\{11 \times 13\}$ SOM: (b) both classes, (c),(d) only one class

an example of the most frequently selected topology over different folds is shown. PCA and SOM projections of the data set and trained Decison Manifold are shown for the data sets. As the PCA and SOM projections are unsupervised, they do not distinguish between labeled samples. Thus, the samples of different classes are shown in different places only if the classes also happen to form a cluster that is preserved after the projection, which is increasingly unlikely for higher dimensions. Furthermore, PCA tends to explain less of the variance at higher dimensions. While the visualizatons provide some insight into the shape of the data set, they do not necessarily have to explain why a particular data set is easily separable or not. If the classes are projected to clearly distinguishable areas, then a separation with a high accuracy should be possible for a classifier, but if the

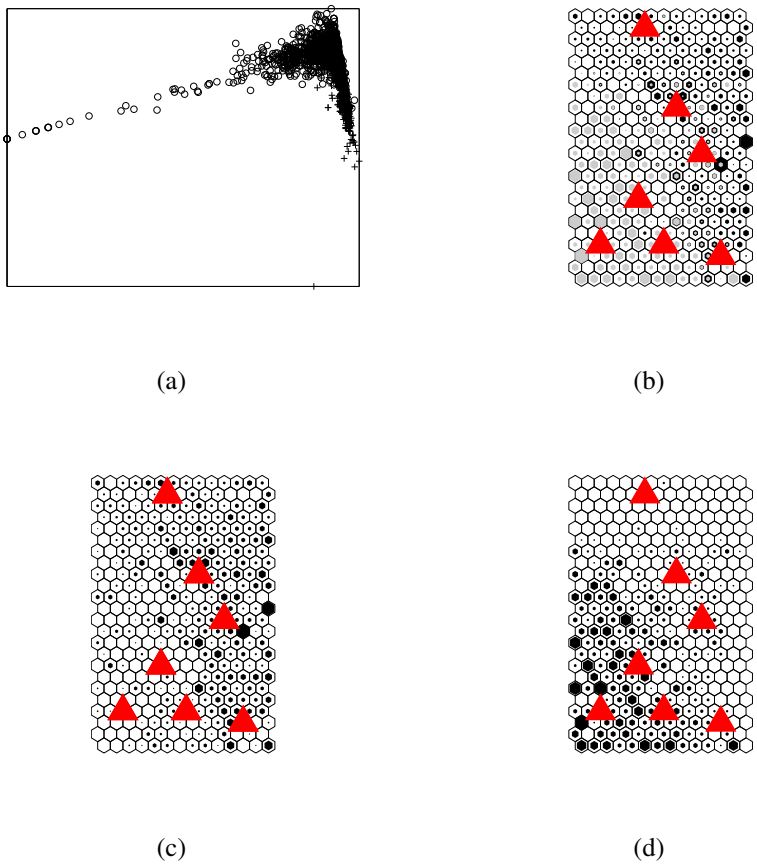


Figure 5.13: Projections of Decision Manifolds on Spam data set: (a) PCA projection, (b)–(d) hit histogram of 2 classes onto $\{11 \times 13\}$ SOM: (b) both classes, (c),(d) only one class

classes are highly overlapping after the projection, this does not mean the data set is not separable. Rather, it is an indication, but not a proof, for the difficulty of classifying such a data set.

A visualization of the Bupa Liver Disorder data set is shown in Figure 5.11. This data set is 6-dimensional, and has 345 samples with highly overlapping classes. The Decision Manifold has a topology of $\{5 \times 2\}$, which is the most frequently selected topology during the crossvalidation, as indicated in Table 5.5. In the SOM projection, one class tends to be clustered slightly above the center, while the other one is clustered on the bottom, both with a high degree of noise. The local classifiers are scattered across the whole map. Decision Manifolds achieve classification errors of 29.4%, and are beaten by Random Forests

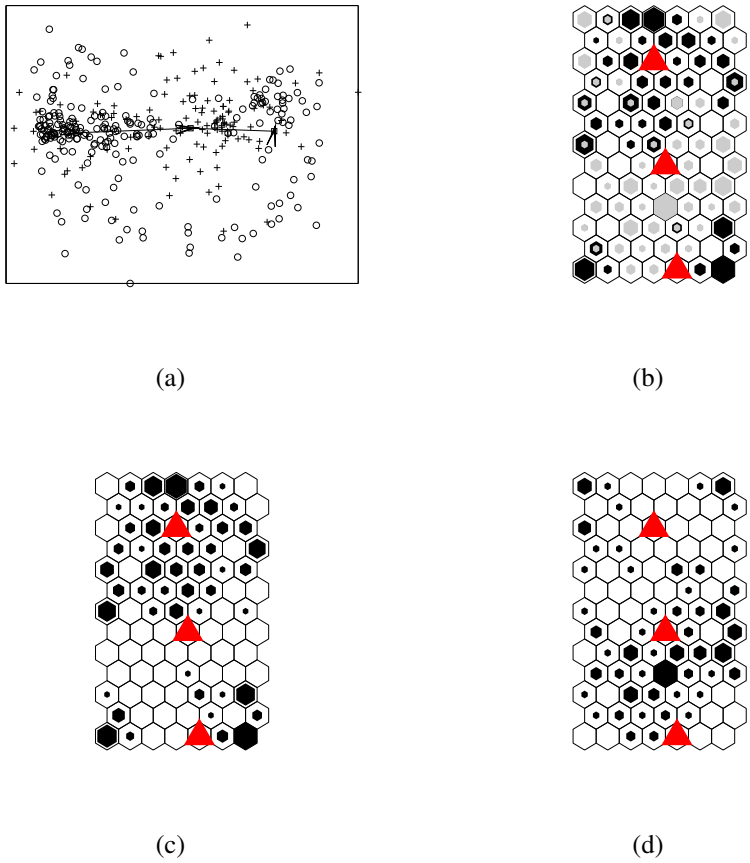


Figure 5.14: Projections of Decision Manifolds on Ionosphere data set: (a) PCA projection, (b)–(d) hit histogram of 2 classes onto $\{7 \times 13\}$ SOM: (b) both classes, (c),(d) only one class

and Decision Trees with 27.2 and 29.3 %, respectively.

The Pima Indian Diabetes data set is visualized in Figure 5.12. It is 8-dimensional and consists of 768 samples. From the PCA and SOM plots a slight separation between the two classes is visible. The center part of the SOM is occupied by samples from both classes, indicating that there is a high degree of overlap in the data set. Decision Manifolds outperform the other classifiers by a margin of 2.4 percentage points.

The Spam data set is 57-dimensional and has 4601 samples, which is visualized in Figure 5.13. Despite the high number of dimensions, the classes can be distinguished even after the projection, especially in case of the SOM projection. The PCA shows a shape that is typical for text data sets which are calculated by

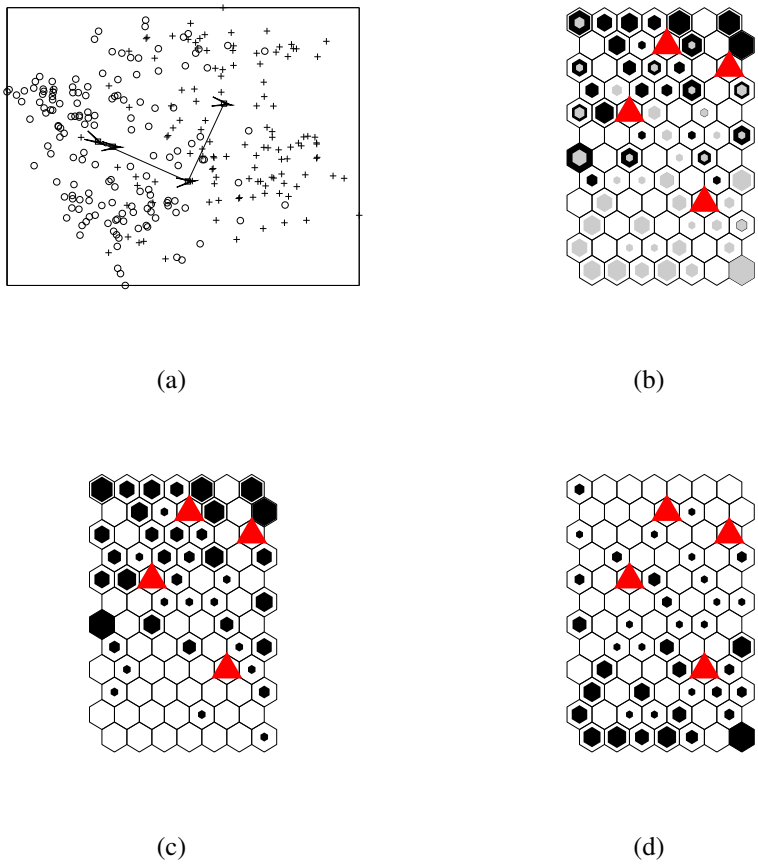
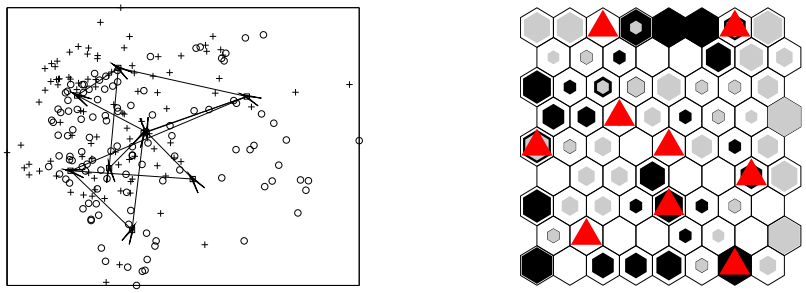


Figure 5.15: Projections of Decision Manifolds on Statlog heart disease data set: (a) PCA projection, (b)–(d) hit histogram of 2 classes onto $\{7 \times 12\}$ SOM: (b) both classes, (c),(d) only one class

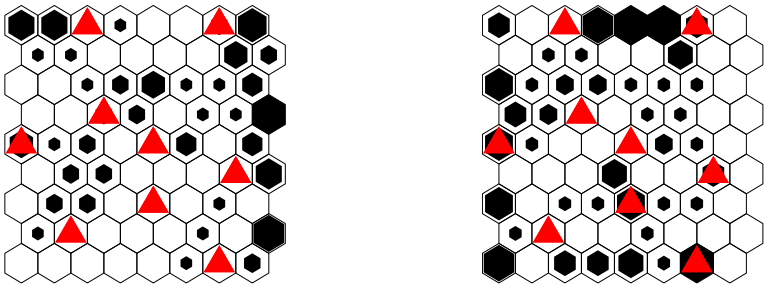
bag-of-words, and shows a cluster with two tails. This data set can also be classified by most algorithms with high accurracy. Random Forests perform best with 4.7%, and Decision Manifolds achieve an error of 7.7 %.

The Ionosphere data set is 34-dimensional with 351 samples. It is depicted in Figure 5.14. It can be seen that the classes are fairly well separated. Decision Manifolds are beaten by both RBF SVM and Random Forests, which achieve almost half the error rates of Decision Manifolds, which misclassifies 12.8 % of the data.

The Statlog Heart Disease data set is 13-dimensional and has 270 samples. Figure 5.15 shows this data set with a $\{4\}$ Decision Manifold. Like the previous example, the classes are well separated, which can be seen from both the PCA and



(a) (b)



(c) (d)

Figure 5.16: Projections of Decision Manifolds on Sonar data set: (a) PCA projection, (b)–(d) hit histogram of 2 classes onto $\{8 \times 9\}$ SOM: (b) both classes, (c),(d) only one class

SOM visualizations. Linear SVMs perform best with this data set at an error rate of 17.0 %, which indicates that there is a linear separation. The other algorithms, including Decision Manifolds, are only slightly worse.

The Sonar data set has only 208 samples but has a high dimension of 60, making it the sparsest data set that is investigated here. It is visualized in Figure 5.16. The PCA and SOM plots show that there is a large amount of overlap between the classes. The most frequently selected topology of $\{3 \times 3\}$ is relatively big compared to the others and is another indicator that the data set is complicated to separate. Decision Manifolds beat the other classifier algorithms with an error of 14.3 % by a small margin.

The German Credit data set, shown in Figure 5.17, is 20-dimensional with

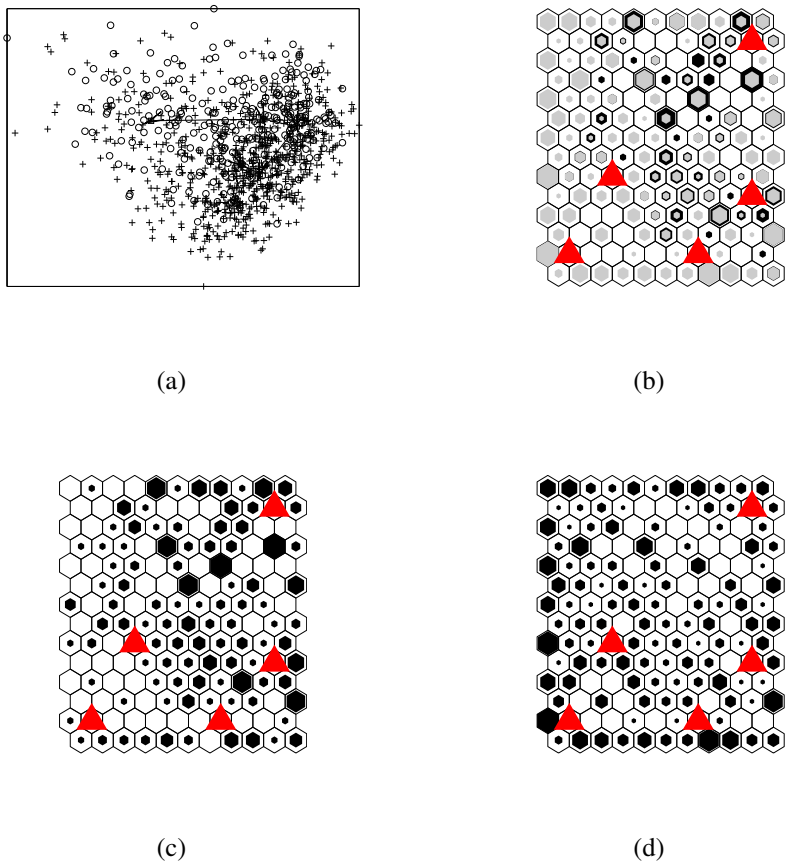


Figure 5.17: Projections of Decision Manifolds on German Credit data set: (a) PCA projection, (b)–(d) hit histogram of 2 classes onto $\{11 \times 14\}$ SOM: (b) both classes, (c),(d) only one class

1000 samples. From the PCA plot, the classes do not seem clustered. The SOM visualization shows a slight separation, but still significant overlap. The algorithms have errors between 23 and 28 %, with Decision Manifolds in the middle.

The non-binary data sets have been encoded as one-against-the-rest, thus creating several binary data sets. The data sets in this category are the Iris, Glass and Contraceptives data sets. The data sets created out of these are denoted by their respective numbers, i.e. “Iris Setosa” refers to the data set where the Setosa class is classified against Versicolor and Virginica, which are combined to form the second class.

The Iris data set consists of 3 classes, each with 50 samples. The version where Versicolor is classified against Setosa and Virginica is depicted in Figure 5.18.

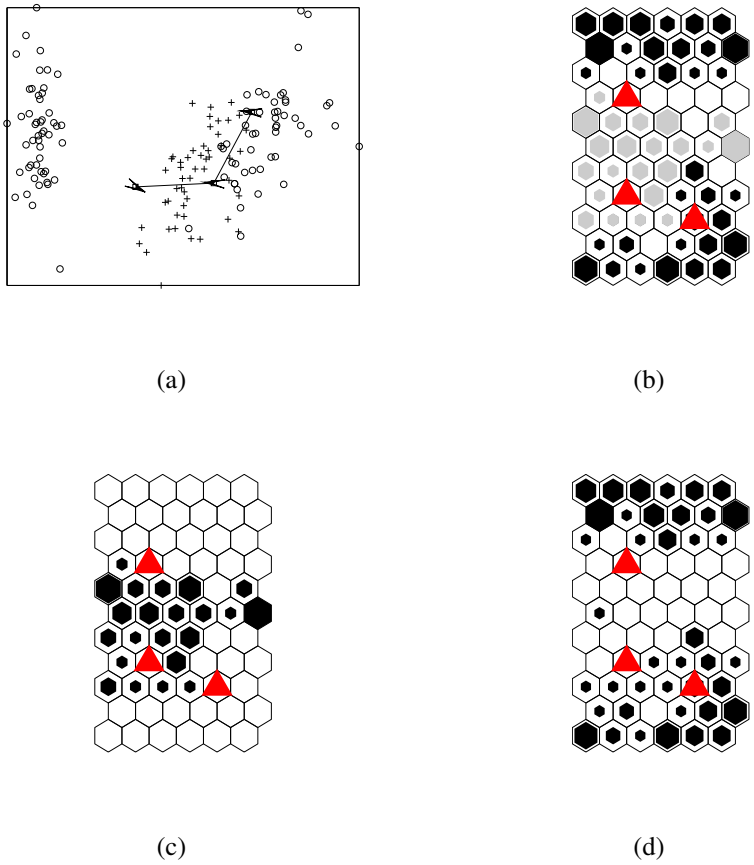


Figure 5.18: Projections of Decision Manifolds on Iris data set (Versicolor vs. rest): (a) PCA projection, (b)–(d) hit histogram of 2 classes onto $\{6 \times 11\}$ SOM: (b) both classes, (c),(d) only one class

The Versicolor class is between the Virginica and Setosa classes. This problem is solved with a Decision Manifold with 3 classifiers, where one local classifier separates the Setosa class, which is located in the upper third of the map, from Versicolor, and two classifiers separate Versicolor from Virginica. For this data set, Decision Manifolds achieve very good results. While the Setosa class can be separated at perfect accuracy by all classifier algorithms, Virginica and Versicolor are harder to separate, with the classifiers achieving error rates between 4 and 8 %, except for linear SVM, where error rates are higher. Decision Manifolds are best with the Virginica data set, ex aequo with RBF SVMs.

The Glass data set consists of 214 samples in 10 dimensions. There are 7 different classes in this data set, which leads to 7 different data sets. An example

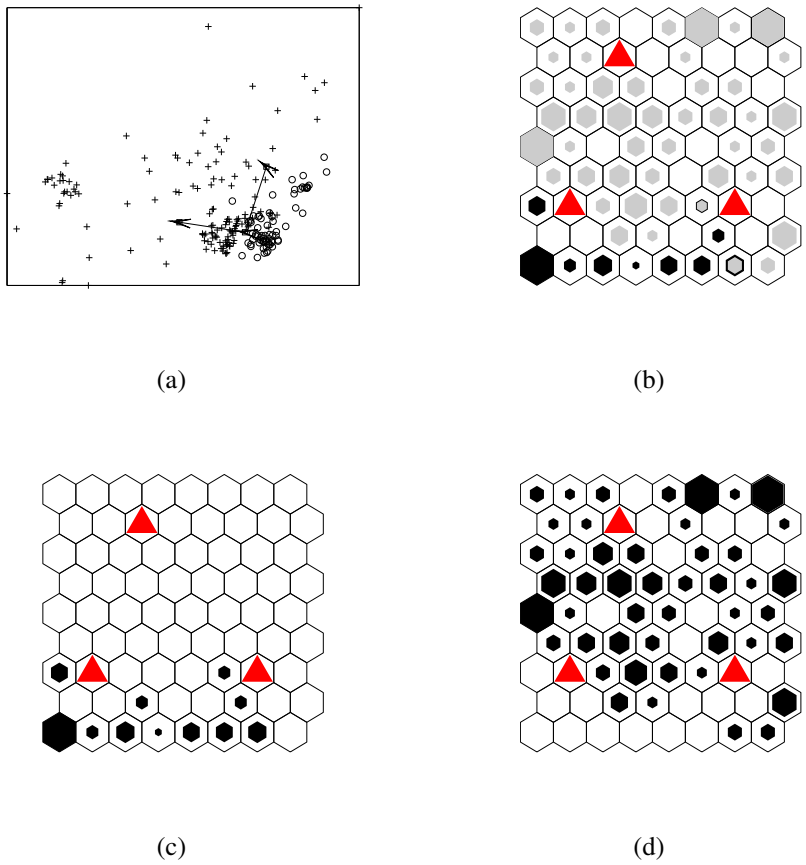


Figure 5.19: Projections of Decision Manifolds on Glass data set: (a) PCA projection, (b)–(d) hit histogram of 2 classes onto $\{8 \times 9\}$ SOM: (b) both classes, (c),(d) only one class

of one of the classes is depicted in Figure 5.19. As there are 7 different classes, the number of samples in each class for the two-class data sets is highly skewed. The PCA and SOM plots show that the smaller class is concentrated in one spot. The Glass data sets are fairly easy to classify, with most of the error rates under 3 %. Random Forests always achieve the best results, with Decision Manifolds reaching the same performance at 3 out of 7 data sets.

The Contraceptives data set lives in a 9 dimensional space and consists of 1473 samples in 3 classes. The PCA and SOM visualizations for one of the classes in Figure 5.20 do not show any easily identifyable separating boundary, but the classes appear more clustered with the SOM plot. The Decision Manifold performs especially good at this group of data sets, achieving the best results on all

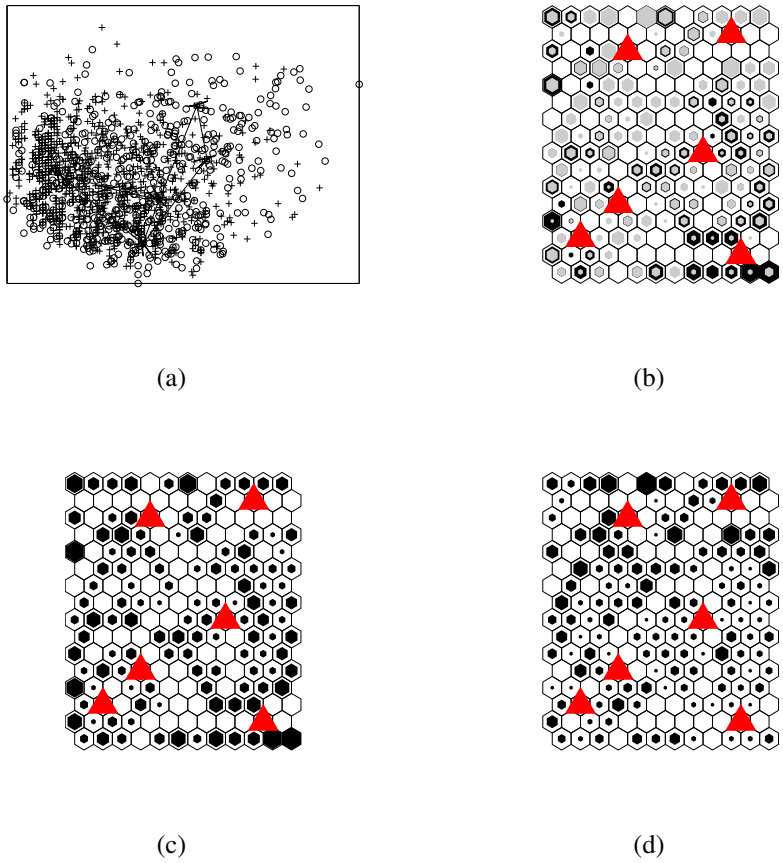


Figure 5.20: Projections of Decision Manifolds on Contraceptives data set: (a) PCA projection, (b)–(d) hit histogram of 2 classes onto $\{12 \times 16\}$ SOM: (b) both classes, (c),(d) only one class

three of them. The classes are not very well separated, resulting in high errors for all algorithms in the range of 20 – 35 %.

To summarize these findings, Decision Manifolds outperform the other algorithms on the Pima, Sonar, and Contraception data sets. Random Forests and linear SVMs each perform better than Decision Manifolds on 5 of the 10 data sets. k-NN, Decision Trees, and polynomial SVMs perform worse in most cases. RBF SVMs are better on 3, and worse on 3 data sets. Overall, there is no algorithm that outperforms every other on all the data sets, and the Decision Manifolds technique yields good classification results.

There is no clear trend on whether Decision Trees perform better than other classifier algorithms when the number of data samples is either high or low. As the

number of dimensions increases, Decision Manifolds seem to be outperformed by Random Forests more often, for example on Ionosphere and Spam. An exception to this are the results from the Sonar data set, which has the highest dimension, but also a very low number of points, which makes it atypical.

The total training times of the classifiers for all 10 folds including parameter tuning can also be seen in Table 5.5 as the numbers in brackets. In terms of training and classification time, the polynomial and RBF SVM models and k-NN are significantly slower at higher sample counts than the remaining algorithms. Decision Manifolds perform better than Random Forests, but worse than linear SVMs and Decision Trees in most cases. Overall, Decision Manifolds perform similar to the best performing classifiers in terms of accuracy, but are slightly faster than these. On the other hand, the Decision Manifolds algorithm is slower than algorithms with higher generalization errors.

5.5.3 Underlying Classifiers

Another experiment that has been conducted was comparing the performance of Decision Manifolds with different underlying linear classifiers. As any linear classifier can be used, the point of this experiment is to investigate the speed and accuracy differences between them. The classifiers that are tested in this section are described in Section 5.3.1. Moore-Penrose Pseudoinverse and Perceptron have performed almost equally in terms of classification accuracy, but training time was roughly twice as high with Perceptrons. For linear SVMs as underlying classifiers, Decision Manifolds have performed better, as one of the models tested is always topology $\{1\}$ which is equivalent to training a simple linear SVM, leading to a performance at least as good as shown in column "lin. SVM" in Table 5.5. However, the computational overhead is considerable as multiple models are estimated that involve repeatedly training linear classifiers, which are thus required to be fast. As a result, Decision Manifolds with SVMs are slower than any other model tested by a factor of up to four, rendering the choice of Decision Manifolds with SVMs infeasible. Further, apart from speed constraints, due to the fact that Perceptrons are stochastic and the Moore-Penrose Pseudoinverse is deterministic, choosing it as linear classifier renders the training algorithm deterministic apart from initialization, which is therefore recommended to be used with Decision Manifolds.

5.6 Summary

In this chapter, a classification algorithm for two-class problems by local approximation of the decision boundaries with a given topology has been proposed. It

consists of a number of nodes that can be represented by its position and a vector that indicates the classification direction of one of the classes. The nodes, or local classifiers, are arranged along the given topology structure. The training algorithm that moves the classifiers into the correct positions is inspired by the unsupervised SOM algorithm.

The Decision Manifold classifier has been shown to fit various low-dimensional non-trivially separable data sets. The examples have demonstrated its ability to approximate continuous non-linear boundaries, but have also highlighted that the algorithm has problems with decision boundaries that are split into many non-adjacent segments.

As the topology has to be given before training starts, a model selection scheme has been proposed. The topology is seen as the hyperparameter that can be tuned by trying different topology configurations that are determined by a PCA-guided approach. Thus, the model evaluates many different topologies, which is feasible due to the fast training time of a single Decision Manifold. Experiments on artificial and benchmark data sets have shown that the classifier performs comparable to state-of-the-art supervised learning algorithms.

Table 5.5: Benchmark data sets: Comparison of average 10-fold cross-validation error (in %) and training time

Data	Topol.	Dec. Mf.	lin. SVM	pol. SVM	RBF SVM	R. F.	Dec. Tr.	k-NN
Bupa (345×6)	$\{5 \times 2\}$	29.4 (4s)	29.6 (1s)	40.0 (4s)	30.1 (11s)	**27.2**$(10s)$	29.3 (1s)	33.9 (2s)
Pima (768×8)	$\{3 \times 2\}$	**20.9** (13s)	23.3 (2s)	25.3 (11s)	23.0 (50s)	23.3 (19s)	25.5 (1s)	25.0 (4s)
Spam (4601×57)	$\{7\}$	7.7 (207s)	7.2 (294s)	21.7 (2507s)	6.8 (11092s)	**4.7** (725s)	8.6 (15s)	9.2 (1986s)
Iono (351×34)	$\{3\}$	12.8 (4s)	11.7 (2s)	12.8 (9s)	**6.0** (32s)	6.3 (13s)	14.0 (1s)	14.8 (5s)
Heart (270×13)	$\{4\}$	18.5 (3s)	**17.0** (1s)	17.8 (5s)	18.5 (15s)	20.0 (9s)	19.6 (1s)	18.1 (3s)
Sonar (208×60)	$\{3 \times 3\}$	**14.3** (3s)	26.4 (3s)	18.3 (10s)	16.3 (29s)	14.8 (12s)	29.3 (2s)	16.8 (6s)
Credit (1000×20)	$\{5\}$	26.5 (31s)	24.8 (46s)	26.6 (199s)	**23.1** (607s)	**23.1** (85s)	27.1 (4s)	26.5 (91s)
Iris Set. (150×4)	$\{1\}$	**0.0** (1s)	**0.0** (1s)	**0.0** (1s)	**0.0** (3s)	**0.0** (2s)	**0.0** (1s)	**0.0** (1s)
Iris Ver. (150×4)	$\{3\}$	4.0 (1s)	25.3 (1s)	4.7 (1s)	**3.3** (3s)	4.0 (2s)	6.0 (1s)	4.0 (1s)
Iris Vir. (150×4)	$\{4\}$	**3.3** (1s)	4.0 (1s)	6.7 (1s)	**3.3** (3s)	6.0 (2s)	8.0 (1s)	4.7 (1s)
Glass 1 (214×10)	$\{3\}$	0.5 (3s)	0.5 (1s)	15.0 (2s)	0.9 (7s)	**0.0** (4s)	0.5 (1s)	1.9 (1s)
Glass 2 (214×10)	$\{3 \times 3\}$	1.4 (7s)	26.2 (1s)	11.2 (2s)	9.3 (7s)	**0.9** (4s)	1.4 (1s)	9.8 (1s)
Glass 3 (214×10)	$\{3\}$	**0.9** (5s)	3.3 (1s)	7.5 (2s)	4.2 (6s)	**0.9** (3s)	**0.9** (1s)	4.2 (1s)
Glass 5 (214×10)	$\{5\}$	**2.3** (5s)	7.0 (1s)	4.2 (1s)	3.3 (26s)	**2.3** (18s)	5.6 (3s)	5.1 (4s)
Glass 6 (214×10)	$\{3 \times 2\}$	2.3 (4s)	0.9 (1s)	3.7 (4s)	1.9 (16s)	**0.5** (7s)	2.3 (1s)	1.9 (3s)
Glass 7 (214×10)	$\{5\}$	**0.0** (5s)	3.3 (1s)	3.3 (4s)	3.3 (16s)	**0.0** (7s)	**0.0** (1s)	2.8 (3s)
Contrac. no (1473×9)	$\{3 \times 2\}$	**29.7** (63s)	32.2 (15s)	32.9 (61s)	29.8 (427s)	30.5 (68s)	**29.7** (2s)	35.0 (15s)
Contrac. l. (1473×9)	$\{3 \times 2\}$	**21.0** (58s)	22.6 (7s)	22.6 (42s)	22.2 (345s)	21.3 (75s)	22.5 (2s)	25.3 (13s)
Contrac. sh. (1473×9)	$\{5\}$	**32.1** (60s)	34.7 (9s)	34.7 (47s)	32.3 (336s)	33.6 (62s)	33.5 (1s)	36.6 (13s)

Chapter 6

Conclusion

In this thesis, three contributions related to data mining with Self-Organizing Maps have been presented.

The first contribution, the Graph visualization method, helps in understanding the relation between the data samples and the map. The method is computed by introducing a measure of similarity between the data samples. A graph structure is constructed that represents this proximity, with one vertex per data vector, and an edge connecting data vectors that are close. This graph is then transferred onto the map by projection in the same way that the usual SOM mapping is performed. The resulting patterns on the map reveal much about the clustering structure and the connections of the data cloud. Specifically, the Graph method is useful for identifying

- the density of clusters and other regions, which can differ widely across different regions of the feature space.
- topology violations, which occur due to the dimensionality reduction by mapping the feature space to the output space. The method also tells, to a certain degree, where the neighborhood on the map is insufficient to depict the topological features of the data cloud.
- the connectivity of data samples large maps, where the data samples are outnumbered by the map units, and for revealing outliers.

Two different methods for calculating the proximity graph have been proposed. The radius method is based on the spherical distance around the data samples, and the nearest neighbors method is based on ranking of the nearest neighbors. Both approaches require parameterization and are best used by providing a series of visualizations of this kind at increasing parameter values.

The usefulness of the Graph method has been demonstrated with the help of a series of artificial, benchmark, and real-world data sets. This method can be combined with other SOM visualizations, such as the U-Matrix, hit histograms, and component planes. By drawing a series of radius method visualizations at increasing threshold levels, the clustering structure can be shown along with the clusters' relations to each other. The nearest neighbors method is used to show topology violations and to unveil points along low-dimensional manifolds hidden in the data set. It is best used together with the Topographic error. No other technique is able to find structure such as samples aligned along a curve in the data.

The second contribution of this thesis has been the Gradient Field method and its dual representation, the Borderline method. This method has been developed to satisfy the need for SOM visualization methods designed for specialists with engineering rather than computing backgrounds. It shows the SOM in a way that the beholder is not tempted to mistake the axes of the map as being significant. The information within the map is communicated through the analogy of the visualization of vector fields, with arrows pointing towards the interiors of clusters. The length of the vectors is related to a map units position relative to the cluster centers.

An important advantage of the Gradient Field method is that it is able to display the clustering structure that underlies the SOM at various levels of detail. This level can be parameterized by setting the width of the kernel function. The kernel is used for smoothing over the region in the vicinity of each map unit. Units within regions that are highly similar to each other are identified as cluster centers. The clusters roughly correspond to the results of hierarchical clustering algorithms, where the threshold distance for the clustering is inversely related to the kernel width parameter.

Further, an extension has been proposed for dealing with groups of variables, or component planes. This grouping can either be based on similarity or on semantic meaning. As subsets of the component planes are themselves SOMs, the Gradient Field method can be calculated and displayed simultaneously for various groups of variables. The reason for doing this is to learn about the mutual dependencies of variable dimensions. As not all mutual dependencies can be captured by measures such as the linear correlation, this technique is useful for uncovering where and how a variable or a group of variable explains another variable. Another use of this method is to explain the contributing factors for each cluster.

The third major contribution of this thesis is the Decision Manifolds method for binary classification. This technique exploits the SOM training algorithm's

ability to align neighboring units according to a super-imposed topology structure. The units from the traditional SOM algorithm are replaced by local linear classifiers that classify the data samples within their proximity. The feature space is thus partitioned into several disjoint regions, each of which is divided into its two classes by separating hyperplanes.

As the Decision Manifold's capability to approximate the decision hypersurface is dependent on the matching between the topological structure of both the hypersurface and the Decision Manifold, a model selection scheme has been proposed in order to estimate the correct topology of the separating border. This model estimation is guided by PCA, which is a heuristic for guessing the upper bound of the number of relevant dimensions for describing the structure of the data cloud.

There are several features that make the Decision Manifolds unique. The separating hypersurface is approximated linearly, resulting in an explicit functional representation of the classifier. Further, similar to SOMs, there is a kernel function that steers the degree of mutual influence between neighboring classifiers. The width of this kernel decreases monotonously as training progresses. After training is finished, this parameter can be used to optimize classification performance.

Extensive testing has been performed to benchmark the Decision Manifold classifier against comparable methods. Decision Manifolds are found to perform favorably both in classification performance and computational efficiency in comparison to state-of-the-art machine learning methods such as Support Vector Machines or Random Forests.

In order to bring this thesis to an end, I hope that these contributions have advanced the area of data mining. Finally, I hope that the data sets that have been carefully designed to be characteristic for specific data mining problems will find their way into the general repository of the machine learning community.

Bibliography

[1] Charu C. Aggarwal, Alexander Hinneburg, and Daniel A. Keim. On the surprising behavior of distance metrics in high dimensional spaces. In *ICDT '01: Proceedings of the 8th International Conference on Database Theory*, pages 420–434, London, UK, 2001. Springer-Verlag.

[2] Charu C. Aggarwal and Philip S. Yu. Outlier detection for high dimensional data. *SIGMOD Rec.*, 30(2):37–46, 2001.

[3] Edgar Anderson. The irises of the gaspe peninsula. *Bulletin of the American Iris Society*, 59:2–5, 1935.

[4] Mohammed Attik, Laurent Bougrain, and Frédéric Alexandre. Self-organizing map initialization. In *Artificial Neural Networks: Biological Inspirations ICANN 2005*, pages 357–362, 2005.

[5] H.-U. Bauer and T. Villmann. Growing a hypercubical output space in a self-organizing feature map. *IEEE Transactions on Neural Networks*, 8(2):218–226, 1997.

[6] Hans-Ulrich Bauer and Klaus R. Pawelzik. Quantifying the neighborhood preservation of self-organizing feature maps. *IEEE Transactions on Neural Networks*, 3(4):570–579, July 1992.

[7] Pavel Berkhin. Survey of clustering data mining techniques. Technical report, Accrue Software, 2002.

[8] C. Bishop, M. Svensen, and C. Williams. Magnification factors for the GTM algorithm. In *Workshop on Self-Organizing Maps (WSOM'97)*, pages 333–338, 1997.

[9] Christopher M. Bishop. *Neural networks for pattern recognition*. Oxford University Press, Oxford, UK, 1996.

[10] C.M. Bishop, M. Svensen, and C. Williams. GTM: The generative topographic mapping. *Neural Computation*, 10(1):215–234, 1998.

[11] L. Bottou and V. Vapnik. Local learning algorithms. *Neural Computation*, 4:888–900, 1992.

[12] P. S. Bradley and U. M. Fayyad. Refining initial points for k-means clustering. In *Proceedings of the International Conference on Machine Learning (ICML'99)*, pages 91–99, 1999.

[13] Leo Breiman. Random Forests. *Machine Learning*, 45:5–32, 2001.

[14] Leo Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and regression trees*. Wadsworth & Brooks-Cole Advanced Books & Software, Monterey, CA, 1984.

[15] Christopher J. C. Burges. A tutorial on Support Vector Machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[16] J. Cerquides and R. López de Mántaras. Robust Bayesian linear classifier ensembles. In *European Conference on Machine Learning (ECML'05)*, pages 72–83, 2005.

[17] H. Chernoff and M. H. Rizvi. Effect on classification error or random permutations of features in representing multivariate data by faces. *Journal of American Statistical Association*, 70:548–554, 1975.

[18] D.L. Davies and D.W. Bouldin. A cluster separation measure. *IEEE Transactions on Pattern Recognition and Machine Intelligence*, 1(2):224–227, 1979.

[19] Pierre Demartines and Jeanny Hérault. Curvilinear component analysis: A self-organizing neural network for nonlinear mapping of data sets. *IEEE Transactions on Neural Networks*, 8(1):148–154, January 1997.

[20] D. L. Donoho. High-dimensional data analysis: The curses and blessings of dimensionality. Lecture on August 8, 2000, to the American Mathematical Society "Math Challenges of the 21st Century", 2000.

[21] Edgar Erwin, Klaus Obermayer, and Klaus Schulten. Self-organizing maps: Ordering, convergence properties and energy functions. *Biological Cybernetics*, 67(1):47–55, 1992.

[22] Ronald A. Fisher. The use of multiple measurements in taxonomic problems. *The Annals of Eugenics*, 7:179–188, 1936.

[23] W. J. Frawley, G. Piatetsky-Shapiro, and C. J. Matheus. Knowledge discovery in databases – an overview. *AI Magazine*, 13:57–70, 1992.

[24] Xin Geng, De-Chuan Zhan, and Zhi-Hua Zhou. Supervised nonlinear dimensionality reduction for visualization and classification. *IEEE Transactions on Systems, Man and Cybernetics – Part B: Cybernetics*, 35(6):1098–1107, 2005.

[25] Geoffrey J. Goodhill and Terrence J. Sejnowsky. Quantifying neighborhood preservation in topographic mappings. In *Proceedings of the 3rd Joint Symposium on Neural Computation*, pages 61–82, University of California, San Diego and California Institute of Technology, 6, Pasadena, CA: California Institute of Technology, 1996. Springer, Berlin.

[26] R. P. Gorman and T. J. Sejnowski. Analysis of hidden units in a layered network trained to classify sonar targets. *Neural Networks*, 1:75–89, 1988.

[27] Thore Graepel, Matthias Burger, and Klaus Obermayer. Phase transitions in stochastic selforganizing maps. *Physical Review E*, 56:3876–3890, 1997.

[28] Georges Grinstein, Marjan Trutschl, and Urska Cvek. High-dimensional visualizations. In *Data Mining Conference KDD Workshop 2001*, pages 7–19, San Francisco, CA, USA, 2001. ACM Press, New York.

[29] Harmen grosse Deters, Wiebke Timm, and Tim Wilhelm Nattkemper. Reefsom - a metaphoric data display for exploratory data mining. *Brains, Minds and Media*, 2, Apr 2006.

[30] Jun Wen Guihua Wen, Lijun Jiang and Nigel R. Shadbolt. Clustering-based nonlinear dimensionality reduction on manifold. In *Pacific Rim International Conference on Artificial Intelligence (PRICAI 2006), LNAI 4099*, pages 444–453. Springer, 2006.

[31] M. Herrmann H.-U. Bauer, R. Der. Controlling the magnification factor of self-organizing feature maps. *Neural Computation*, 8(4):757–771, 1996.

[32] Maria Halkidi, Yannis Batistakis, and Michalis Vazirgiannis. On clustering validation techniques. *Journal of Intelligent Information Systems*, 17(2–3):107–145, 2001.

[33] J. A. Hartigan and M. A. Wong. A K-means clustering algorithm. *Applied Statistics*, 28:100–108, 1979.

[34] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer-Verlag, 2001.

[35] J. Himberg. Enhancing SOM-based data visualization by linking different data projections. In *International Symposium on Intelligent Data Engineering and Learning (IDEAL'98)*, 1998.

[36] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[37] A. Inselberg and B. Dimsday. Parallel coordinates: A tool for visualizing multidimensional geometry. In *Proceedings of IEEE Conference on Visualization*, pages 361–378, 1990.

[38] R. A. Jacobs and M. I. Jordan. Hierarchical mixtures of experts and the EM algorithm. *Neural Computation*, 6:181–214, 1994.

[39] A. Jain, M. Murty, and P. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–323, 1999.

[40] W.C. Dickson W.C. Knowler R.S. Johannes J.W. Smith, J.E. Everhart. Using the adap learning algorithm to forecast the onset of diabetes mellitus. In *Symposium on Computer Applications and Medical Care*, pages 261–265, 1988.

[41] Samuel Kaski and Krista Lagus. Comparing SelfOrganizing Maps. In *Proceedings of the International Conference on Artifical Neural Networks (ICANN'96)*, pages 809–814. Springer, Berlin, 1996.

[42] Samuel Kaski, Janne Nikkilä, and Teuvo Kohonen. Methods for exploratory cluster analysis. In *International Conference on Advances in Infrastructure for Electronic Business, Science, and Education on the Internet*, L'Aquila, Rome, Italy, 2000.

[43] Leonard Kaufman and Peter J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley & Sons, Inc., 1990.

[44] R. Khoussainov, A. Hess, and N. Kushmerick. Ensembles of biased classifiers. In *International Conference on Machine Learning (ICML'05)*, pages 425–432, 2005.

[45] J. Kiviluoto. Topology preservation in selforganizing maps. In *Proceedings of the International Conference on Neural Networks (ICNN'96)*, pages 296–299, Piscataway, New Jersey, USA, 1996.

[46] Teuvo Kohonen. Self-organized formation of topologically correct feature maps. *Biological Cybernetics*, 43:59–69, 1982.

[47] Teuvo Kohonen. Improved versions of learning vector quantization. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN'90)*, pages 545–550, 1990.

[48] Teuvo Kohonen. *Self-Organizing Maps*, chapter "Optimization Approaches", pages 981–990. Elsevier Science Publishers, 1991.

[49] Teuvo Kohonen. The Self-Organizing Map. *Neurocomputing*, 21:1–6, 1998.

[50] Teuvo Kohonen. *Self-Organizing Maps, 3rd edition*. Springer, 2001.

[51] Petri Kontkanen, Jussi Lahtinen, Petri Myllymäki, Tomi Silander, and Henry Tirri. Supervised model-based visualization of high-dimensional data. *Intelligent Data Analysis*, 4:213–227, 2000.

[52] Sanjeev Kulkarni, Gabor Lugosi, and Santosh Venkatesh. Learning pattern classification – a survey. *IEEE Transactions on Information Theory*, 44(6):2178–2206, 1998.

[53] Jouko Lampinen and Erkki Oja. Clustering properties of hierarchical self-organizing maps. *Journal of Mathematical Imaging and Vision*, 2(2–3):261–272, 1992.

[54] Khalid Latif and Rudolf Mayer. Sky-metaphor visualisation for self-organising maps. In *Proceedings of the 7th International Conference on Knowledge Management (I-KNOW'07)*, Graz, Austria, September 5 - 7 2007.

[55] Yoseph Linde, Andres Buzo, and Robert M. Gray. An algorithm for vector quantizer design. *IEEE Transactions on Communications*, 28(1):84–95, 1980.

[56] M. E. Maron. Automatic indexing: An experimental inquiry. *Journal of the ACM*, 8(3):404–417, 1961.

[57] Rudolf Mayer, Thomas Lidy, and Andreas Rauber. The map of mozart. In *Proceedings of the 7th International Conference on Music Information Retrieval (ISMIR)*, pages 351–352, October 8-12 2006.

[58] Rudolf Mayer, Dieter Merkl, and Andreas Rauber. Mnemonic soms: Recognizable shapes for self-organizing maps. In Marie Cottrell, editor, *Proceedings of the Fifth Workshop on Self-Organizing Maps (WSOM'05)*, pages 131–138, Paris, France, September 5–8 2005.

[59] Tom Mitchell. *Machine Learning*. McGraw Hill, 1997.

[60] J. Moody and C. J. Darkin. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1(2):281–294, 1989.

[61] Robert Neumayer, Rudolf Mayer, Georg Pölzlbauer, and Andreas Rauber. The metro visualisation of component planes for self-organising maps. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN'07)*, Orlando, FL, USA, August 12 - 17 2007. IEEE Computer Society.

[62] Robert Neumayer, Rudolf Mayer, and Andreas Rauber. Component selection for the metro visualisation of the som. In *Proceedings of the 6th International Workshop on Self-Organizing Maps (WSOM'07)*, Bielefeld, Germany, September 3 - 6 2007.

[63] E. Pampalk, A. Rauber, and D. Merkl. Content-based Organization and Visualization of Music Archives. In *Proceedings of the ACM Multimedia*, pages 570–579, Juan les Pins, France, December 1-6 2002. ACM.

[64] E. Pampalk, A. Rauber, and D. Merkl. Using smoothed data histograms for cluster visualization in self-organizing maps. In *Proceedings of the International Conference on Artifical Neural Networks (ICANN'02)*, Madrid, Spain, 2002. Springer.

[65] J. M. Pena, J.A. Lozano, and P. Larranaga. An empirical comparison of four initialization methods for the k-means algorithm. *Pattern Recognition Letters*, 20:1027–1040, 1999.

[66] Daniel Polani. Measures for the organization of self-organizing maps. In Udo Seiffert and Lakhmi C. Jain, editors, *Self-Organizing Neural Networks: Recent Advances and Applications*, pages 13–44. Physica-Verlag, 2002.

[67] Georg Pölzlbauer. Survey and comparison of quality measures for self-organizing maps. In Ján Paralič, Georg Pölzlbauer, and Andreas Rauber, editors, *Proceedings of the Fifth Workshop on Data Analysis (WDA'04)*, pages 67–82, Sliezsky dom, Vysoké Tatry, Slovakia, June 24–27 2004. Elfa Academic Press.

[68] Georg Pölzlbauer. Visualization of data from the petroleum industry with vector fields on top of self-organizing maps. In Peter Bednár, Tomáš Horváth, Ján Paralič, and Andreas Rauber, editors, *Proceedings of the Sixth*

Workshop on Data Analysis (WDA'05), pages 42–49, Abaújszántó, Tokaj-region, Hungary, June 20–22 2005. Elfa Academic Press.

[69] Georg Pölzlbauer, Michael Dittenbach, and Andreas Rauber. Gradient visualization of grouped component planes on the som lattice. In Marie Cottrell, editor, *Proceedings of the Fifth Workshop on Self-Organizing Maps (WSOM'05)*, pages 331–338, Paris, France, September 5–8 2005.

[70] Georg Pölzlbauer, Michael Dittenbach, and Andreas Rauber. A visualization technique for self-organizing maps with vector fields to obtain the cluster structure at desired levels of detail. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN'05)*, pages 1558–1563, Montreal, Canada, July 31 – August 5 2005. IEEE Computer Society.

[71] Georg Pölzlbauer, Michael Dittenbach, and Andreas Rauber. Advanced visualization of self-organizing maps with vector fields. *Neural Networks*, 19(6–7):911–922, July–August 2006.

[72] Georg Pölzlbauer, Thomas Lidy, and Andreas Rauber. Decision manifolds: Classification inspired by self-organization. In *Proceedings of the Sixth Workshop on Self-Organizing Maps (WSOM'07)*, Bielefeld, Germany, September 3–6 2007.

[73] Georg Pölzlbauer, Thomas Lidy, and Andreas Rauber. Decision manifolds – a supervised learning algorithm based on self-organization. *accepted for publication in IEEE Transactions on Neural Networks*, 19(9):1518–1530, September 2008.

[74] Georg Pölzlbauer, Andreas Rauber, and Michael Dittenbach. Advanced visualization techniques for self-organizing maps with graph-based methods. In Zhang Yi Jun Wang, Xiaofeng Liao, editor, *Proceedings of the Second International Symposium on Neural Networks (ISNN'05)*, pages 75–80, Chongqing, China, May 30 – June 1 2005. Springer-Verlag.

[75] Georg Pölzlbauer, Andreas Rauber, and Michael Dittenbach. Graph projection techniques for self-organizing maps. In Michel Verleysen, editor, *Proceedings of the European Symposium on Artificial Neural Networks (ESANN'05)*, pages 533–538, Bruges, Belgium, April 27–29 2005. d-side publications.

[76] Georg Pölzlbauer, Andreas Rauber, and Michael Dittenbach. A SOM-view of oilfield data: A novel vector field visualization for self-organizing maps and its applications in the petroleum industry. In Klaus Tochtermann

and Hermann Maurer, editors, *Proceedings of the Fifth International Conference on Knowledge Management (I-KNOW'05)*, pages 502–509, Graz, Austria, June 29 – July 1 2005. J.UCS - Journal of Universal Computer Science.

[77] Georg Pölzlbauer, Andreas Rauber, and Michael Dittenbach. A vector field visualization technique for self-organizing maps. In Huan Li Tu Bao Ho, David Cheung, editor, *Proceedings of the Ninth Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'05)*, pages 399–409, Hanoi, Vietnam, May 18–20 2005. Springer-Verlag.

[78] Dorian Pyle. *Data Preparation for Data Mining*. Morgan Kaufmann, 1999.

[79] J. R. Quinlan. Discovering rules by induction from large collections of examples. In D. Michie, editor, *Expert Systems in the Micro-Electronic Age*, pages 168–201. Edinburgh University Press, Edinburgh, 1979.

[80] J. Ross Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[81] W. Steinbrunn M. Pfisterer J. Schmid S. Sandhu K. Guppy S. Lee V. Froelicher R. Detrano, A. Janosi. International application of a new probability algorithm for the diagnosis of coronary artery disease. *American Journal of Cardiology*, 64:304–310, 1989.

[82] D. De Ridder, O. Kouropteva, O. Okun, M. Pietikainen, and R. Duin. Supervised locally linear embedding. In *Joint International Conference on Artificial Neural Networks ICANN/ICONIP, LNCS 2714*, pages 333–341, 2003.

[83] Helge Ritter and Klaus Schulten. On the stationary state of Kohonen's self-organizing sensory mapping. *Biological Cybernetics*, 54:99–106, 1986.

[84] Frank Rosenblatt. The Perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[85] Sam T. Roweis, Lawrence K. Saul, and Geoffrey E. Hinton. Global coordination of local linear models. In *Neural Information Processing Systems*, pages 889–896, 2001.

[86] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(9):533–536, 1986.

[87] S. L. Salzberg. On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Mining and Knowledge Discovery*, 1(3):317–328, 1997.

[88] John W. Sammon. A nonlinear mapping for data structure analysis. *IEEE Transactions on Computers*, C-18(5):401–409, May 1969.

[89] L.K. Saul and S. T. Roweis. Think globally, fit locally: Unsupervised learning of low dimensional manifolds. *Journal of Machine Learning Research*, 4:119–155, 2003.

[90] Claude Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 1948.

[91] V. Sigillito, S. Wing, L. Hutton, and K. Baker. Classification of radar returns from the ionosphere using neural networks. *Johns Hopkins APL Technical Digest*, 10:262–266, 1989.

[92] V.D. Silva and J. B. Tenenbaum. Global versus local methods in nonlinear dimensionality reduction. In *Neural Information Processing Systems*, pages 705–712, 2003.

[93] Jack Sklansky and Leo Michelotti. Locally trained piecewise linear classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2(2):101–111, 1980.

[94] André Skupin. A picture from a thousand words. *Computing in Science and Engineering*, 6(5):84–88, 2004.

[95] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[96] Jari Kangas Jorma Laaksonen Teuvo Kohonen, Jussi Hynninen and Kari Torkkola. Lvq_pak: The learning vector quantization program package. Technical report, Helsinki University of Technology, Laboratory of Computer and Information Science, FIN-02150 Espoo, Finland, 1996.

[97] P. F. Thall and S. C. Vail. Some covariance models for longitudinal count data with over-dispersion. *Biometrics*, 46:657–671, 1990.

[98] P. Tino, I. Nabney, and Y. Sun. Using directional curvatures to visualize folding patterns of the GTM projection manifolds. In *International Conference on Artificial Neural Networks (ICANN'01)*, pages 421–428. Springer, 2001.

[99] W.S. Torgerson. Multidimensional scaling: I. Theory and method. *Psychometrika*, 17:401–419, 1952.

[100] Edward Tufte. *The visual display of quantitative information*. Graphics Press, 2001.

[101] Alfred Ultsch. Data mining and knowledge discovery with emergent self-organizing feature maps for multivariate time series. In Erkki Oja and Samuel Kaski, editors, *Kohonen Maps*, pages 33–46. Elsevier Science, 1999.

[102] Alfred Ultsch. Maps for the visualization of high-dimensional data spaces. In *Proceedings of the Workshop on Self organizing Maps*, Kyushu, Japan, 2003.

[103] Alfred Ultsch. U*-matrix: a tool to visualize clusters in high dimensional data. Technical report, Departement of Mathematics and Computer Science, Philipps-University Marburg, 2003.

[104] Alfred Ultsch and H. Peter Siemon. Kohonen's self-organizing feature maps for exploratory data analysis. In *Proceedings of the International Neural Network Conference (INNC'90)*, pages 305–308. Kluwer, 1990.

[105] Paul E. Utgoff. Incremental induction of decision trees. *Machine Learning*, 4:161–186, 1989.

[106] C. J. Van Rijsbergen. *Information Retrieval, 2nd edition*. Dept. of Computer Science, University of Glasgow, 1979.

[107] V. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.

[108] Michel Verleysen. *Limitations and future trends in neural computation*, chapter Learning high-dimensional data, pages 141–162. IOS Press, 2003.

[109] J. Vesanto. SOM-based data visualization methods. *Intelligent Data Analysis*, 3(2):111–126, 1999.

[110] J. Vesanto. *Data Exploration Process Based on the Self-Organizing Map*. PhD thesis, Helsinki University of Technology, 2002.

[111] Juha Vesanto and Jussi Ahola. Hunting for correlations in data using the self-organizing map. In *International ICSC Congress on Computational Intelligence Methods and Applications (CIMA'99)*, pages 279–285. ICSC Academic Press, 1999.

[112] Juha Vesanto and Esa Alhoniemi. Clustering of the self-organizing map. *IEEE Transactions on Neural Networks*, 11(3):586–600, 2000.

[113] Juha Vesanto, Johan Himberg, Esa Alhoniemi, and Juha Parhankangas. Self-organizing map in Matlab: The SOM toolbox. In *Proceedings of the Matlab DSP Conference 1999*, pages 35–40, 1999.

[114] Juha Vesanto, Mika Sulkava, and Jaakko Hollmén. On the decomposition of the self-organizing map distortion measure. In *Proceedings of the Workshop on Self organizing Maps (WSOM'03)*, pages 11–16, Hibikino, Kitakyushu, Japan, 2003.

[115] Thomas Villmann and Jens Christian Claussen. Investigation of magnification control in Self-Organizing Maps and Neural Gas. *Neural Computation*, 18(2):446–469, 2006.

[116] Thomas Villmann, Ralf Der, Michael Herrmann, and Thomas M. Martinetz. Topology preservation in self-organizing feature maps: Exact definition and measurement. *IEEE Transactions on Neural Networks*, 8(2):256–266, 1997.

[117] M. Vlachos, C. Domeniconi, D. Gunopulos, G. Kollios, and N. Koudas. Non-linear dimensionality reduction techniques for classification and visualization. In *8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 645–651, 2002.

[118] J. H. Ward. Hierarchical grouping to optimize a bijective function. *Journal of the American Statistical Association*, 58:236–244, 1963.

[119] P.D. Wasserman. *Advanced Methods in Neural Computing*. New York: Van Nostrand Reinhold, 1993.

[120] Guihua Wen, Lijun Jiang, and Nigel R. Shadbolt. Using graph algebra to optimize neighborhood for isometric mapping. In *Proceedings of the International Joint Conference of Artificial Intelligence*, pages 2398–2403, 2007.

[121] Paul John Werbos. *The roots of backpropagation: From ordered derivatives to neural networks and political forecasting*. Wiley-Interscience, New York, NY, USA, 1994.

[122] K. Woods, W. P. Kegelmeyer, and K. Bowyer. Combination of multiple classifiers using local accuracy estimates. *IEEE Transactions on Pattern Analalysis and Machine Intelligence*, 19(4):405–410, 1997.

[123] Georg Zangl and Josef Hannerer. *Data Mining: Applications in the Petroleum Industry*. Round Oak Publishing, 2003.

Appendix A

Notational conventions

$\|\cdot\|$	Euclidean norm
$\mathfrak{S}$	a set
$\|\mathfrak{S}\|$	cardinality of S, i.e. the number of elements of a set
$\mathbf{x}$	a vector
$\tilde{\mathbf{x}}$	a homogeneous vector with bias coordinate
x_i	i-th element of vector $\mathbf{x}$
$\mathbf{X}$	a matrix
$\mathbf{x}_i$	i-th row vector of matrix $\mathbf{X}$
$\mathbf{x}_{(j)}$	j-th column vector of matrix $\mathbf{X}$
$\mathbf{X}^{\mathrm{t}}$	transpose of matrix $\mathbf{X}$
$\tilde{\mathbf{X}}$	a matrix in homogeneous coordinates, i.e. with bias coordinates in the first row

Appendix B

Abbreviations

ANN	Artificial Neural Network
BMU	Best-Matching Unit
DB-Index	Davies Bouldin Index
k-NN	k-Nearest Neighbors
LBG	Linde-Buzo-Gray Batch k-means algorithm
LDA	Linear Discriminant Analysis
MDS	Multi-Dimensional Scaling
PCA	Principal Component Analysis
pdf	probability density function
RBF	Radial Basis Function
SOM	Self-Organizing Map
SVM	Support Vector Machine
SDH	Smoothed Data Histograms

Appendix C

Additional definitions and formulas

C.1 Definition of nearest neighbors

For a strictly formal definition that fits into the framework of the rest of this thesis, the k-th nearest neighbor is defined in this section, along with abbreviations for commonly used calculations. The index of the k-th nearest vector, contained in a set of vectors denoted as matrix $\mathbf{X}$, of vector $\mathbf{v}$ is written as

$$N^{(k)}(\mathbf{v}, \mathbf{X}) = \arg \min_{i \in \mathfrak{S}^{(k)}(\mathbf{v}, \mathbf{X})} \|\mathbf{v} - \mathbf{x}_i\|, \tag{C.1}$$

where the set $\mathfrak{S}^{(k)}$ contains the indices of the available vectors, which is defined recursively by removing the vectors from the pool of possible candidates:

$$\mathfrak{S}^{(1)}(\mathbf{v}, \mathbf{X}) = \{i \,|\, \mathbf{x}_i \in \mathbf{X}\}, \tag{C.2}$$

$$\mathfrak{S}^{(k)}(\mathbf{v}, \mathbf{X}) = \mathfrak{S}^{(k-1)}(\mathbf{v}, \mathbf{X}) \setminus \{N^{(k-1)}(\mathbf{v}, \mathbf{X})\}. \tag{C.3}$$

A special case is the nearest neighbor, denoted by omitting the superscript

$$N(\mathbf{v}, \mathbf{X}) = N^{(1)}(\mathbf{v}, \mathbf{X}) = \arg_i \min_{\mathbf{x}_i \in \mathbf{X}} \|\mathbf{v} - \mathbf{x}_i\|. \tag{C.4}$$

To provide a less cumbersome notation for the most commonly needed calculations, shortcuts are defined for the best-matching unit, the k-th best matching unit, and the k-th nearest neighbor of a vector out of a set that contains the vector itself. The index of the k-th best matching unit $I^{(k)}(\mathbf{x}_i)$, and the best matching unit $I(\mathbf{x}_i)$ of sample $\mathbf{x}_i$ and codebook $\mathbf{M}$ are abbreviated as

$$I^{(k)}(\mathbf{x}_i) = N^{(k)}(\mathbf{x}_i, \mathbf{M}), \tag{C.5}$$

$$I(\mathbf{x}_i) = I^{(1)}(\mathbf{x}_i). \tag{C.6}$$

The k-nearest prototype vector within the codebook, and the k-nearest data sample within the data set are written as

$$N_{\mathbf{X}}^{(k)}(\mathbf{x}_i) = N^{(k)}(\mathbf{x}_i, \mathbf{X} \setminus \{\mathbf{x}_i\}) \tag{C.7}$$

$$N_{\mathbf{M}}^{(k)}(\mathbf{m}_j) = N^{(k)}(\mathbf{m}_j, \mathbf{M} \setminus \{\mathbf{m}_j\}) \tag{C.8}$$

Appendix D

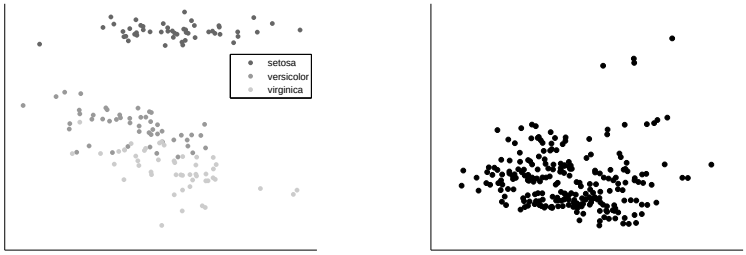
Data sets

D.1 Benchmark data sets

In this section, the benchmark data sets used in this thesis are listed and their characteristics explained, along with a PCA projection for each of them. The first couple of data sets are unlabeled and are therefore solely used for unsupervised learning. These are used in Chapters 2, 3, and 4 for visualization and clustering. In the latter part of this section, the data sets for supervised learning are described. Each of them has 2 levels, and they are used in Chapter 5 for classification, but are also used for visualization.

D.1.1 Iris

Reference	Originally collected by Anderson [3], this data set has been introduced by Fisher [22] to the statistics community.
Number of samples	150
Attributes	4 (sepal length, sepal width, petal length, petal width)
Labels	Setosa, Versicolor, Virginica; each of these classes has 50 samples.
Description	The Iris data set is maybe the best known data set in the data mining community. The Setosa class is linearly separable from the others, while the Virginica and Versicolor classes are overlapping to some degree. The PCA plot in Figure D.1(a) explains 95% of the variance.

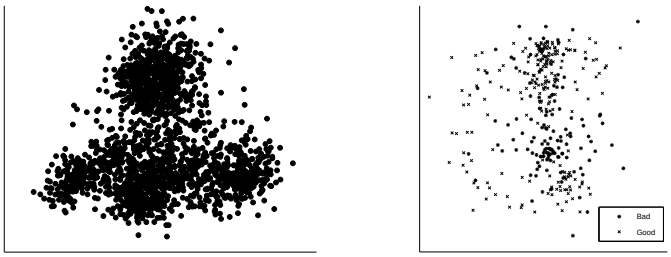


(a) (b)

Figure D.1: PCA for (a) Iris (95% of variance explained), (b) Epileptics (56% of variance explained)

D.1.2 Epileptics

Reference	The epileptics data set has been collected in 1990 by Thall and Vail [97].
Number of samples	236
Attributes	8
Labels	Several categorical variables, for example "treatment": Placebo (47.5%), Progabide (52.5%)
Description	This data set describes the seizure count of 59 epileptics over periods of 2 weeks. The experiments have been conducted over 4 such periods, resulting in 236 measurements. The data set has been originally intended for prediction of the number of seizures in a regression setting. In this thesis, the predicted variable is omitted, and the data set is used for unsupervised methods only. The variables include binary and integer variables, such as the subject's age and an indicator of whether a placebo has been used. Figure D.1(b) shows a PCA plot of the data. The first two axes explain 56% of the variance in this data set.

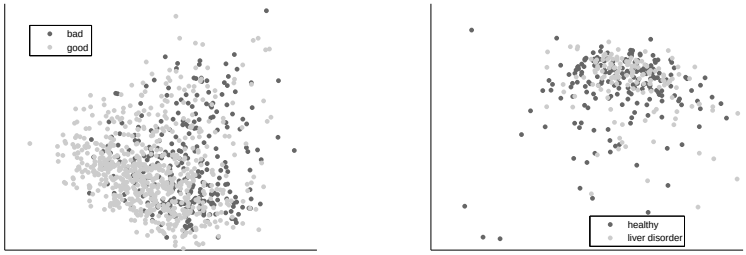


(a) (b)

Figure D.2: PCA for (a) Phonetic (29% of variance explained), (b) Ionosphere (39% of variance explained)

D.1.3 Phonetic

Reference	The phonetic data set is bundled with the LVQ-PAK software [96].
Number of samples	1962 (for each of 2 data sets)
Attributes	20
Labels	20 different phonemes from Finnish language.
Description	The data set consists of 1962 cepstral-coefficient vectors picked up from continuous Finnish speech from the same speaker. The samples are labeled into 20 classes. Phonemes of the same label are usually clustered in feature space. The PCA plot is shown in Figure D.2(a) and explains 29% of the variance in the data set, making it rather unsuitable for visualiztion.

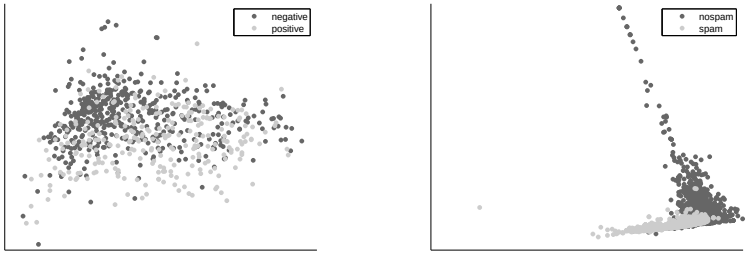


(a) (b)

Figure D.3: PCA for (a) German Credit (11% of variance explained), (b) Bupa Liver Disorder (57% of variance explained)

D.1.4 Ionosphere

Reference	The data has been collected in Goose Bay, Labrador, by Sigillito, Wing, Hutton, and Baker [91].
Number of samples	351
Attributes	34 (all numeric)
Labels	Good (64.1%) and Bad (35.9%)
Description	The data set describes the results from a radar system. The attributes correspond to a phased array of 16 high-frequency antennas. The targets are free electrons in the ionosphere, and samples labeled as "good" correspond to evidence of structure in the ionosphere. Figure D.2(b) shows the PCA plot of the Ionosphere data. It explains 39% of the variance in the data set. This hints at a data space where the attributes are stronger correlated than for example the phonetic data set. The phonetic data consisted of 20 attributes and the PCA of the two most important eigenvectors explained only 29% of the variance in the data, compared to the 34-dimensional ionosphere data set where the PCA explains 39%.

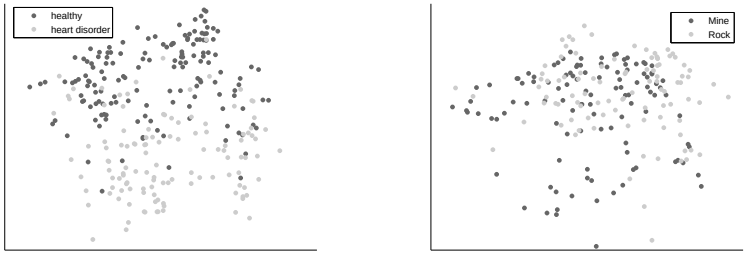


(a) (b)

Figure D.4: PCA for (a) Pima Indian Diabetes (48% of variance explained), (b) Spam (17% of variance explained)

D.1.5 German Credit

Reference	The German Credit data set has been collected by Hans Muller and published in 1994. It is available at the UCI Machine Learning Repository.
Number of samples	1000
Attributes	20
Labels	Good (70%) and Bad (30%)
Description	The German Credit data set describes 1000 people according to their credit-worthiness based on 20 attributes. These variables are either categorical or integer. Figure D.3(a) shows a PCA projection of this data set, which explains only 11% of the variance in this data set, an unusually low number. This is due to the low correlation between the variables.



(a) (b)

Figure D.5: PCA for (a) Heart Disease (32% of variance explained), (b) Sonar (39% of variance explained)

D.1.6 Bupa Liver Disorders

Reference	The Liver Disorders data set has been published by Richard Forsyth of the BUPA Medical Research Ltd.
Number of samples	345
Attributes	6
Labels	Healthy (58%) and Liver Disorder (42%)
Description	This data set contains the results from 5 blood tests for 345 male patients. The last attribute describes the amount of alcohol drunk per day. The predicted variable describes whether the patient suffers from liver disorder. In Figure D.3(b), a PCA plot of the Bupa Liver Disorders data is shown, revealing 57% of the variance.

D.1.7 Pima Indian Diabetes

Reference	The Pima data set has been collected by the National Institute of Diabetes and Digestive and Kidney Diseases. It has been published by Smith et al [40].
Number of samples	768
Attributes	8
Labels	Negative (65.1%) and Positive (34.9%)
Description	This data set describes female adult patients who are of Pima Indian heritage. The predicted variable is whether the patient is tested positive of diabetes. In Figure D.4(a) the PCA projection is given, explaining 48% of the variance.

D.1.8 Spam

Reference	The Spam data set has been created by Hopkins, Reeber, Forman, and Suermondt.
Number of samples	4601
Attributes	57
Labels	Spam (39.4%) and No Spam (60.6%)
Description	The samples in this data set correspond to email messages, which have manually been labeled as Spam or No Spam. The variables describe various features calculated for the emails, for example word frequencies, character frequencies, or percentage of capitals used. Figure D.4(b) shows the PCA projection. It explains 17% of the variance.

D.1.9 Heart Disease

Reference	The Heart Disease data set has been assembled in 4 clinics by Janosi, Steinbrunn, Pfisterer, and Detrano [81].
Number of samples	270
Attributes	13
Labels	Healthy (55.6%) and Heart Disease (44.4%)
Description	This data set contains relevant data of patients' medical records. The predicted variable tells whether the patient suffers from heart disease. As the original data set contains missing values, it has been reduced such that only full records are remaining. Figure D.5(a) shows a PCA plot that explains 32% of the variance.

D.1.10 Sonar

Reference	The Sonar data set has been first used by Gorman and Sejnowski [26].
Number of samples	208
Attributes	60
Labels	Mine (53.4%) Rock (46.6%)
Description	This data set consists of measurements of cylindrical objects from a sonar device. The attributes describe frequency-related information over different bands. The target variable is an indicator of whether the object is a metal cylinder, and thus potentially a mine, or a rock with roughly cylindrical shape. In Figure D.5(b), a PCA projection is shown, explaining 39% of the variance in this data set.

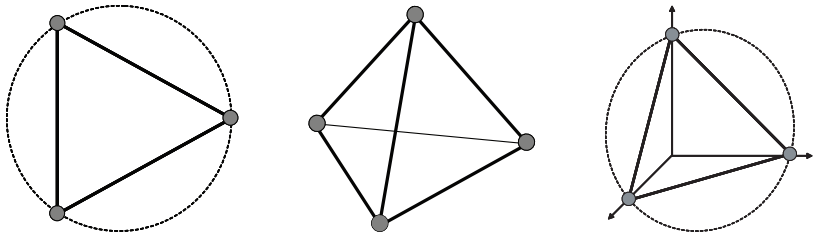
D.2 Artificial data sets

The artificial data sets are generated to show a particular feature of a specific data analysis method, or to empirically show that an algorithm is able to cope with a particular problem. For these data sets, there is no constant data sample, but the generating probability density function is given. The data sets are referred to by specifying the number of samples and any parameters required by the process of creating it.

D.2.1 Equidistant clusters data set

The Equidistant clusters data set consists of c clusters, the cluster centers of which have a common distance d from each other. The data set is $(c-1)$ dimensional. Examples are shown in Figure D.2.1, where the layouts for $c = 3$ and $c = 4$ are displayed. The gray vertices denote the cluster centers, and the lines symbolize the distances between the clusters, which is the same between any two cluster centers. In the two-dimensional case, the resulting shape is that a of an equilateral triangle, and in three dimensions, it is a tetrahedron.

The most simple way to construct the coordinate vectors of the graph is to initially use the unit vectors in a c dimensional space and project it to a $(c-1)$ dimensional space. The unit vectors in a Cartesian coordinate system are always equidistant. The set of unit vectors is the identity matrix when written in matrix form. When performing a linear projection along the vector $(1, 1, \ldots, 1)$ onto the hyperplane spanned by the unit vectors, the dimension of the space can be reduced by 1 to $(c-1)$. As the SOM and other projection methods only work with pairwise



(a) (b) (c)

Figure D.6: High dimensional graph structure of Equidistant clusters and Fully-connected data sets. For the Equidistant clusters data set, the data points are clustered around the vertices. In case of the Fully-connected data set, the samples are equally distributed along the edges: (a) 3 vertices in 2 dimensions, (b) 4 vertices in 3 dimensions, displayed as a linear projection, (c) construction of two-dimensional data set by projection of the unit vectors of the three-dimensional space

distances, and the data set is normalized by dividing by the standard deviation, the exact coordinates are not relevant for describing this data set, the only property that is of interest is that the centers are equally far apart.

Once the cluster centers have been computed, the actual data points are sampled from a multivariate normal distribution around each cluster center. The covariance matrix for each of these distributions is $\Sigma = \text{diag}(\sigma^2, \ldots, \sigma^2)$. As the data set is normalized before the SOM is trained on it, the absolute value of the variance is not important, only its relative size to d. Therefor, it is useful to introduce a spread ratio parameter $s = \frac{\sigma}{d}$ that expresses the standard deviation as a percentage of the inter-cluster distance. The Equidistant clusters data set is specified by the spread parameter s, the number of cluster centers c, and the number of samples per cluster.

Examples of the Equidistant clusters data sets are shown in Figure D.2.1 with $s = 20\%$ and 200 samples per cluster. The PCA projection fails to reveal the individual clusters at higher dimensional problems, although the clusters are well separated. The Equidistant clusters data set serves as a benchmark to assess the vector quantization capabilities of a projection algorithm.

D.2.2 Fully-connected data set

The Fully-connected data set is also created from the high-dimensional graph introduced in the previous section. The difference lies in the data points are dis-

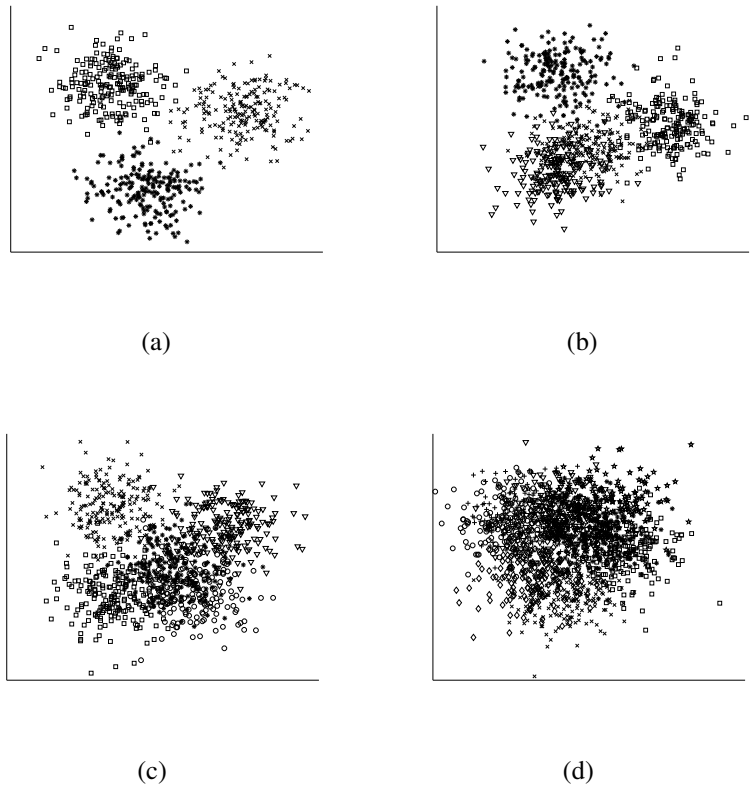


Figure D.7: PCA projections of Equidistant clusters data sets with 200 data points per cluster and spread ratio $s = 20\%$: (a) 3 vertices, (b) 4 vertices, (c) 5 vertices, (d) 8 vertices

tributed along the edges rather than clustered around the vertices. This data set is similar to a fully connected graph where the vertices are equidistant. The resulting data set can not be represented by a SOM without major topology violations for 5 or more vertices, as the edges of the graph would intersect if projected onto a two dimensional plane.

PCA projections of Fully-connected data sets are shown in Figure D.2.2. For this data set, PCA is very good at revealing the graph structure of the data, because the intersections of the edges do not distort the projection. Other than the SOM, the figures do not reveal that nearby points in ouptut space can be actually far apart, as is the case with these intersections. The rationale for using the Fully-connected data set is to test whether a projection algorithm is able to place points that are close in feature space next to each other in output space. It can be used as a benchmark for a certain aspect of a technique's vector projection capabilities.

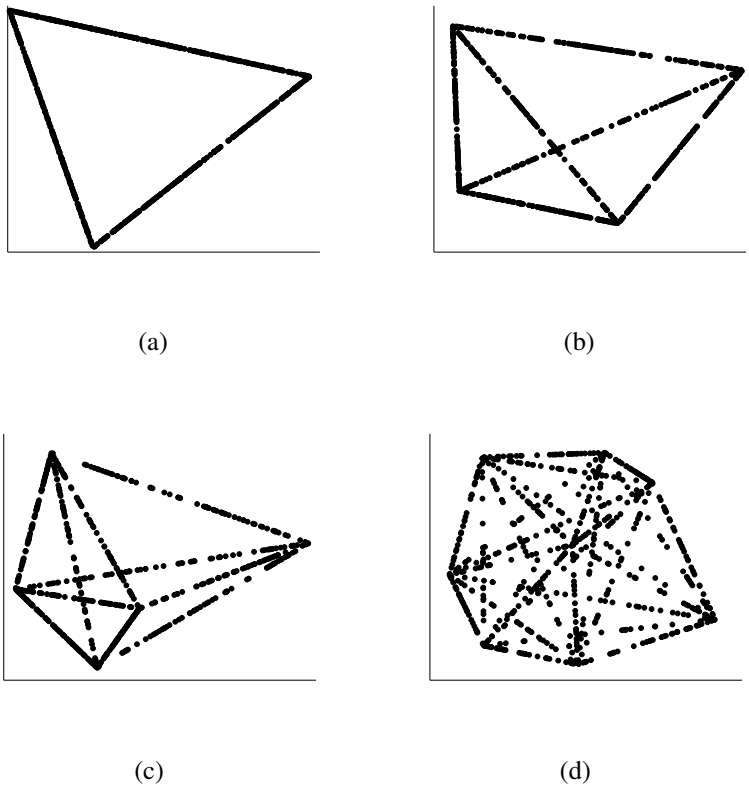


Figure D.8: PCA projections of Fully-connected data sets with 700 data points: (a) 3 vertices, (b) 4 vertices, (c) 5 vertices, (d) 8 vertices

D.2.3 Multi-challenge data set

The Multi-challenge data set consists of several sub-data sets that are placed in a 10-dimensional space. The subsets themselves may live in spaces of lower dimensions. This data set is used to demonstrate how a data analysis method deals with clusters of different densities and shapes when these different characteristics are present in the same data set.

A PCA projection of this data set is shown in Figure D.2.3. Each subset consists of the same number of sample points. The subsets are described in the order of their numeric label:

The first subset consists of a Gaussian cluster and another cluster that is itself divided into three Gaussian clusters, all of them living in a three-dimensional space. This subset is used to demonstrate how an algorithm deals with different levels of granularity in cluster structures. The distance between the centers of the

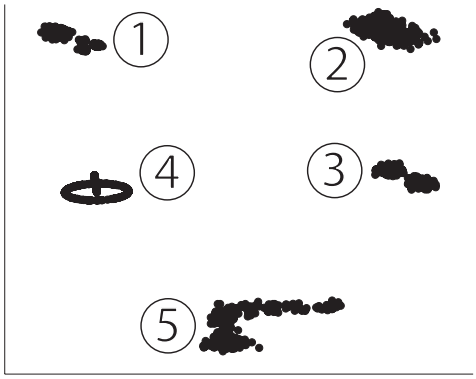


Figure D.9: PCA of Multi-challenge data set

two main clusters, i.e. the the big cluster and the cluster that consists of the three smaller ones, is 5 times the standard deviation d of the first main cluster. The three smaller clusters are arranged around the center of the second cluster, which they themselves form, on a circle of radius $\frac{5 \cdot d}{3}$ equidistantly from each other. The three small cluster centers and the center of the large cluster lie in the same plane. The small clusters each have a standard deviation of $\frac{d}{3}$ and one third of the number of data points of the large cluster.

The second subset is a three-dimensional data set that consists of two overlapping Gaussian clusters. The distribution of points into these clusters is skewed: While the clusters both share the same covariance matrix and thus the same density, the first cluster has twice the amount of points than the second one. The point of introducing this data set is to test how a data analysis method copes with clusters with different numbers of data samples. The distance between the centers is 3 times the standard deviation of the clusters.

The third subset is a 10-dimensional set of two well-separated Gaussian clusters with the same covariance matrix. It is used to contrast higher- with lower-dimensional structures, as the other four subsets are of lower dimensionality. The distance between the clusters is again 3 times their standard deviation.

The fourth subset is a classic intertwined rings data set. It lives in a three-dimensional data space. This data set is used for showing how a method deals with non-intersecting structures that cannot be separated in a linear way. The rings are circles that run through each others centers. The planes on which they are located are orthogonal to each other. The data points are sampled from these rings with no additional noise.

The fifth subset is sampled along a curve that consists of 4 short lines that are

patched together at the endpoints. The data points are arranged along this curve with a level of noise that increases close to the end of the curve. This subset lives in a four-dimensional space. It is introduced in order to show how data analysis methods cope with piecewise linear structures that extend to multiple dimensions. In detail, the four parts of this subset are line segments each parallel to one of the axis. The data points are sampled from this structure with a small amount of Gaussian noise with standard deviation of $\frac{1}{20}$ of the length of the line segments.

The subsets are normalized individually to zero mean and unit variance. They are then arranged on a plane, which can be seen in Figure D.2.3. The distance between the data sets is 10 times their standard deviation.