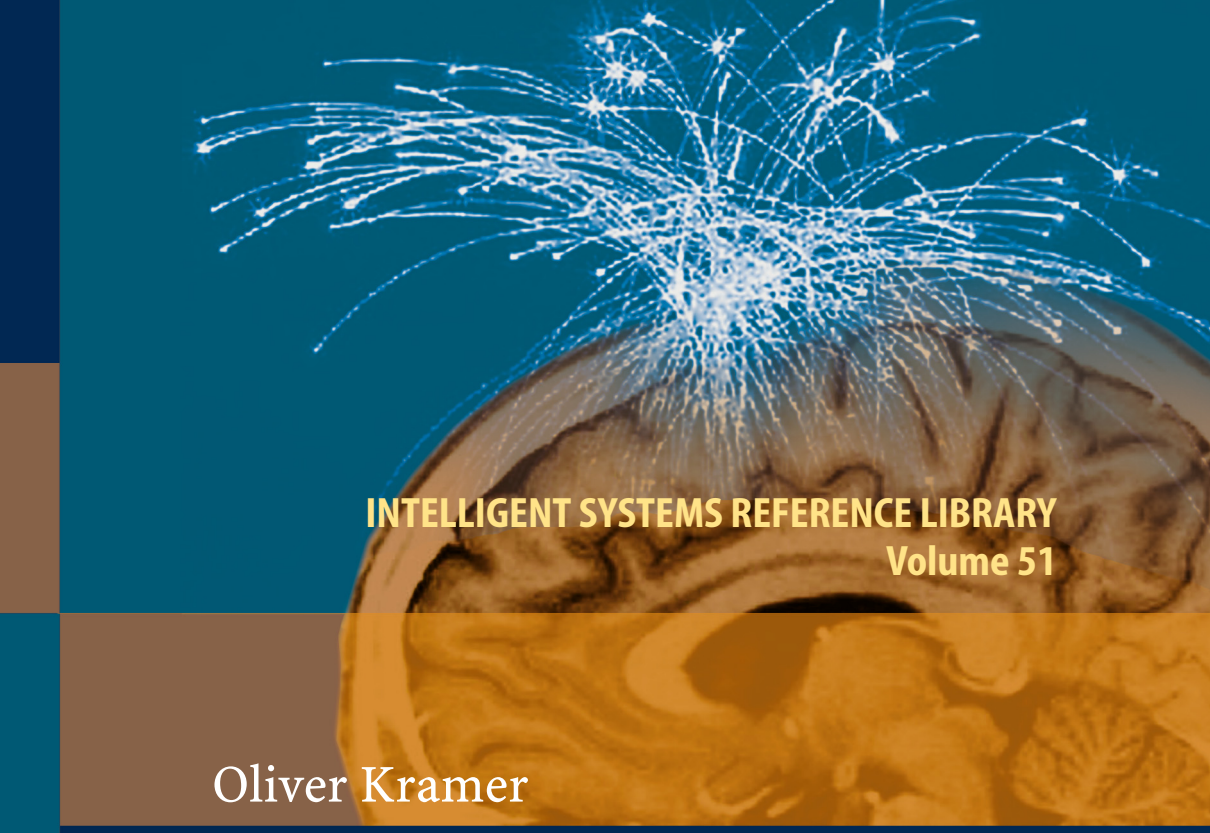


INTELLIGENT SYSTEMS REFERENCE LIBRARY
Volume 51

Oliver Kramer

Dimensionality Reduction with Unsupervised Nearest Neighbors

Springer

Intelligent Systems Reference Library

Volume 51

Series Editors

J. Kacprzyk, Warsaw, Poland
L. C. Jain, Adelaide, Australia

For further volumes:
http://www.springer.com/series/8578

Oliver Kramer

Dimensionality Reduction with Unsupervised Nearest Neighbors

Springer

Oliver Kramer
Computational Intelligence Group
Computer Science Department
Carl von Ossietzky University Oldenburg
26111 Oldenburg
Germany

ISSN 1868-4394 ISSN 1868-4408 (electronic)
ISBN 978-3-642-38651-0 ISBN 978-3-642-38652-7 (eBook)
DOI 10.1007/978-3-642-38652-7
Springer Heidelberg New York Dordrecht London

Library of Congress Control Number: 2013939181

Abstract

The growing information infrastructure in a variety of disciplines involves an increasing requirement for efficient data mining techniques. Fast dimensionality reduction methods are important for understanding and processing of large data sets of high-dimensional patterns. In this work, unsupervised nearest neighbors (UNN), an efficient iterative method for dimensionality reduction, is presented. Starting with an introduction to machine learning and dimensionality reduction, the framework for unsupervised regression is introduced, which is the basis of UNN. Algorithmic variants are developed step by step, reaching from a simple iterative strategy in discrete latent spaces to stochastic kernel-based submanifolds with independent parameterizations. Experimental comparisons to related methodologies taking into account real-world data sets and missing data scenarios show the behavior of UNN in practical scenarios.

Acknowledgements

First, I like to thank Michael Sonnenschein and Hans-Jürgen Appelrath at the Department of Computing Science in Oldenburg for their great support. I like to thank the German Research Foundation (Deutsche Forschungsgemeinschaft) and the Präsidium of the Carl von Ossietzky Universität Oldenburg for the financial support while writing this work. Further, I like to thank all graduate students, postdocs, professors, and mentors I have been working with since the beginning of my research activities. I had many interesting discussions and research collaborations with Fabian Gieseke, Sascha Hunold, Hans Kleine Büning, Richard Karp, Jörg Lässig, Günter Rudolph, Benjamin Satzger, Hans-Paul Schwefel, Benno Stein, Dirk Sudholt, and Timon Rabczuk. For the NIALM data used in Chapter 3, I thank Olaf Wilken and the OFFIS in Oldenburg. For their support in astronomy, I thank the astronomy group of Jürgen Dettmar in Bochum.

Oliver Kramer
Oldenburg, April 2013

Contents

List of Figures

List of Tables

1

Introduction

1.1 Artificial Intelligence

Artificial intelligence (AI) belongs to the most interesting and dynamic areas in computer science. With a focus on the imitation and creation of intelligent behavior by machines and robots, AI is an area with great achievements, but also disappointments of early expectations. The discipline AI can be divided into a number of subfields like reasoning, pattern recognition, planning and learning. Many classic AI algorithms are based on advanced search techniques, as in many cases planing and reasoning corresponds to searching in a solution space. Examples reach from simple search strategies in graphs like breadth-first and depth-first search to advanced reinforcement strategies for learning of complex behaviors in uncertain environments. Many AI research objectives aim at the solution of special problem classes. Subareas like speech processing have shown impressive achievements in recent years that come close to human abilities.

The research on AI is strongly related to philosophical questions. The goal of *weak* AI is to program machines that imitate human behavior. The goal of *strong* AI is to program machines that do not only imitate humans, but think like humans. In spite of the progress in neuroscience, in practice, we are far away from this goal, and the question arises, if it will ever be achieved. Many AI researchers think that an intelligent machine can be constructed and human thinking can be achieved, if the processes of neural information processing are emulated.

The area of computational intelligence (CI) is related to AI and comprises nature-inspired algorithms. The motivation for the imitation of methods from nature is that many interesting problem solving strategies can be observed in biology like evolution, and natural neural networks. Many complex problems can hardly be treated mathematically. The strict mathematical modeling of real-world problems often induces very large solution spaces. Classical mathematical models have problems with noise and uncertainty. Nature has evolved systems able to cope with these challenges. CI methods

O. Kramer: *Dimensionality Reduction with Unsupervised Nearest Neighb.*, ISRL 51, pp. 1–9.
DOI: 10.1007/978-3-642-38652-7_1

are inspired by natural information processing and designed to solve such problems. The classical fields are evolutionary computing, artificial neural networks and fuzzy systems. Evolutionary methods are inspired by the natural process of evolution, i.e., recombination, mutation, and selection. Evolutionary processes can be applied as optimization methods, i.e., to solve numerical or combinatorial optimization problems. Even the evolvement of programs called genetic programming is possible. Artificial neural networks are inspired by information processing of natural neural networks like the human brain. Biological neurons are modeled as artificial neurons with excitations, firing, and thresholds. Many artificial neural networks are applied to data mining task. The approach to imitate natural neural networks close to the biological counterpart is still an ambitious research task. The field of neural networks is strongly related to data mining and machine learning, e.g., via the famous support vector machines, whose concept is related to Rosenblatt's perceptron. Research in neural networks got stuck due to the failure of single layer perceptrons in learning the simple XOR-problem. The third branch of CI is fuzzy logic that offers a formalism for representing real-world knowledge with uncertainty. It allows to cope with uncertain linguistic terms and expressions. Fuzzy logic has become popular, because it turned out to be an easy to apply inference method and powerful controller. Fuzzy controllers are successful and interpretable methodologies in many engineering applications. Emerging fields of CI are swarm intelligence inspired by natural swarms like ant colonies and flocks of birds, and artificial immune systems.

1.2 Machine Learning

Many disciplines in our modern information society are based on collecting high-dimensional patterns: from astronomy to psychology, from civil engineering to social web services. Digitizing the world often means that high-dimensional patterns are generated. The collection and understanding of data allows us to improve the efficiency of processes in a variety of domains. Algorithms are required that are able to process the data sets efficiently. There are numerous examples that reflect the importance of the understanding of large data sets. The sensor quality is steadily improving. This requires the development of dimensionality reduction methods that allow an efficient data analysis process.

Astronomy is a good example for the growing demand for efficient dimensionality reduction methods. Observations in astronomy helped to understand processes in our universe and have been a fruitful source for new theories in physics. Recent developments affect the improvement of the quality of telescopes. Advanced techniques in imaging and an enormous increase of the number of telescopes leads to an explosion of available data, both in terms of dimensionality of patterns and of their number. Modern telescopes produce

tera-, some even petabyte of data each night. This development increases the demand for fast and efficient data mining techniques that allow the physicists to understand and automatically interpret the data.

There are many task that can be accomplished with collected data. Sorting patterns, visualizing, recognizing structures, learning functional models for prediction of labels for unknown observations – the list of problem classes and of methods seems to be overwhelming at first. But when having a closer look, many concepts and methods are similar to each other. For beginners, it is not easy to get a complete overview, as methodologies and modeling approaches sometimes come from very different disciplines: analysis, linear algebra, stochastics, combinatorics, optimization, cognitive science, and artificial intelligence. For a thorough understanding of data mining processes, a deeper knowledge of the application domain is required.

1.3 Supervised Learning

The objective of supervised machine learning methods is to learn a functional model f from observations to derive relationships, e.g., between patterns and labels. The two most important machine learning problem classes are supervised and unsupervised learning. Supervised methods are based on observed features and corresponding labels. Prediction means that observations of patterns and corresponding labels are used to determine labels of unknown patterns. If the labels are discrete, the learning problem is called classification problem, because patterns are assigned to classes (a discrete label is called class label). If labels are continuous, the task is a regression problem. An important questions in machine learning is how a pattern is defined. Basically, a pattern consists of features that have been recorded and collected, e.g., from sensors, cameras, microphones, but also non-physical data sources like questionnaires or web surfing behavior. But in applications, it often makes sense to preprocess the data, e.g., to reduce noise. In Chapter 5, I will employ techniques that allow learning in the presence of noise.

If patterns $\mathbf{x}_i \in \mathbb{R}^q$ with $i = 1, \ldots, N$ are observed with labels y_i, the set of pattern-label-pairs $(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)$ can be used to train a functional model f. The label can be a discrete class label or a continuous value $y_i \in \mathbb{R}$. As the true distribution of patterns and labels is typically not known, the search for f can be performed by minimizing the empirical risk (cf. Chapter 2) based on the available observations with loss function $L(\cdot)$ measuring the deviations between predictions $f(\mathbf{x}_i)$ and labels y_i. This problem is also known as model selection. The optimal result is guaranteed, if the search takes place in the set of all functions $\mathcal{F}$. In practice, it is not reasonable to search in the whole set $\mathcal{F}$. Instead, it is reasonable to choose a certain method corresponding to a function subset $F \subset \mathcal{F}$ and to optimize its free parameters w.r.t. the empirical risk resulting in model f. If the function space F is large, overfitting may occur, and it is a reasonable approach to restrict it

by penalizing the complexity of model f with a regularizer, which is often a functional norm $\|f\|$. Then, the objective becomes to minimize the regularized risk balancing between empirical risk minimization and smoothness of the function.

A comparatively simple classification method is the K-nearest neighbor (KNN) classifier [28]. For an unknown pattern $\mathbf{x}_j$, it assigns the class label of the K-closest patterns in data space. For this sake, a distance measure has to be defined in the space of patterns. In $\mathbb{R}^q$, it is reasonable to employ the Minkowski metric (p-norm). In other solution spaces, adequate distance functions have to be chosen, e.g., the Hamming distance in $\mathbb{B}^N$. The choice of K defines how *local* KNN is. Further prominent classification methods are decision tress like ID3 [81], backpropagation networks [93, 96], and support vector machines (SVMs) [98, 101]. Figure 1.1 shows the learning result of a two-class SVM with radial basis function (RBF) kernel on (a) the OR-problem and (b) the XOR-problem that is not linearly separable. The XOR-problem led to a stagnation in neural network research in the nineties, as the one-layer perceptron fails to learn non-separable data sets. For a detailed introduction to machine learning, I refer the interested reader to books like Bishop [12] and Hastie *et al.* [40].

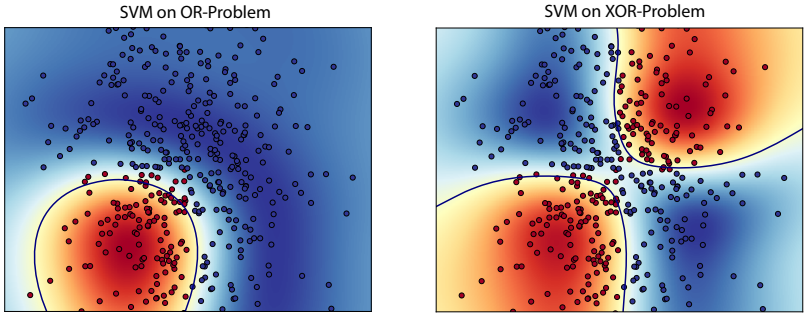


Fig. 1.1 An SVM classifier with RBF kernel on (a) the OR and (b) the XOR-problem, which is not linearly separable

The question arises how to choose the parameters of supervised learning methods, e.g., kernel parameters of an SVM and neighborhood sizes of KNN. Various techniques like cross-validation can be used to choose the best model. Cross-validation divides the set of observed patterns into training and validation sets and successively computes the error w.r.t. different settings to avoid overfitting. Outlier or novelty detection is a special variant of supervised learning. The task is to learn an estimator of patterns with given labels and to let this classifier determine, if novel patterns belong to the same distribution or if they can be classified as *outliers*.

1.4 Unsupervised Learning

In the unsupervised learning scenario, no labels are given. The learning task is to capture as much information as possible from the intrinsic structure of the data in the learning result. Clustering is a famous example for learning without a *teacher* or label, respectively. Patterns that lie closely together in data space form groups also known as clusters. The task of clustering algorithms is to detect these groups autonomously, only based on the intrinsic structure of the data. A famous and efficient clustering algorithm is K-means. It is based on distributing codebook vectors in data space such that they lie in the centers of potential clusters. The algorithm works iteratively in two steps. First, patterns are assigned to their closest codebook vector. Second, new codebook vectors become the novel cluster centers.

The *intrinsic* dimensionality of data is often lower than the actual dimensionality. Substantial information may only be present in a subset of features. In this case, the selection and the computation of high-level features is reasonable. If the importance of particular features is known in advance, one can significantly reduce the data space size by feature selection. Otherwise, dimensionality reduction methods can be applied. Their task is to compute low-dimensional features maintaining important properties of their high-dimensional counterparts. Preservation of topological properties is a main objective in dimensionality reduction, i.e., to maintain neighborhoods and distances of data space in the low-dimensional latent space. Chapter 4 will introduce dimensionality reduction in detail. There are many application domains for dimensionality reduction problems in practice. Two examples will be presented in the following.

1.5 Monitoring of Wind Time Series

In the following, I give two examples for dimensionality reduction from our recent projects. Sustainability is an important objective due to increasing energy demands and limited resources. In the field of power production and consumption, methods are required that improve energy efficiency. Prediction belongs to the most important task. The better power consumption and production can be predicted, the more efficient energy systems are. To guarantee electricity supply, overcapacities are necessary. But this reserve energy can be reduced to a minimum, if a precise forecast of demand and production is available.

The extension of renewable energy resources and the growing information infrastructure allow monitoring of energy resources with high resolutions. As wind is a volatile resource, state observation has an important part to play for grid management, fault analysis and planning strategies of grid operators. High-dimensional time series are usually difficult to understand and to visualize. The existing infrastructure of wind turbines can be used to learn

statistical models and for monitoring of energy production. For visualization of wind time series, we employed self-organizing maps (SOMs) that will be introduced in Chapter 4. A SOM-based mapping to symbols allows the application of pattern recognition techniques, e.g., dynamic time warping [77, 111] for the recognition of critical time series and alert states.

To give an example, I visualize high-dimensional wind energy time series of a wind park in Tehachapi, California, with SOMs of different sizes [68]. I employ a data set from the US National Renewable Energy Laboratory (NREL) [89] consisting of 20 grid points (a grid point usually comprises ten wind turbines). These are randomly chosen and are aggregated to 20-dimensional patterns. For a better readability, I concentrate on the visualization of a period of seven days, corresponding to approximately 1,000 measurements, one every 10 minutes. The two sequences of Figure 1.2 show the time series visualization of 20-dimensional grid points of Tehachapi. The following steps have been conducted:

- The time series of $1,000$ measurements of 20 grid points have been allocated to $N = 1000$ patterns that are 20-dimensional.
- Two SOMs (2×2 / 10×10) have been trained for 100 / 500 training cycles.
- In the original order, the 1,000 patterns of a test sequence are presented to the trained SOM, and the colors of the corresponding winner neurons are used for sequence visualization along the time axis.

The upper part of Figure 1.2 shows the visualization based on the 2×2-SOM with 100 learning cycles to the output of a 10×10-SOM with 500 learning cycles. Both SOM-training results lead to structurally similar topological mappings. The color assignments may vary gradually in different runs, as the training is a non-deterministic process.

It can be observed that both sequence visualizations share similar color changes, while the larger SOM allows a finer resolution of time sequence states. The green Part A left to the middle of both sequences is a period, when the wind is almost not blowing. The corresponding segments of both sequences employ similar colors. In contrast, Part B shows a segment with the same color generated by the 2×2 SOM, but varying colors in case of the 10×10-SOM. The latter is able to visualize differences with a higher resolution in the same period. For interpretation purposes, the SOM allows a backward mapping by showing the neural weights corresponding to the multi-dimensional patterns that represent the system states.

1.6 Gesture Recognition

The understanding of human motions is an important research field, e.g., in healthcare and robotics. A special task is the recognition of manual actions like hand movements, postures, and gestures. Human motions can be understood as multi-dimensional time series data. I show a problem from gesture

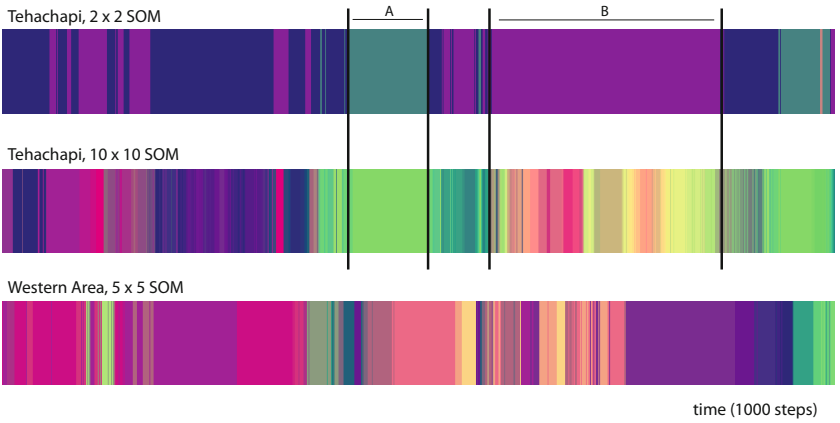


Fig. 1.2 Upper two sequences: Visualization of time series of 20 wind spots in the area of Tehachapi. The sequences are 20-dimensional time series of 1,000 steps, starting from January 2006. The results are generated with a 2×2-SOM with 100 learning cycles and a 10×10-SOM with 500 learning cycles. Lower part: Visualization of time series of 20 randomly chosen wind spots in the whole area of the western wind data set, i.e., Nampa, Salt Lake City, Casper, Fort Collins, Colorado Springs, Amarillo, Carlsbad in Mexico, Las Vegas, Palm Springs, and Tehachapi using a 5×5-SOM.

recognition as second example for dimensionality reduction. Manual intelligence is an important task in human-computer interaction and robotics, see Ritter *et al.* [53]. Hands are the most important manipulators in a human's interaction with the environment. Data can be captured from multiple sources, e.g., acceleration sensors, cameras, hand gloves or visual marker systems.

In our experiment [77], we were interested in analyzing everyday tasks like pouring a cup of milk and writing into a book. To record the manual actions, we employed a Vicon system capturing 3D spatial data. Vicon is a digital optical motion capture system that allows high-precision 3D object tracking [1]. Our setup consists of a cage of length 2.1 m, width 1.3 m, and height 2.1 m that employs 14 MX3+ cameras capturing at 200 frames per second. Reflective markers were placed near the tips of each finger, on each knuckle, and on the back of the hand (cf. Figure 1.3). After translation of the coordinates into time series, reasonable features have been computed. The question arises, what the best features are to describe manual motions. We concentrated on features like *inclination of the hand*, i.e., the angle between the x-y-plane and the axis going through the index finger knuckle and baby finger knuckle markers. Other features are magnitudes of velocities, angles between successive velocity vectors, or the average distance of the five finger tip markers to the barycenter.

After recording of the features and the feature extraction process, dimensionality reduction methods have been employed that allow to assign the

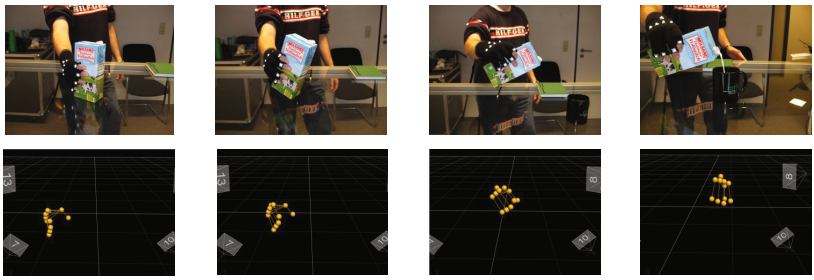


Fig. 1.3 Example of manual action scenario: a carton of milk is grasped, picked up, and milk is poured into a cup. The lower figures show the corresponding visualization of the markers in Nexus [1], a software program from Vicon. The high-dimensional gesture patterns are mapped to symbols via dimensionality reduction, and gestures are recognized via string matching.

high-dimensional patterns to symbols. A successful approach was the employment of a SOM, cf. Section 4.4. In a trained SOM, each neuron can be understood as symbol such that gestures become sequences of symbols. Further methods have been integrated into the system for comparison. For the string matching process, we used dynamic time warping [111], which allows to cope with insertions and deletions of symbols. In a training phase, a set of sequences of reference motions have been recorded. In the recognition phase, movements that are similar to the motions in the training set are performed consecutively and in arbitrary order. The string matching approach was able to identify most of the training motions. Unknown movements that have not been shown in the training phase have successfully been assigned to the most similar training set motions.

The adaptation of dimensionality reduction algorithms to subsequent processing with symbolic algorithms is an important challenge. An interesting example is the design of cognitive systems that collect high-dimensional sensorimotor data. But the high-dimensional perception has to be translated into discrete symbols for processing of symbolic algorithms. From this perspective, the mapping from high-dimensional data to symbols is called symbol grounding. Symbol grounding is about whether these systems can, based on this data, construct symbols that serve as a vehicle for higher symbol-oriented cognitive processes. The challenge is to design dimensionality reduction algorithms that are geared towards grounding symbols in an unsupervised way, only with a feedback on the level of higher objectives. A step into this direction is my work [56], where I proposed a target-oriented optimization procedure as solution to the symbol grounding problem. It is demonstrated that the machine learning perspective is consistent with the philosophical perspective of constructivism. Dimensionality reduction is the most important part

in the symbol grounding process. The integration of dimensionality reduction to cognitive systems will be one of the most exciting directions of our future work.

1.7 Overview and Preliminary Publications

This book is devoted to a novel approach to dimensionality reduction based on the famous nearest neighbor method. Parts of this work are based on preliminary *peer-reviewed* publications in proceedings of international conferences and in international journals. Nearest neighbors is a simple, yet efficient method for classification, regression, and time series prediction. The unsupervised formulation for dimensionality reduction problems will be introduced step by step, presenting numerous variants of the novel technique, from fast constructive heuristics [55, 58] to stochastic approaches [57, 59, 60, 61]. Unsupervised nearest neighbors (UNN) is based on fitting K-nearest neighbor regression to the unsupervised regression framework for learning of low-dimensional manifolds. Similar to related approaches that are mostly based on kernel methods, UNN optimizes latent variables w.r.t. the data space reconstruction error. The problem of optimizing latent neighborhoods is difficult to solve, but the UNN formulation allows an efficient strategy of iteratively embedding latent points into discrete and continuous latent spaces.

The book will give deeper insights into aspects that are relevant from a theoretical perspective like runtime and kernelization, but also practical aspects like loss functions for handling noise and methods to handle incomplete data. It provides a consistent view on my past research activities and highlights important aspects of the line of research on unsupervised nearest neighbors. The remainder of this book will be written in a scientific style with the use of “we” rather than “I”.

Part I

Foundations

2

K-Nearest Neighbors

2.1 Introduction

This chapter gives an introduction to pattern recognition and machine learning via K-nearest neighbors. Nearest neighbor methods will have an important part to play in this book. The chapter starts with an introduction to foundations in machine learning and decision theory with a focus on classification and regression. For the model selection problem, basic methods like cross-validation are introduced. Nearest neighbor methods are based on the labels of the K-nearest patterns in data space. As local methods, nearest neighbor techniques are known to be strong in case of large data sets and low dimensions. Variants for multi-label classification, regression, and semi-supervised learning settings allow the application to a broad spectrum of machine learning problems. Decision theory gives valuable insights into the characteristics of nearest neighbor learning results.

2.2 Classification

Classification is the problem to predict discrete class labels for unlabeled patterns based on observations. Let $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ be the set of observations of q-dimensional patterns $\mathcal{X} = \{\mathbf{x}\}_{i=1}^{N} \subset \mathbb{R}^q$ and a corresponding set of labels $\mathcal{Y} = \{\mathbf{y}_i\}_{i=1}^{N} \subset \mathbb{R}^d$. The goal in classification is to learn a functional model f that allows a reasonable prediction of class label y' for an unknown pattern $\mathbf{x}'$. Patterns without labels should be assigned to labels of patterns with known assignment that are similar, e.g., that are close to the target pattern, come from the same distribution, or lie on the same side of a separating function. But learning from observed patterns can be difficult. Training sets can be noisy, important features may be unknown, similarities between patterns may not be easy to define, and observations may not be sufficiently described by simple distributions. Further, learning simple functional models can be a difficult undertaking, as classes may not be linearly

O. Kramer: *Dimensionality Reduction with Unsupervised Nearest Neighb.,* ISRL 51, pp. 13–23.
DOI: 10.1007/978-3-642-38652-7_2

separable and may be difficult to separate with simple rules or mathematical equations.

Famous classification methods are decision tress, e.g., ID3 and C4.5 [81], neural networks [93, 96], and SVMs [98, 101]. For an introduction to these methods, we refer the reader to books like Bishop [12] and Hastie *et al.* [40]. SVMs will also briefly be introduced in Chapter 3. In the following, we concentrate on a simple yet powerful method: the nearest neighbor classifier.

2.2.1 KNN Classifier

Nearest neighbor classification, also known as K-nearest neighbors (KNN), is based on the idea that the nearest patterns to a target pattern $\mathbf{x}'$, for which we seek the label, deliver useful label information. KNN assigns the class label of the majority of the K-nearest patterns in data space. For this sake, we have to be able to define a similarity measure in data space. In $\mathbb{R}^q$, it is reasonable to employ the Minkowski metric (p-norm)

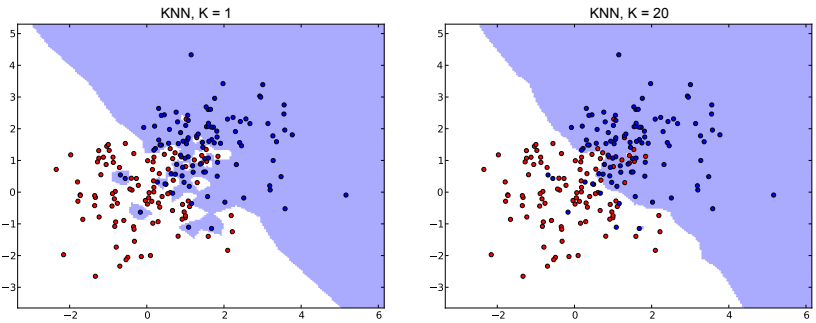
$$\|\mathbf{x}' - \mathbf{x}_j\|^p = \left(\sum_{i=1}^{q} |(x_i)' - (x_i)_j|^p \right)^{1/p}, \tag{2.1}$$

which corresponds to the Euclidean distance for $p = 2$. In other data spaces, adequate distance functions have to be chosen, e.g., the Hamming distance in $\mathbb{B}^q$. In the case of binary classification, the label set $\mathcal{Y} = \{1, -1\}$ is employed, and KNN is defined as

$$f_{\mathrm{KNN}}(\mathbf{x}') = \begin{cases} 1 & \text{if } \sum_{i \in \mathcal{N}_K(\mathbf{x}')} y_i \geq 0 \\ -1 & \text{if } \sum_{i \in \mathcal{N}_K(\mathbf{x}')} y_i < 0 \end{cases} \tag{2.2}$$

with neighborhood size K and with the set of indices $\mathcal{N}_K(\mathbf{x}')$ of the K-nearest patterns.

The choice of K defines the *locality* of KNN. For $K = 1$, little neighborhoods arise in regions, where patterns from different classes are scattered. For larger neighborhood sizes, e.g. $K = 20$, patterns with labels in the minority are ignored. Figure 2.1 illustrates the difference in classification between KNN with $K = 1$ and $K = 20$ on a simple 2-dimensional data set consisting of two overlapping data clouds of each 50 Gaussian-sampled red and blue points. The data space locations that would be classified as *blue* are shown in bright blue, while areas classified as *red* are shown in white. For $K = 1$, the prediction is *local*. For example, a blue point that is an outlier from the blue class is located at the center of the red cloud. For large K, the classifier generalizes ignoring small agglomerations of patterns. KNN induces a Voronoi tessellation in data space. In case of large data sets, KNN has to search for the K-nearest patterns in the whole space, but can already yield a good approximation based on the K-nearest neighbors in a scanned subset.



(a) (b)

Fig. 2.1 A comparison of KNN classification on two Gaussian-based data clouds for two neighborhood sizes ((a) $K = 1$ and (b) $K = 20$). For small neighborhoods, KNN tends to overfit becoming *local*, while KNN generalizes for larger K ignoring small agglomerations of patterns.

The question arises, how to choose K, i.e., which neighborhood size achieves the best classification result. This problem is also known as model selection, and various techniques like cross-validation can be employed to choose the best model and parameters (cf. Section 2.5).

2.2.2 Multi-class K-Nearest Neighbors

KNN can also be applied to multi-class classification problems. For an unknown pattern $\mathbf{x}'$, KNN for multi-class classification predicts the class label of the majority of the K-nearest patterns in data space

$$f_{\text{KNN}}(\mathbf{x}') = \arg\max_{y \in \mathcal{Y}} \sum_{i \in \mathcal{N}_K(\mathbf{x}')} \mathcal{I}(y_i = y) \tag{2.3}$$

with indicator function $\mathcal{I}(\cdot)$ that returns *one*, if its argument is true[1] and *zero* otherwise. This definition will also be used for the ensemble classifier in Section 3.4.

2.3 KNN Regression

Closely related to classification is regression. Functional regression models map patterns to continuous labels, i.e., to a subspace of $\mathbb{R}^d$. Although in practice, machine accuracy only allows a mapping to a discrete set of numbers, the difference becomes obvious: the set of labels is very large in comparison to classification problems, where the set of labels is restricted to

[1] i.e., label y_i of pattern $\mathbf{x}_i$ is y.

few elements. The problem in regression is to predict labels $\mathbf{y}' \in \mathbb{R}^d$ for new patterns $\mathbf{x}' \in \mathbb{R}^q$ based on a set of N observations, i.e., labeled patterns $\{(\mathbf{x}_1, \mathbf{y}_1), \ldots, (\mathbf{x}_N, \mathbf{y}_N)\}$. The regression learning problem will be derived from a statistical learning perspective in Section 2.4.

The goal is to learn a function $\mathbf{f} : \mathbb{R}^q \to \mathbb{R}^d$ known as regression function. For an unknown pattern $\mathbf{x}'$, KNN regression computes the mean of the function values of its K-nearest neighbors

$$\mathbf{f}_{KNN}(\mathbf{x}') = \frac{1}{K} \sum_{i \in \mathcal{N}_K(\mathbf{x}')} \mathbf{y}_i \tag{2.4}$$

with set $\mathcal{N}_K(\mathbf{x}')$ containing the indices of the K-nearest neighbors of $\mathbf{x}'$. The idea of averaging in KNN is based on the assumption of locality of functions in data and label space. In local neighborhoods of $\mathbf{x}_i$, patterns $\mathbf{x}'$ are expected to have similar continuous labels $\mathbf{f}(\mathbf{x}')$ like $\mathbf{y}_i$. Hence, for an unknown $\mathbf{x}'$ the label must be similar to the labels of the closest patterns, which is modeled by the average of the label of the K-nearest patterns. KNN has been proven well in various applications, e.g., in the detection of quasars based on spectroscopic data [33].

Also in regression, the neighborhood size K of KNN is an important parameter. For $K = 1$, KNN regression *overfits* to the label of the nearest neighbor of $\mathbf{x}'$, for $K = N$ it averages over all patterns Figure 2.2 shows a comparison between KNN regression with the two neighborhood sizes (a) $K = 2$ and (b) $K = 5$. Weighted KNN regression induces plateaus.

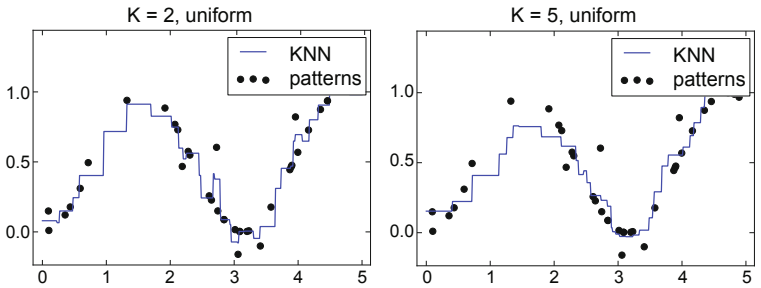


Fig. 2.2 Illustration of uniform KNN regression for (a) $K = 2$ and (b) $K = 5$

2.4 Statistical Learning Theory

From statistical learning theory, we can get insights into the quality a functional model should have. We assume that observed patterns and labels can be modeled with random variables. This allows to model noise and fluctuations. In the following, we consider the univariate case. We assume two continuous

random variables: X for patterns and Y for labels, which can be described by probability distribution functions, i.e,. $p(X = x)$ and $p(Y = y)$, as well as the joint distribution $p(X = x, Y = y)$. The task in supervised learning is to learn a functional model f, which predicts the correct label y for a given pattern x. A loss function $L(y, f(x))$ allows to measure the error, which is the deviation of the correct label and the prediction of the functional model. The task is to learn a model with minimal error. Often, the square loss is employed

$$L(y, f(x)) = (y - f(x))^2. \tag{2.5}$$

The probability distribution functions allow the evaluations of the quality of the functional model f with the expected prediction error

$$E_{exp}(f) = \langle L(Y, f(X)) \rangle = \int \int L(y, f(x)) p(x, y) dx dy, \tag{2.6}$$

while $\langle \cdot \rangle$ is defined as expected value of a random variable. Now, we can replace the joint probability density function by the Bayes expression

$$p(x, y) = p(y|x)p(x) = p(Y = y|X = x)p(X = x), \tag{2.7}$$

becoming

$$E_{exp}(f) = \int \left(\int (y - f(x))^2 p(y|x) dy \right) p(x) dx \tag{2.8}$$

with the square loss. We seek for the best functional model f^* that minimizes the *empirical risk* $E_{exp}(f)$. Minimization of E_{exp} can be accomplished by point-wise minimization of the inner integral for each x yielding the regression function

$$\hat{f}^*(x) = \int y p(y|x) dy = \int y \frac{p(x, y)}{p(x)} dy = \langle y|x \rangle. \tag{2.9}$$

The consistency criterium postulates that the empirical risk converges to *zero* in the limit of an infinite number of training examples. The problem to find the best functional model is also known as model selection problem. The optimal functional model f^* could easily be found, if we knew $P(x, y)$, and if we searched in the set of all functions $\mathcal{F}$. But as we have to restrict to minimizing Equation 2.8, it is not reasonable to search in $\mathcal{F}$. Instead, we choose a parameterized model (e.g. KNN) that represents a smaller function space $F \subset \mathcal{F}$ and to minimize the empirical risk w.r.t. this model

$$f^* = \arg\min_{f \in F} E_{emp}(f). \tag{2.10}$$

It is not very probable that the true optimal model f^* lies in F. On the one hand, it is reasonable to increase the function space. On the other hand, we have to avoid functions that only reconstruct the observations, e.g., that return $f(\mathbf{x}_i) = \mathbf{y}_i$ for $i = 1, \ldots, N$ and yield false values, e.g., the opposite class label in binary classification for all patterns we have not observed yet. Such an effect is known as overfitting.

The increase of function space F, from which f is chosen, may be reasonable to get closer to the true functional model. This is also known as increasing the capacity of the model. But to avoid overfitting, it is often sufficient to penalize the complexity of model f with a regularizer, which can be a functional norm $\|f\|$. Then, the regularized risk

$$E_{reg}(f, \lambda) = E_{emp}(f) + \lambda \|f\| \tag{2.11}$$

is minimized, where $\lambda \in \mathbb{R}^+$ is known as regularization parameter balancing between empirical risk minimization and smoothness of the function. From the perspective of KNN, we get a direct solution of this formulation. The expectation is approximated by averaging over training patterns. For large training set sizes N, conditioning of f is better than for small N, as points are likely to be close to $\mathbf{x}$. As Bishop [12] states, for $N, k \to \infty$, i.e., $k/N \to 0$ we get $f_{KNN}(x) \to E(Y|X = x)$. But the curse of dimensionality (cf. Section 2.6) weakens this argument. KNN is an excellent model for (1) low-dimensions and (2) large training set sizes. But in case of high-dimensional data spaces or few patterns, extensions of KNN are necessary [12, 40].

2.5 Cross-Validation

The problem in supervised learning is to find an adequate model f and its parameterizations. To avoid overfitting, an often employed model selection strategy is cross-validation. The idea of cross-validation is to split up the N observations $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$ into training, validation, and test set. The training set is used as basis for the learning algorithm given a potential parameter set. The validation set is used to evaluate the model given the parameter set. The optimized model $\hat{f}^*$ based on a training process with cross-validation is finally evaluated on an independent test set.

Minimization of the empirical risk on the validation set is basis of the training phase. For optimization, weak strategies like grid search are often sufficient. Finally, the approach is evaluated on the test set that has not been used for training model f. An advanced strategy to avoid overfitting is k-fold cross-validation that repeats the learning process k times with different training and validation sets. For this sake, the data set is slit up into k disjoint sets. In each step, model f employs $k-1$ sets for training and is evaluated on the remaining validation set. The error is aggregated to select the best parameters of model f on all k validation sets and is called cross-validation score. Advantage of this procedure is that all observations have been used for training the model and not only a subset of a small data set.

In case of tiny data sets, the number of patterns might be too small to prevent that model f is not biased towards the training and validation set. In this case, the k-fold cross-validation variant $k = N$ called leave-one-out

cross-validation (LOO-CV) is a recommendable strategy. In LOO-CV, one pattern is left out for prediction based on the remaining $N-1$ training patterns. The whole procedure is repeated N times.

2.6 Curse of Dimensionality

Many machine learning methods have problems in high-dimensional data spaces. The reason is an effect also know as *curse of dimensionality* or *Hughes effect*. In high-dimensional data spaces, many patterns are required to cover the whole data space, and our intuition often breaks down. Hastie *et al.* [40] gives interesting arguments for this effect that we review in the following.

Let us assume we sample points uniformly in the unit hypercube. The volume of the q-dimensional unit hypercube is 1^q. Let r be the hypercube edge length of a smaller hypercube corresponding to the volume of the unit hypercube we want to capture. Then, r^q is its volume and at the same time the volume fraction v of the unit hypercube $v = r^q$. Hence, $r = v^{1/q}$, which for example means that in $q = 10$ dimensions we have to cover $r = 0.1^{1/10} \approx 0.8$ of each variable to cover 10% of the volume of the unit hypercube with labeled patterns. In other words, covering 0.8 of each dimension with points means that still 90% of the 10-dimensional hypercube is empty.

2.7 Nearest Neighbor Queries

The search for nearest neighbors is a frequent problem in machine learning. If we have a nearest neighbor query for a pattern $\mathbf{x}'$, a simple, brute-force approach is to test the distance to each other pattern $\|\mathbf{x}' - \mathbf{x}_i\|^2$ for $i = 1, \ldots N$, with $\mathbf{x}' \neq \mathbf{x}_i$ in $O(N)$ time. But there are various possibilities to accelerate the nearest neighbor queries. If more than $\log N$ queries are necessary, sorting all pattern w.r.t. their distance may be reasonable, e.g., if $K > \log N$ for KNN. Sorting can be accomplished in the average case in $O(N \log N)$, e.g., with Quicksort, but in $O(N^2)$ in worst case. Another option might be appropriate for high-dimensional patterns. Computing the distances usually means to sum up distance values per dimension. The computation of a nearest neighbor request can be stopped, if the maximum distance to the previously computed K-nearest patterns is exceeded.

The employment of efficient data structures like k-d trees [9] and balltrees that partition the data space can result in $O(\log N)$ neighborhood requests for certain data distributions and lower dimensions. Figure 2.3 illustrates binary space partitioning trees. Every non-leaf node of a k-d tree induces a hyperplane that splits the data space into two parts. Each subtree represents the corresponding subspace. A k-d tree is constructed by first sorting the patterns w.r.t. the first dimension. The median of the sorting result is taken as pivot element and represents a node. Elements left of the node belong to

the left subtree, elements right of the node to the right subtree. Then, the two groups are sorted w.r.t. the second dimension. The process is recursively repeated until the corresponding sets consist of one element. If the data is smoothly (e.g. uniformly) distributed, half of the patterns can be excluded for each node resulting in a neighbor query lying in $O(\log N)$. Building a k-d tree takes $O(N \log N)$ time in average. Adding and removing of elements also take $O(\log N)$ time.

Experiments have shown that balltrees show better results than k-d trees, when the data is clustered, sparse, or has extra structure [84]. In this case, k-d trees may fall back to $O(N)$ runtime for one neighbor query. A ball in Euclidean space is the region bound by a hypersphere. It can be represented by center coordinates and radius. A balltree is a binary tree, whose nodes correspond to the smallest balls that contain all balls that belong to nodes of its subtrees. The balls of a balltree may intersect and do not need to cover the entire data space, which makes them applicable to sparse and non-smooth data. For a discussion on balltree construction algorithms, we refer to Omohundro [84] and Hastie *et al.* [40].

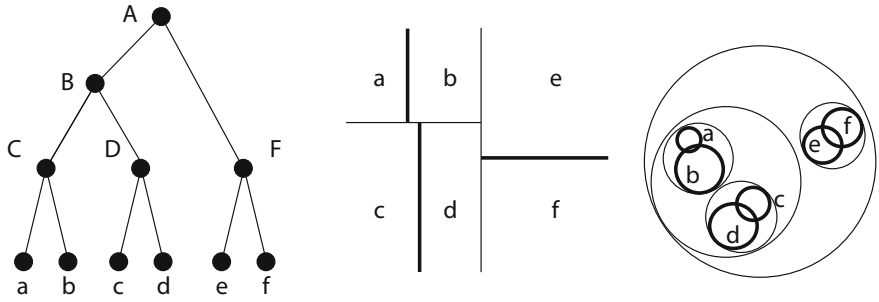


Fig. 2.3 Illustration of binary space partitioning trees: (a) If the binary tree is a k-d tree, (b) its nodes induce linear borders and divide the data space into *half-spaces*, while (c) a balltree defines a hierarchy of balls

2.8 Nearest Neighbor Variants

KNN is a technique with a long tradition. It has first been mentioned by Fix and Hodges [28] in the fifties in an unpublished US Air Force School of Aviation Medicine report as non-parametric classification technique. Cover and Hart [21] investigated the approach experimentally in the sixties. Interesting properties have been found, e.g., that for $K = 1$ and $N \to \infty$, KNN is bound by twice the Bayes error rate. Many variants of KNN have been presented in the past. Two variants are presented in the following, and a semi-supervised KNN modification is presented.

2.8.1 Model-Based KNN

The idea of model-based KNN is to replace the training set by a set of reference points (or codebook vectors) that achieve the same prediction results. This concept is related to the landmark variant of unsupervised kernel regression, which will be introduced in Section 4.9.5. The collection of landmark points is called model. The selection of a set of landmarks is treated as optimization problem, i.e., we have to search for the optimal subset of landmark vectors that achieve the same nearest neighbor result as KNN on the complete set of patterns. First, a similarity matrix from the data set is computed. All labels of $\mathbf{y}_i$ are set to *ungrouped*. Then, we seek the neighborhood that covers the largest number of neighbors with the same label. Their label is set to *grouped*. The last steps are repeated until all labels are set to *grouped*. The resulting model contains a selection for landmark vectors that can be employed as surrogate for the original KNN model.

2.8.2 Distance-Weighted KNN

KNN induces locally constant outputs. From the optimization perspective, this means we get an output space with plateaus: for neighborhood size K and N patterns in KNN regression, $\binom{N}{K}$ different output values are possible. Plateaus can hinder optimization methods from a fast approximation of the optimal solution, as not much information about promising search directions can be gained during optimization. Bailey and Jain [5] introduced distance-weighted KNN rules in the late seventies to smooth the prediction function weighting the prediction with the similarity $\Delta(\mathbf{x}', \mathbf{x}_i)$ of the nearest patterns $\mathbf{x}_i$ with $i \in \mathcal{N}_K(\mathbf{x}')$ to the target $\mathbf{x}'$

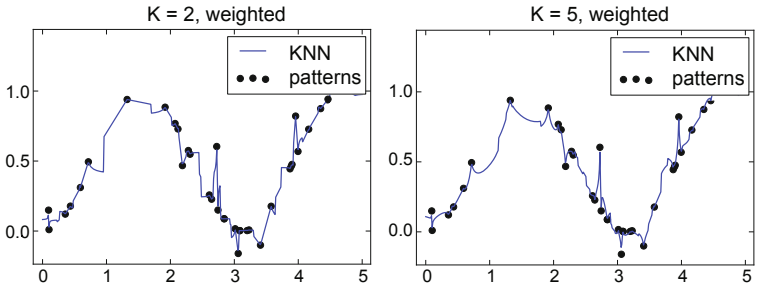
$$\mathbf{f}_{wKNN}(\mathbf{x}') = \sum_{i \in \mathcal{N}_K(\mathbf{x}')} \frac{\Delta(\mathbf{x}', \mathbf{x}_i)}{\sum_{j \in \mathcal{N}_K(\mathbf{x}')} \Delta(\mathbf{x}', \mathbf{x}_j)} \mathbf{y}_i. \tag{2.12}$$

Patterns close to the target should contribute more to the prediction than patterns that are further away. Similarity can be defined with the distance between patterns, e.g. by

$$\Delta(\mathbf{x}', \mathbf{x}_i) = 1/\|\mathbf{x}' - \mathbf{x}_i\|^2. \tag{2.13}$$

Model $\mathbf{f}_{wKNN}$ introduces a continuous output. Figure 2.4 shows the KNN prediction based on KNN regression in the weighted variant on the trigonometric function. Weighted KNN regression interpolates between the points in contrast to the uniform variant (cf. Figure 2.2).

Also weighted KNN maps to a discrete number of solutions. Machine accuracy may restrict the output space to, e.g., 2^{64} in case of 64 bits used. Uniform KNN restricts the number of possible output values to $\binom{N}{K}$. As a final remark, we state that for $K = N$ we take every pattern into account



(a) (b)

Fig. 2.4 Illustration of weighted KNN regression for (a) $K = 2$ and (b) $K = 5$

$$\mathbf{f}_{wKNN_{K=N}}(\mathbf{x}') = \sum_{i=1}^{N} \frac{\|\mathbf{x}' - \mathbf{x}_i\|^2}{\sum_{j=1}^{N} \|\mathbf{x}' - \mathbf{x}_j\|^2} \mathbf{y}_i. \tag{2.14}$$

resulting in a simplification that does not afford the computation of the nearest neighbors.

2.8.3 Propagating 1-Nearest Neighbor

Propagating 1-nearest neighbor is an approach for semi-supervised learning [116]. In semi-supervised learning, we have given a set of labeled patterns $L = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)\}$ and a set of unlabeled patterns $U = \{\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_u\}$. The idea of semi-supervised learning is that the unlabeled data enriches the learning process by yielding implicit information about the underlying data distributions. Propagating 1-nearest neighbor works as follows. In each step, the unlabeled pattern closest to any of the (already) labeled patterns is selected

$$\tilde{\mathbf{x}}^* = \min_{\tilde{\mathbf{x}} \in U} \min_{\mathbf{x} \in L} \|\tilde{\mathbf{x}} - \mathbf{x}\|^2. \tag{2.15}$$

Its label y is determined by the label of the nearest neighbor $\mathbf{x}$ in the set of labeled patterns L. Pattern $(\tilde{\mathbf{x}}^*, y)$ is added to L and removed from U. The algorithm terminates, when U is empty.

2.9 Conclusions

In this chapter, we gave an introduction to basic principles in machine learning, concentrating on supervised learning and nearest neighbor methods. Classification is the prediction of discrete class labels based on observed pattern-label pairs. Regression is the prediction of continuous values based on pattern-label observations. Nearest neighbor approaches for classification

and regression rely on the label of the K-nearest patterns in data space. In supervised learning, overfitting may occur. It can be prevented by regularization and cross-validation. Regularization is a method to avoid that functional models become too complex, while cross-validation avoids overfitting to small data sets. LOO-CV is a variant with $k = N$. Classification becomes difficult in high-dimensional data spaces, known as curse of dimensionality or Hughes effect. In case of nearest neighbor methods, the training set size has to be increased to improve the learning result. KNN variants have been introduced: from distance-weighted KNN to propagating 1-nearest neighbor for semi-supervised learning scenarios.

3

Ensemble Learning

3.1 Introduction

The hybridization of classifiers can lead to significant improvements of learning results. In this chapter, we introduce an ensemble of K-nearest neighbor and SVM classifiers and analyze its performance in a real-world application [69]. The ensembles are hybrids of *local* nearest neighbors classifiers that are based on averaging labels in the neighborhood of unknown patterns and the *global* SVMs that use separating hyperplanes.

3.2 Ensembles

To overcome algorithmic shortcomings and to achieve synergetic effects, hybridization of different methods can be an effective strategy. In recent years, a lot of research contributions devoted to hybrid solution strategies have been presented. The *no-free-lunch theorem* by Wolpert and Macready [113] states that there exists no optimal algorithm for every problem, but algorithms are tailored to special problem instances. From the perspective of hybridization, it can be an effective strategy to exploit the abilities of two or more of specialized algorithms instead of relying on a single result. The combination of predictions in classification and the exchange of successful candidate solutions in optimization improve both classifiers and optimization techniques. Two main strategies can be distinguished w.r.t. how ensembles are trained:

- *Bagging* ensembles consist of components that are trained independently. No algorithm uses knowledge about the performance of the other components and of the whole ensemble [14].
- *Boosting* ensembles try to compensate the weakness of the ensemble by concentrating on the training of its components w.r.t. the overall performance or the performance of single classifiers [29].

Bagging ensembles work as follows. From a data set consisting of N patterns, T randomly chosen subsets $S_1, \ldots, S_T$ are selected. The classifiers

O. Kramer: *Dimensionality Reduction with Unsupervised Nearest Neighb.,* ISRL 51, pp. 25–32.
DOI: 10.1007/978-3-642-38652-7_3

$f_1, \ldots, f_T$ are trained, each with the corresponding training subset. At the end, the decisions of the set of classifiers are aggregated. In case of discrete classes, the majority vote of all classifiers can be chosen as decision $f_{\mathrm{BAG}}(\mathbf{x}')$ of the ensemble given the unknown pattern $\mathbf{x}'$. In case of numerical labels, the bagging ensemble averages the predictions of the T classifiers

$$f_{\mathrm{BAG}}(\mathbf{x}') = \frac{1}{T} \sum_{i=1}^{T} f_i(\mathbf{x}'). \tag{3.1}$$

Many practical applications have shown that the best bagging ensembles combine weak and unstable classifiers, while bagging stable classifiers does often not lead to improvements.

3.3 Support Vector Machines

The ensemble classifier we use in this chapter hybridizes multi-class SVMs and multi-class KNN for appliance recognition. SVMs are based on optimizing a decision boundary in data space, while KNN is based on aggregating labels of the closest patterns. In the following, we introduce SVMs for classification. The simplest form of prediction is binary classification with label set $\{1, -1\}$. For the introduction of SVMs, we start with a linear classifier. Let $\mathbf{x} \in \mathbb{R}^q$ be a pattern in data space. The linear discriminant function

$$f(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle + b, \tag{3.2}$$

with weight $\mathbf{w} \in \mathbb{R}^q$, bias $b \in \mathbb{R}$, and scalar product (also known as inner product)

$$\langle \mathbf{w}, \mathbf{x} \rangle = \sum_{j=1}^{q} w_j x_j \tag{3.3}$$

defines a decision hyperplane. Function $f(\mathbf{x})$ divides the data space into two half spaces depending on its sign. The SVM's learning task is to find optimal parameter settings for $\mathbf{w}$ and b that separate both classes and allow the correct classification of pattern $\mathbf{x}'$

$$f_{\mathrm{LIN}}(\mathbf{x}') = \begin{cases} +1, & \text{if } f(\mathbf{x}') \geq 0 \\ -1, & \text{if } f(\mathbf{x}') < 0 \end{cases} \tag{3.4}$$

and is also known as linear classifier. The optimal linear decision hyperplane has a large distance to its closest patterns. This distance, which can be written as $1/\|\mathbf{w}\|^2$, is called *margin*. The *hard margin* SVM defines a linear decision hyperplane that correctly classifies all patterns. The optimization problem is defined as

$$\begin{aligned} &\text{minimize}_{\mathbf{w},b} \tfrac{1}{2}\|\mathbf{w}\|^2 \\ &\text{subject to : } y_i(\langle \mathbf{w}, \mathbf{x}_i \rangle + b) \geq 1 \\ &\text{for } i = 1, \ldots, N. \end{aligned} \tag{3.5}$$

In order to increase the ability to generalize, the distance between patterns and the separating hyperplane $f(\mathbf{x})$ has to be maximized, subject to a correct classification of all training patterns. As patterns are often not linearly separable, the condition of separability has to be relaxed. For this sake, *slack variables* ξ are introduced that measure the deviation of patterns from the corresponding hyperplane. The optimization problem is enhanced with a summand for the slack variables $C \cdot \sum_{i=1}^{N} \xi_i$ with $C \in \mathbb{R}$. Further, for high-dimensional and non-linear data, SVMs use kernel functions that project the data into a feature space of *higher* dimensions. In this transformed feature space, the patterns become linearly separable, and the decision hyperplane can be computed. We will employ kernel functions in Chapter 7 for dimensionality reduction tasks.

3.4 KNN-SVM-Ensemble

SVMs are advantageous in *global* scenarios, i.e., high dimensions and sparse data [40], while KNN is a *local* method, appropriate for low-dimensional data spaces and a large number of training patterns. Hence, the hybridization of both classifiers is a reasonable undertaking in practical applications, where training set sizes and numbers of features may vary. Algorithm 1 shows the pseudocode of the ensemble classifier template that is the basis of our ensemble classification process. The ensemble classifier gets the training set $\mathcal{T}$ as input. For a pattern $\mathbf{x}'$ to be classified, each classifier $f_i \in \mathfrak{f}$ of the ensemble is trained and returns a prediction. In our ensemble, all classification results are equally weighted and aggregated to one label. The decision is defined by

Algorithm 1. Ensemble Classifier

Require: Training set $\mathcal{T}$, pattern $\mathbf{x}'$, classifiers $\mathfrak{f}$
1: **for** $f_i \in \mathfrak{f}$ **do**
2: compute $f_i(\mathbf{x}')$
3: **end for**
4: **return** $f_{\text{ENS}}(\mathbf{x}') = \arg\max_{y \in \mathcal{Y}} \sum_{f_i \in \mathfrak{f}} \mathcal{I}(f_i(\mathbf{x}') = y)$

$$f_{\text{ENS}}(\mathbf{x}') = \arg\max_{y \in \mathcal{Y}} \sum_{f_i \in \mathfrak{f}} \mathcal{I}(f_i(\mathbf{x}') = y). \tag{3.6}$$

Like the multi-class KNN classifier (cf. Section 2.2.2), $f_{\text{ENS}}(\cdot)$ employs an indicator function $\mathcal{I}(\cdot)$ to count the number of classifiers that vote for each label and chooses the decision the majority of classifiers vote for. The idea of a majority vote is that the majority corrects potentially false decisions of the minority. If there is no reason to believe that one of the classifiers achieves a better accuracy than the others, a majority vote probably obtains the best predictive performance. This principle is employed in Algorithm 1. We combine the classifiers to the following ensembles:

- ENS-SVM is an SVM-ensemble classifier, which combines the SVM with linear kernel, and the SVM with RBF-kernel.
- ENS-KNN combines three KNN classifiers with different neighborhood sizes, i.e., $K = 1, 5$ and 7.
- ENS* combines all five classifiers (SVMs with both kernels and KNN with three neighborhood sizes).

In the following, we test the introduced ensembles on a real-world data set.

3.5 Recognition of Appliances

Nonintrusive appliance load monitoring (NIALM) is the problem to recognize appliances via changes in voltage and current. It can be used in households for recognition of appliances and for energy consumption analysis [39].

3.5.1 Nonintrusive Appliance Load Monitoring

In a smart grid, NIALM can be used for a variety of problem classes, e.g.: (1) In energy management and consulting, the improvement of energy-efficiency for everyday processes is an important task. But it affords the recognition of usage habits of appliances, e.g., to answers questions like *which* appliance is used *when* and *how often*. (2) For assistance systems and health-care scenarios, monitoring of everyday activities of old needy or disabled humans via the usage of appliances allows the recognition of alert states and emergency situations. (3) Load forecasting of appliances allows to balance energy systems. Balancing authorities have to consider produced and consumed energy, in particular in distributed smart grid scenarios with volatile renewable energy resources. Appliance recognition is the first step of many load forecasting systems.

Nonintrusive load monitoring of appliances has a long tradition since the mid-nineties, see Hart [39], who introduced an approach able to recognize appliances considering a continuous signal with 1 Hz sample rate. Patel *et al.* [87] employed SVMs for the recognition of 19 appliances and achieved an accuracy rate of $85 - 90\%$. Lin and Tsai [74] presented a nonintrusive load monitoring system based on hierarchical SVMs decomposing the multi-class problem into a series of binary classification problems. The approach is based on transient features from electricity waveforms. Further classification methods have been employed in the past, e.g., by Chang *et al.* [17], who apply backpropagation and learning vector quantization to load monitoring. The selection of appropriate features has an important part to play for successful recognition processes. Evolutionary approaches have been applied for feature selection [7] in load monitoring.

3.5.2 Feature Computation

For the experimental part of this chapter, we employ a data set that contains measurements of everyday appliances that are turned *on* and *off*. We use two data sets (cf. Appendix A): (1) The *install* data set consists of 120 patterns that have manually been recorded and labeled at the beginning of the field study, when the system was installed. This data set serves as minimal training set and is consequently very important for practical scenarios, e.g., for calibration of a novel system. It is balanced, i.e., the number of patterns for each class is approximately equal. (2) The data of the *field study* consists of patterns that have been recorded in a household test environment for a timespan of approximately one month. We used motion sensors in every room of the test environment for manual labeling of the data.

Based on the measurement of electrical parameters voltage $U_{\text{eff}}(t)$, amperage $I_{\text{eff}}(t)$, phase angle $\varphi(t)$, and power $P(t)$ by a current sensor with a sample rate of 5 Hz that was centrally installed, the active resistance $R(t)$ is computed

$$R(t) = \frac{U_{\text{eff}}(t)}{I_{\text{eff}}(t) \cdot \cos\varphi(t)} = \frac{U_{\text{eff}}^2(t)}{P(t)}. \tag{3.7}$$

This parameter is usually specific for the same appliance in different environments and different numbers of concurrently running appliances at a circuit. Based on the resistance from the *turn on* event of an appliance, the following features are extracted. The mean x_1, the corresponding standard deviation x_2 of the set of resistance values $\{R(1), \ldots, R(n)\}$[1] and the maximum phase of the discrete Fourier transform x_3 are computed. From the *turn off* event, the median resistance x_4 is computed. The data has not been normalized, as normalization did not have a significant influence on the experimental results. For the experimental part, we employ measurements of 15 appliances that can be turned *on* and *off*, resulting in $N = 2,620$ four-dimensional patterns and 30 different classes.

3.6 Experimental Analysis of SVM-KNN-Ensemble

In the following, we experimentally analyze multi-class SVMs, KNN, and the ensemble classifiers for appliance recognition, first for the *install* data set, then for the whole *field study* data set.

[1] The first two values $R(1)$ and $R(2)$ turned out to vary too much for the same appliance and are therefore left out.

3.6.1 Installation Data

In the first experimental setup, we employ the small data set *install* as training and validation set[2], see first row of Table 3.1. The complete *field study* data set is employed as test set. The SVM with linear kernel, KNN with $K = 1$ and two of the ensemble methods achieve a low error rate. An appliance recognition rate of over 92% is achieved, which may be sufficient for most practical scenarios. While the SVM with linear kernel is the best classifier in this scenario, the SVM with RBF-kernel fails. We expect the strength of KNN on small training sets. The data set *install* is obviously too small for the KNN classifiers. For $K = 5$ and $K = 7$, error rates larger than 25% have been achieved. The KNN-ensemble also achieves high error rates due to the results of both weak KNN variants. The other ensembles take advantage of the strengths of the SVM with linear kernel or KNN with $K = 1$.

Table 3.1 Experimental results of the SVM, KNN, and ensemble classifiers on the *install* and the *field study* data set with varying training set sizes α (proportion of training patterns and the data set size). The lowest error rates are shown in **bold** and the second best in *italic* numbers. The best classifier in each experiment gets two points, the second best one point for the score.

training set	SVM linear	SVM RBF	KNN $K=1$	KNN $K=5$	KNN $K=7$	ENS SVM	ENS KNN	ENS *
install	**0.0787**	0.4767	0.0883	0.2977	0.2927	0.0837	0.2739	*0.0802*
10^{-1}	*0.0526*	0.0915	0.0652	0.0560	0.1430	0.0594	0.0560	**0.0514**
9^{-1}	**0.0480**	0.0858	0.0606	0.0537	0.0697	0.0549	0.0526	*0.0491*
8^{-1}	**0.0480**	0.0823	0.0629	0.0549	0.0663	0.0560	0.0514	*0.0491*
7^{-1}	**0.0480**	0.0800	0.0617	0.0549	0.0617	0.0549	**0.0480**	**0.0480**
6^{-1}	0.0491	0.0789	0.0629	0.0537	0.0629	0.0549	**0.0469**	*0.0480*
5^{-1}	0.0480	0.0778	0.0606	**0.0446**	0.0491	0.0537	*0.0469*	*0.0469*
4^{-1}	0.0491	0.0709	0.0594	**0.0446**	0.0480	0.0549	*0.0469*	*0.0469*
3^{-1}	0.0514	0.0663	0.0606	**0.0434**	*0.0480*	0.0549	0.0503	0.0491
2^{-1}	0.0457	0.0617	0.0617	0.0491	**0.0434**	0.0526	0.0457	*0.0446*
$2 \cdot 3^{-1}$	0.0453	0.0572	0.0617	*0.0400*	**0.0389**	0.0572	0.0434	0.0446
$\sum$ score	9	0	0	7	5	0	6	11

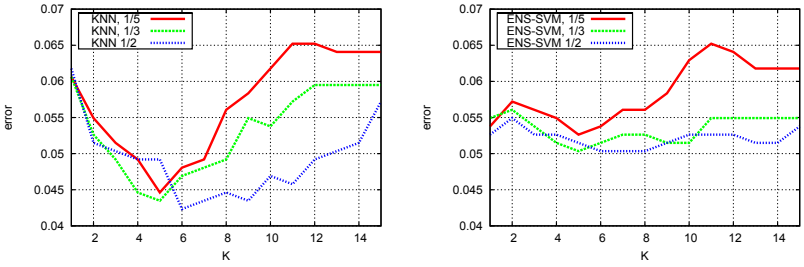
3.6.2 Field Study Data

In the second experimental setup, we analyze the classification error rate w.r.t. the rate $\alpha = \frac{|\mathcal{T}|}{N}$ of the training set size $|\mathcal{T}|$ and the number of all patterns N. The error rate is computed on a test set of size $1/3 \cdot |\mathcal{T}|$. It can

[2] For the SVM, we employ cross-validation and grid search in the ranges $C = 10^{-20}, \ldots, 10^{20}$ and for the RBF-kernel $\gamma = 10^{-20}, \ldots, 10^{20}$.

be observed that the smallest rate $\alpha = 10^{-1}$, corresponding to a training set size of $N = 262$ patterns, can achieve an accuracy of up to $\approx 95\%$ for the SVM-KNN-ensemble ENS* and also for the SVM with linear kernel. The best recognition rate (error 3.89%) is achieved with KNN and $K = 7$. For the SVM approach, we can observe that a linear kernel achieves better results than an RBF-kernel and better results than KNN with $K = 5$ and $K = 7$ for training sets smaller and equal to 6^{-1}. While the SVM with linear kernel takes significantly longer for training with training set sizes larger than 5^{-1}, it is a good recommendation for small training set sizes. With the four *high-level* features and large training set sizes, KNN achieves low error rates.

Concerning the ensemble classifiers, we can observe low error rates in most of the experiments. The ensemble classifier ENS* that employs all five classifiers turns out to be the most robust algorithm with low errors for all settings. It is the best or second best classifier in nine of eleven cases, which is also reflected by the highest sum of scores. Also the KNN-ensemble classifier ENS-KNN achieves good results on the *field study* data. The results are similar to KNN with $K = 5, 7$ (also the failure in case of the *install* data set, which cannot be compensated by KNN with $K = 1$). The constantly good results of ENS* in comparison to most other classifiers motivate the employment of the SVM-KNN-ensemble in practical scenarios.



(a) (b)

Fig. 3.1 Study of neighborhood size K w.r.t. training set size 5^{-1}, 3^{-1}, and 2^{-1} for (a) KNN and (b) ENS*. The neighborhood size has a significant influence on the classification error in case of the KNN classifiers, but the effect is compensated in the ensemble.

3.6.3 Neighborhood Sizes of KNN

In the following, we analyze the influence of neighborhood size K of the KNN classifiers and of ensemble ENS* on the recognition rate. Figure 3.1 shows the influence of K on the error rate for the KNN classifier w.r.t. different training set sizes. For KNN, see Figure 3.1(a), we can observe that neighborhood sizes around $K = 4$ to $K = 6$ are optimal for small training sets. On larger training

sets (2^{-1}), the influence of the neighborhood size is less significant. If many patterns are available, KNN can average over more training patterns without a deterioration of the classification error. In case of the ensemble ENS*, the neighborhood size of KNN is less important for both larger training sets 3^{-1} and 2^{-1}. The SVM classifiers compensate the negative effect of too large neighborhoods, which is a good motivation for the employment of ensembles.

3.7 Conclusions

We have shown that KNN and KNN-ensembles can serve as efficient and robust recognition techniques in the practical application of load monitoring. The experimental results confirmed the expectations that SVMs are a good choice in case of small training sets, while KNN shows its strengths on large training sets. We recommend to combine both worlds, i.e., KNN that can adapt to any situation without assumptions on the data distribution, but turns out to be unstable in many situations (high variance and low bias) and SVMs that are based on the assumption of linearity of the data, which is softened by kernel functions and slack variables (low variance and high bias) [40]. The ensemble classifiers are well appropriate to solve the recognition task, as practical NIALM data sets are often unbalanced, vary in training set sizes and in the number of training patterns. In particular, ENS* that employs all five classifiers with a bagging majority vote achieves the best and second best recognition rates in most experiments. The observed robustness and high recognition rates are important steps towards an efficient integrated approach of label retrieval and appliance recognition.

4

Dimensionality Reduction

4.1 Introduction

Dimensionality reduction is the task to reduce the dimensionality of patterns, while preserving important information. Many dimensionality reduction methods focus on finding low-dimensional representations of high-dimensional patterns called latent variables, latent representations, or latent embeddings. Dimensionality reduction can be employed for various tasks, e.g., visualization, preprocessing for pattern recognition methods, or for symbolic algorithms. To allow human understanding and interpretation of high-dimensional data, the reduction to 2- and 3-dimensional spaces is an important task.

Most methods compute a point-wise mapping $\mathbf{F} : \mathbb{R}^d \rightarrow \mathbb{R}^q$ from a high-dimensional data space $\mathbb{R}^d$ to a latent space of lower dimensionality $\mathbb{R}^q$ with $q < d$. For each high-dimensional pattern $\mathbf{y}_i \in \mathbb{R}^d$ with $i = 1, \ldots, N$ from a set of patterns that can be written as matrix $\mathbf{Y} = [\mathbf{y}_i]_{i=1}^N \in \mathbb{R}^{d \times N}$, a low-dimensional embedding $\mathbf{x}_i \in \mathbb{R}^q$ is computed. The set of low-dimensional representations, also organized as matrix $\mathbf{X} = [\mathbf{x}_i]_{i=1}^N \in \mathbb{R}^{q \times N}$, is called a manifold $\mathcal{M}$. The tasks is to find a manifold, which looses as little topological information as possible and thus, to identify the low-dimensional *intrinsic structure* of the high-dimensional patterns. Intrinsic structure can be defined in various kinds of ways, e.g., geometrically and semantically. We focus on methods that maintain neighborhoods and preserve topological information, in particular:

- Neighborhoods: neighbored patterns in data space should have neighbored representations in latent space.
- Distances: close patterns in data space should have latent embeddings close to each other, while distant patterns should be comparatively far away from each other in latent space.

Many dimensionality reduction methods use an implicit definition of the optimization problem they solve. The problem to learn a functional dimensionality reduction model $\mathbf{F}$ can be a hard optimization problem, because

O. Kramer: *Dimensionality Reduction with Unsupervised Nearest Neighb.,* ISRL 51, pp. 33–52.
DOI: 10.1007/978-3-642-38652-7_4

the optimal latent points are unknown. The reconstruction mapping back can also be desirable $\mathbf{f} : \mathbb{R}^q \to \mathbb{R}^d$. Some methods learn the mapping back implicitly.

In this chapter, we present well known dimensionality reduction methods and discuss measures for the quality of manifolds. We put an emphasis on unsupervised regression, which is the basis of unsupervised nearest neighbors. It is a framework that allows to employ regression methods for dimensionality reduction. The concept of unsupervised regression is to map from latent space into the high-dimensional data space employing a regression model. The learning task is to compute a set of latent points that reconstructs the high-dimensional patterns.

4.2 Feature Selection and Extraction

The simplest form of dimensionality reduction is feature selection. In many cases, only a subset of features might be relevant to solve a machine learning task, while other features are irrelevant. For example, it might be sufficient to choose only few dimensions to get an impression of the data distribution of patterns. For visualization, it might be sufficient to plot two of the N dimensions. A scatter plot presents $\binom{N}{2}$ combinations of two dimensions and often allows to get a first impression of the data space structure. A survey of feature selection methods has been introduced by Guyon [36].

Feature extraction is a further famous kind of way to reduce the dimensionality of patterns. This category comprises some of the methods presented in the remainder of this chapter, e.g. PCA. Specialized feature extraction methods are tailored to the problem domain. For auditory data, meaningful features can be generated from the high-dimensional raw features, e.g., the *mel frequency cepstral coefficient* (MFCC) features for speech recognition. From the high-dimensional patterns, meaningful new low-dimensional representations are generated that capture importance aspects to accomplish a certain task.

4.3 Clustering with K-Means

Clustering is a prominent example for unsupervised learning. The clustering task is to find groups of patterns depending on their intrinsic structure. The question is how similar patterns are to each other, and if they belong to the same group called cluster. In this learning scenario, no further information is given that helps the algorithm to group the patterns into clusters, but only their own ability to identify similarities among the patterns in the data set. We can characterize an optimal cluster assignment as follows:

- homogeneity among patterns in the same cluster and
- heterogeneity of patterns in different clusters.

Figure 4.1 shows three groups of patterns that form typical clusters. In most cases, clustering is applied to numerical patterns employing the Euclidean distance. Various evaluation criteria for clustering have been presented in the past, e.g., the Dunn index [25]. Some techniques require the specification of the number of clusters at the beginning, e.g., K-means, which is a prominent clustering method. K-means will be described in the following.

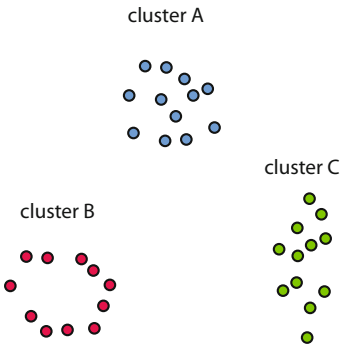


Fig. 4.1 Three agglomerations of points that form typical clusters

We define $\mathbf{c}_j$ as the barycenter of cluster j with $1 \leq j \leq K$

$$\mathbf{c}_j = \frac{\sum_{i=1}^{N} \mathcal{I}(\mathbf{y}_i, \mathbf{c}_j) \cdot \mathbf{y}_i}{\sum_{i=1}^{N} \mathcal{I}(\mathbf{y}_i, \mathbf{c}_j)}. \tag{4.1}$$

We use indicator variables $\mathcal{I}(\mathbf{y}_i, \mathbf{c}_j)$. If pattern $\mathbf{y}_i$ is assigned to *cluster* j, we set $\mathcal{I}(\mathbf{y}_i, \mathbf{c}_j) = 1$, elsewise we set $\mathcal{I}(\mathbf{y}_i, \mathbf{c}_j) = 0$. The idea of K-means is to minimize the sum of all distances between cluster centers $\mathbf{c}_1, \ldots, \mathbf{c}_K$ and patterns $\mathbf{y}_1, \ldots, \mathbf{y}_N$, which is

$$E = \sum_{i=1}^{N} \sum_{j=1}^{K} \mathcal{I}(\mathbf{y}_i, \mathbf{c}_j) \left\| \mathbf{y}_i - \mathbf{c}_j \right\|^2. \tag{4.2}$$

The underlying assumption of this minimization problem is that the cluster centers best represent the clusters, if the distances between the patterns in data space and the cluster centers are minimal. At the beginning, K-means randomly generates K initial cluster centers $\mathbf{c}_1, \ldots, \mathbf{c}_K$. In order to minimize the sum of distances E, K-means works iteratively in two steps. In the first step, each pattern $\mathbf{y}_i$ is assigned to the cluster j^* with minimal distance

$$j^* = \arg \min_{j=1,\ldots,K} \left\| \mathbf{y}_i - \mathbf{c}_j \right\|^2. \tag{4.3}$$

Equation 4.3 minimizes E keeping the cluster centers $\mathbf{c}_j$ fixed. In the next step, K-means computes the new cluster centers $\mathbf{c}_j$, while not changing the cluster assignment, see Equation 4.1. K-means repeats both steps until a termination condition is fulfilled. A frequent termination condition is that the cluster assignment does not change, or that the change of cluster centers falls below a threshold value $\psi > 0$. The process converges, but may get stuck in local optima.

4.4 Self-organizing Maps

Self-organizing maps are biologically inspired neural models. SOMs are similar to vector quantization, where patterns are iteratively presented to a set of codebook vectors $\mathbf{w}_1, \ldots, \mathbf{w}_K$. The closest codebook vector $\mathbf{w}^*$ is moved into the direction of the pattern $\mathbf{y}_i$ w.r.t. a learning rate $\eta \in (0, 1)$ with

$$\mathbf{w}^{*\prime} = \mathbf{w}^* + \eta(\mathbf{y}_i - \mathbf{w}^*). \tag{4.4}$$

The SOM by Teuvo Kohonen [54] employs a similar concept. A SOM consists of a map of neurons $n_1, \ldots, n_K$, each equipped with a weight vector $\mathbf{w}_1, \ldots, \mathbf{w}_K$ and a position $p_1, \ldots, p_K \in \mathbb{R}^q$ on a q-dimensional map. Usually, the map is 2-dimensional, and the neurons are arranged on a grid. The Euclidean or the Manhattan distance can be used as distance measure on the map.

Algorithm 2 shows the pseudocode of the SOM learning algorithm. At the beginning, the neural weights are initialized with random values. During the training phase, the high-dimensional patterns $\mathbf{y}_i$ are presented to the map in

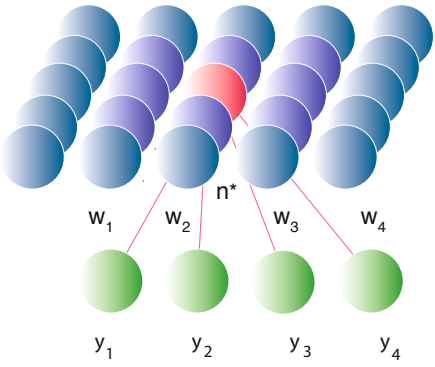


Fig. 4.2 Illustration of a 2-dimensional SOM in a 4-dimensional data space [54]. Each neuron employs a weight vector. The winner neuron n^* is shown in red. Its weights and the weights of its neighbors are moved into the direction of the input pattern.

random order. For each pattern $\mathbf{y}_i$, the SOM compares the distance to the weights of each neuron n_j of the map defined in the space of neural positions. The neuron n^* with the lowest distance δ^* to its weight vector is the winner neuron $\mathbf{w}^*$ with distance

$$\delta^* = \min_{1 \le j \le K} \|\mathbf{y}_i - \mathbf{w}_j\|^2, \tag{4.5}$$

see Figure 4.2. The weights of the winner neuron n^* and its neighbors are pulled into the direction of the pattern $\mathbf{y}_i$ depending on the learning rate $\eta \in (0, 1)$ and the neighborhood function h defined on the map, i.e., in the space of neural positions. The neighborhood function $h(n^*, n_j, r)$ must have the following characteristics:

- h maps into the interval $[0, 1]$,
- h is maximal at the center of winner neuron n^*,
- outside the range of r, i.e., for $\|p^* - p_j\|^2 > r$, it yields values close to zero.

Parameter $r \in \mathbb{R}$ is called neighborhood radius. A typical neighborhood function that fulfills these requirements is the Gaussian function.

Algorithm 2. SOM

Require: $\mathbf{Y} = [\mathbf{y}_i]_{i=1}^N$, δ, h, r
1: initialize weight vectors $\mathbf{w}_i$ of each neuron n_i
2: **repeat**
3: randomly select a pattern $\mathbf{y}_i$
4: compare $\mathbf{y}_i$ to each weight vector $\mathbf{w}_j$ of the SOM
5: winner neuron n^* has minimal distance $\delta^* = \min_{1 \le j \le K} \|\mathbf{y}_i - \mathbf{w}_j\|^2$
6: update of weight vectors $\mathbf{w}'_j = \mathbf{w}_j + \eta \cdot h(n^*, n_j, r) \cdot (\mathbf{y}_i - \mathbf{w}_j)$
7: decrease learning rate η and neighborhood radius r
8: **until** termination condition

In each generation, the SOM updates the weights of the winner and its neighborhood with the help of η und h, so that they are pulled into the direction of pattern $\mathbf{y}_i$ with

$$\mathbf{w}'_j = \mathbf{w}_j + \eta \cdot h(n^*, n_j, r) \cdot (\mathbf{y}_i - \mathbf{w}_j). \tag{4.6}$$

The algorithm leads to a mapping from data space $\mathbb{R}^d$ to the map. The mapping maintains the topology of the neighborhood. Close patterns in data space lie closely together on the map. Usually, the radius r and the learning rate η are decreased in the course of the algorithm to ensure convergence. Ritter [92] motivates the SOM from the point of view of computational neuroscience. The working principle is a result of local excitation and lateral inhibition. Within the neural layers the fields of perception receive the sensorial information.

The result of a trained 5×5-SOM employing images of galaxies from the EFIGI data set [6] (cf. Appendix A) is shown in Figure 4.3. After training, the closest image is assigned to each neuron. The SOM has obviously distributed its weights in the whole space of images considering the topological properties of distance and neighborhood preservation. Similar types of galaxies are neighbored on the map.

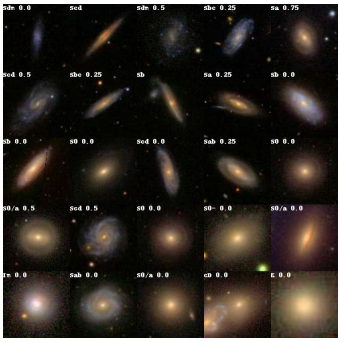


Fig. 4.3 Trained 5×5-SOM on photos of galaxies form the EFIGI data set

Sometimes, the neural weights are ill-conditioned and the network of weights is twisted around one vector [93]. To avoid such topological errors, a SOM-variant called neural gas has been introduced by Martinez [78]. The neural gas adapts the weights according to distance relations in data space. In contrast to the SOM, the neural gas does not have a predefined topological structure. Neighborhood relations are defined by the locations and topological properties in data space. Like a SOM, the neural gas employs K neurons with weight vectors $\mathbf{w}_j$. During training, for each neuron n_l the number g_j of neurons is computed that have a distance lower than the neuron itself to a pattern $\mathbf{y}_i$ with

$$g_j = \left| \{ n_l \mid l \in 1, \ldots, K \text{ with } \|\mathbf{y}_i - \mathbf{w}_l\|^2 < \|\mathbf{y}_i - \mathbf{w}_j\|^2 \} \right|. \tag{4.7}$$

Now, g_j is the basis for the weight adaptation. Like the SOM learning rule, the weight update

$$\mathbf{w}_j' = \mathbf{w}_j + \eta \cdot h(g_j, r) \cdot (\mathbf{y}_i - \mathbf{w}_j), \tag{4.8}$$

with learning rate $\eta \in \mathbb{R}^+$ depends on a neighborhood function with similar properties. It must be maximal for the winner neuron. Neighborhood function

$$h(g_j, r) = \exp\left(\frac{g_j}{r}\right) \tag{4.9}$$

is a frequent choice for the neural gas.

4.5 Principal Component Analysis

A very famous dimensionality reduction method is principal component analysis (PCA). It is designed for linear data sets. An early variant has been introduced by Pearson [88], who fitted lines and planes to a given set of points. The standard PCA, as it is most frequently known today, works as follows [47]. Given a set of d-dimensional patterns $\mathbf{y}_i \in \mathbb{R}^d$ with $i = 1, \ldots, N$, the goal of PCA is to find a linear manifold of a lower dimension $q < d$ that captures the most variance of the patterns. PCA computes the covariance matrix of the patterns

$$\mathbf{C} = \frac{1}{N-1} \sum_{i=1}^{N} (\bar{\mathbf{y}} - \mathbf{y})(\bar{\mathbf{y}} - \mathbf{y})^T \tag{4.10}$$

with mean

$$\bar{\mathbf{y}} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{y}_i. \tag{4.11}$$

If $\lambda_1 \geq \ldots \geq \lambda_q$ are the eigenvalues of the covariance matrix $\mathbf{C}$, and if $\hat{\mathbf{e}}_1, \ldots, \hat{\mathbf{e}}_d$ are the corresponding eigenvectors, we can define a $d \times q$-matrix

$$\mathbf{V}_q = [\hat{\mathbf{e}}_1, \ldots, \hat{\mathbf{e}}_q], \tag{4.12}$$

and an affine mapping $\mathbf{F}(\mathbf{y}_i) : \mathbb{R}^d \to \mathbb{R}^q$ defined as

$$\mathbf{F}(\mathbf{y}_i) = \mathbf{V}_q^T (\mathbf{y}_i - \bar{\mathbf{y}}) \tag{4.13}$$

from the data space to the q-dimensional space of the principal components $\mathbf{x}_i = \mathbf{F}(\mathbf{y}_i)$. The inverse mapping

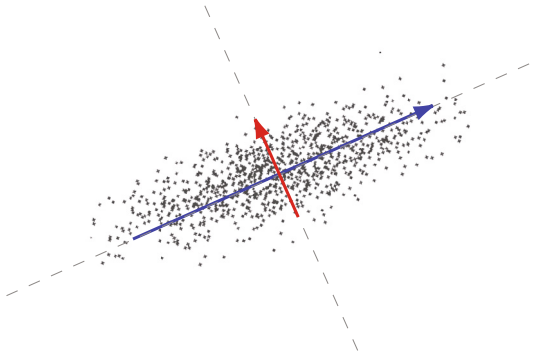


Fig. 4.4 PCA on 2-dimensional Gaussian distributed patterns and a visualization of the eigenvectors

$$\tilde{\mathbf{y}}_i = \mathbf{f}(\mathbf{x}_i) = \mathbf{V}_q \mathbf{x}_i + \bar{\mathbf{y}} \tag{4.14}$$

back to the d-dimensional data space with $\tilde{\mathbf{y}}_i$ is the projection of the pattern $\mathbf{y}_i$ onto the linear manifold. The principal components $\mathbf{x}_i$ have *zero* mean and are uncorrelated, i.e., it holds $\sum_{i=1}^{N} \mathbf{x}_j \mathbf{x}_k = 0 \; \forall \; j \neq k$.

4.6 Isometric Mapping

Isometric mapping (ISOMAP) belongs to the earliest methods for embedding non-linear data. It is based on three basic steps: First, it generates a neighborhood graph with neighborhood size K. This can be achieved using balltrees (cf. Section 2.7). Second, an $N \times N$ matrix $\mathbf{D}$ of distances is computed, setting each entry D_{ij} to the length of the shortest path between the corresponding patterns $\mathbf{y}_i$ and $\mathbf{y}_j$. The shortest-paths problem in graphs has been solved by Dijkstra [23] efficiently in $O(N^2(K + \log(N)))$. Matrix $\mathbf{D}$ contains *geodesic*, also called curvilinear distances, which allow to handle non-linear and intertwined data, see Figure 4.5. For a non-connected graph, e.g., in case of a too small choice of neighborhood size K, ISOMAP is not able to embed all patterns and a larger K has to be chosen. The idea of the last step of ISOMAP is to perform multi-dimensional scaling [70], i.e., to compute a low-dimensional matrix of points $\mathbf{X} = [\mathbf{x}_i]_{i=1}^{N}$ with distances that are consistent with the high-dimensional data space. This can be formulated as optimization problem

$$\min_{\mathbf{X}} \|\mathbf{x}_i - \mathbf{x}_j\|^2 \approx \|\mathbf{y}_i - \mathbf{y}_j\|^2 \tag{4.15}$$

with $i < j$. This step is computed with a partial eigenvalue decomposition with an approximate cost of $O(dN^2)$. Depending on the number of patterns the overall complexity of ISOMAP is in $O(N^2 \log N)$.

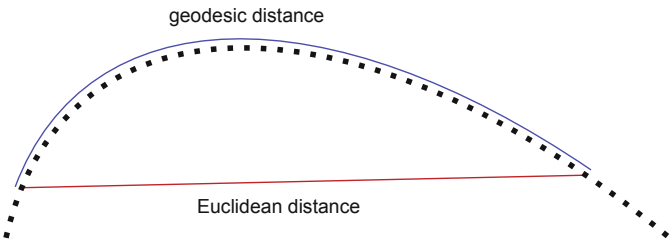


Fig. 4.5 Geodesic distances in comparison to Euclidean distance for a 2-dimensional curve

4.7 Locally Linear Embedding

For non-linear manifolds, locally linear embedding (LLE) by Roweis and Saul [94] is often employed. LLE assumes locally linearity of manifolds and computes point-wise embeddings. First, LLE computes weights that allow a linear reconstruction of point $\mathbf{y}_i$ from its K-nearest neighbors minimizing the cost function

$$E(\mathbf{w}) = \sum_{i=1}^{N} \|\mathbf{y}_i - \sum_{j=1}^{K} w_{ij}\mathbf{y}_j\|^2. \tag{4.16}$$

The search for parameters w_{ij} can be obtained as follows. First, create a matrix $\mathbf{D}$ consisting of all neighbors of $\mathbf{y}_i$, subtract $\mathbf{y}_i$ from every column of $\mathbf{D}$. Then, compute the local covariance $\mathbf{C} = \mathbf{D}'\mathbf{C}$ and solve the linear system $\mathbf{C}\mathbf{w} = 1$ for $\mathbf{w}$. Set w_{ij}, if j is not a neighbor of i. Last, set the remaining elements in the i-th row of $\mathbf{w}$ equal to $w \sum_i w_i$. The computation requires the solution of a $K \times K$ linear equation for each of the N neighborhoods, resulting in a complexity of $O(dNK^3)$. The resulting weights capture the geometric structure of the data, as they are invariant under rotation, scaling, and translation. Then, LLE computes the vector $\mathbf{x}$ best reconstructed by the weights of the previous step minimizing

$$E(\mathbf{x}) = \sum_{i=1}^{N} \|\mathbf{x}_i - \sum_{j=1}^{K} w_{ij}\mathbf{x}_j\|^2, \tag{4.17}$$

which can again be computed with a partial eigenvalue decomposition. The overall complexity of LLE w.r.t. the number of patterns is $O(N \log N + N + N^2) \in O(N^2)$. For a detailed introduction to LLE, we refer to [94] and Chang and Yeung [16] for a variant robust against outliers. In matrix form, one can take advantage of sparse matrix shortcuts, as most involved matrices have at most K non-zero entries from the neighborhood relation. LLE is employed as initialization routine for many unsupervised regression methods, cf. Section 4.9.4

A lot of further dimensionality reduction methods have been introduced that are related to each other. Independent component analysis (ICA) [19] is a popular method that assumes mutual statistical independence of the non-Gaussian source signals. Projection pursuit [30, 45] incorporates higher than second-order information and is thus able to capture non-Gaussian distributions. For an overview of methods and a further introduction to dimensionality reduction, we recommend textbooks like Hastie *et al.* [40], Bishop [12], and Lee and Verleysen [72].

4.8 Unsupervised Regression

Let $\mathcal{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_N\} \subset \mathbb{R}^d$ be the set of high-dimensional patterns with corresponding pattern matrix $\mathbf{Y} = [\mathbf{y}_i]_{i=1}^N \in \mathbb{R}^{d \times N}$. Let $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\} \subset \mathbb{R}^q$

be an arbitrary set of low-dimensional representations/latent points that define a manifold with $q < d$. Matrix $\mathbf{X} = [\mathbf{x}_i]_{i=1}^N \in \mathbb{R}^{q \times N}$ is the corresponding latent representation. The pairs $(\mathbf{x}_i, \mathbf{y}_i)$ with $1 \leq i \leq N$ are the patterns with their latent points (positions in latent space). The low-dimensional representation should represent typical characteristics of the high-dimensional data, and should loose as less information as possible, e.g., data space neighborhood relations and distances. The problem is a hard optimization problem, since the latent variables $\mathbf{X}$ are unknown.

We define the mapping $\mathbf{f} : \mathbb{R}^{q \times N} \to \mathbb{R}^{d \times N}$ from latent space $\mathbb{R}^q$ to data space $\mathbb{R}^d$ for matrices as follows

$$\mathbf{f}_{\mathbf{X}}(\mathbf{X}) = [\mathbf{f}_{\mathbf{X}}(\mathbf{x}_j)]_{j=1}^N. \tag{4.18}$$

Figure 4.6 illustrates the mapping from latent space to data space. For the optimal manifold $\mathbf{X}^*$, the data space reconstruction error (DSRE)

$$E(\mathbf{X}) = \|\mathbf{f}_{\mathbf{X}}(\mathbf{X}) - \mathbf{Y}\|_F^2 \tag{4.19}$$

with Frobenius norm $\|\cdot\|_F^2$ is minimal, i.e., it holds

$$\mathbf{X}^* = \arg \min_{\mathbf{X} \in \mathbb{R}^{q \times N}} E(\mathbf{X}). \tag{4.20}$$

For many regression methods, the optimal solution $\mathbf{X}^*$ is not unique according to the above definition, as scaling would allow an infinite number of

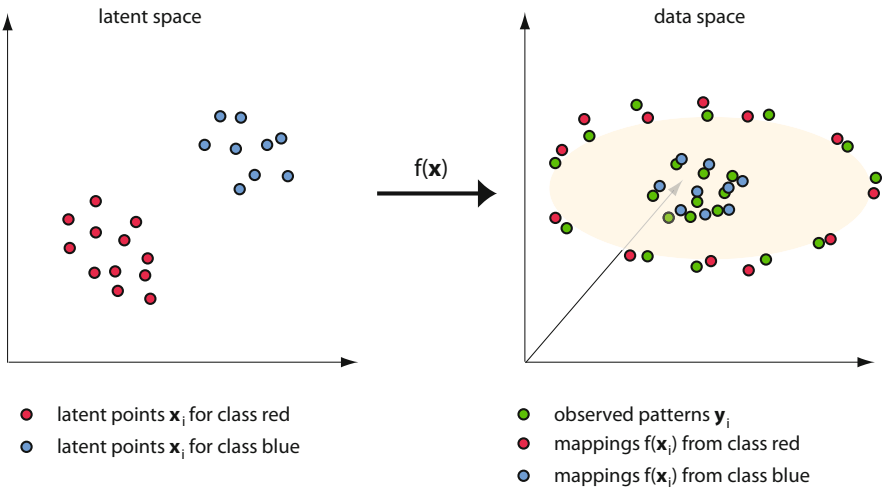


Fig. 4.6 Illustration of mapping from latent space to data space. The observed green patterns $\mathbf{y}_i$ should optimally be reconstructed by the mapping from latent variables $\mathbf{x}_i$ (red and blue) to data space $\mathbf{f}(\mathbf{x}_i) = \mathbf{y}_i$ (also red and blue).

optimal solutions. To avoid this, a regularization term $\lambda\|\mathbf{X}\|_F^2$ is added, and the optimization problem becomes

$$\mathbf{X}^* = \arg\min_{\mathbf{X}\in\mathbb{R}^{q\times N}} E(\mathbf{X}) + \lambda\|\mathbf{X}\|_F^2 \tag{4.21}$$

with penalty weight $\lambda \in \mathbb{R}^+$ that restricts latent space extension.

Most unsupervised regression approaches are based on initialization of the latent variables $\mathbf{X} = [\mathbf{x}_i]_{i=1}^N$ with LLE. Then, the DSRE $E(\mathbf{X})$ is minimized employing an optimization scheme, e.g., based on gradient descent [53]. UNN constructs a solution w.r.t. to the DSRE for an iteratively growing solution.

4.9 Unsupervised Kernel Regression

In this section, unsupervised kernel regression (UKR), one of the most prominent variants of unsupervised regression, is introduced. UKR employs the Nadaraya-Watson estimator for regression. An example for the application of UKR is the learning of low-dimensional manual actions [100]. The task is part of human-robot communication. Manual actions have been recorded with a dataglove, e.g., the movement to open a bottle. The task is to imitate the movement with a robot hand. For this sake, a low-dimensional action manifold is learned from the high-dimensional trajectory data. The objective of the approach is to represent sequences of hand postures that correspond to movement cycles. UKR was extended in such a way that time and movement representation occurred explicitly as represented latent dimensions. The structured manifolds allow to navigate the space of movements with specific characterizations like the radius of the bottle that is important in manual actions.

4.9.1 Unsupervised Regression with Nadaraya-Watson

Kernel regression weights the labels $\mathbf{y}_i \in \mathbb{R}^d$ of patterns $\mathbf{x}_i \in \mathbb{R}^q$ for $i = 1, \ldots, N$ with relative kernel densities in data space. The idea has been introduced by Nadaraya and Watson [82, 110] and is known as *Nadaraya-Watson estimator*. The UKR regression function maps from latent space to data space and is defined as follows

$$\mathbf{f}_{\mathbf{X}}(\mathbf{x}) = \sum_{i=1}^{N} \frac{K(\mathbf{x}-\mathbf{x}_i)}{\sum_{j=1}^{N} K(\mathbf{x}-\mathbf{x}_j)}\mathbf{y}_i, \tag{4.22}$$

for a given latent matrix $\mathbf{X}$ that define a candidate manifold, i.e., the low-dimensional representation of the high-dimensional pattern matrix $\mathbf{Y}$.

For convenience, Klanke and Ritter *et al.* [53] introduced a vector $\mathbf{b}(\mathbf{x}) = [b_i(\mathbf{x})]_{i=1}^N \in \mathbb{R}^N$ of basis functions that define the ratios of the density kernels with

$$b_i(\mathbf{x}) = \frac{K(\mathbf{x} - \mathbf{x}_i)}{\sum_{j=1}^N K(\mathbf{x} - \mathbf{x}_j)}. \tag{4.23}$$

Each component i of the vector $\mathbf{b}(\mathbf{x})$ contains the relative kernel density of pattern $\mathbf{x}$ w.r.t. pattern $\mathbf{x}_i$, which is the i-th column vector of matrix $\mathbf{X}$. Equation 4.22 can also be written in terms of these basis functions

$$\mathbf{f}_\mathbf{X}(\mathbf{x}) = \sum_{i=1}^N \mathbf{y}_i b_i(\mathbf{x}) = \mathbf{Y}\mathbf{b}(\mathbf{x}). \tag{4.24}$$

The matrix $\mathbf{Y}$ of high-dimensional patterns is fixed, while the basis functions $b_i(\mathbf{x})$ are tuned during the learning process. They sum up to one as they are normalized by the denominator. The optimization problem can conveniently be formulated introducing a matrix $\mathbf{B}(\mathbf{X}) \in \mathbb{R}^{N \times N}$, whose columns consist of the vectors of basis functions

$$\mathbf{B}(\mathbf{X}) = [\mathbf{b}(\mathbf{x}_j)]_{j=1}^N. \tag{4.25}$$

The product of $\mathbf{Y} \in \mathbb{R}^{d \times N}$ and $\mathbf{B} \in \mathbb{R}^{N \times N}$ results in a $d \times N$-matrix, which is the Nadaraya-Watson estimate for the whole pattern set. The quality of the manifold is evaluated with the DSRE that can now conveniently be formulated as

$$E(\mathbf{X}) = \frac{1}{N}\|\mathbf{Y} - \mathbf{Y}\mathbf{B}(\mathbf{X})\|_F^2. \tag{4.26}$$

Restriction of the solution space is necessary to avoid overfitting, cf. Section 4.9.4.

4.9.2 Kernel Density Functions

Kernel regression and UKR are based on a density estimate of patterns with a kernel density function $K : \mathbb{R}^q \to \mathbb{R}$. A typical kernel density function is the multivariate Gaussian kernel

$$K_G(\mathbf{z}) = \frac{1}{(2\pi)^{q/2}\det(\mathbf{H})} \exp\left(-\frac{1}{2}\left|\mathbf{H}^{-1}\mathbf{z}\right|^2\right), \tag{4.27}$$

with bandwidth matrix $\mathbf{H} = \mathrm{diag}(h_1, h_2, \ldots, h_q)$. Another frequent choice is the Epanechnikov kernel

$$K_E(\mathbf{z}) = D_E\left(\frac{|\mathbf{z}|}{h}\right), \tag{4.28}$$

with

$$D_E(t) = \frac{3}{4}[1 - t^2]_+ = \begin{cases} \frac{3}{4} \cdot (1 - t^2) & |t| < 1 \\ 0 & |t| \geq 1 \end{cases}. \tag{4.29}$$

The bandwidth h defines the radius of the supported region, similar to the standard deviation of the Gaussian function. Both kernel functions have useful asymptotic characteristics. For $h \to 0$, both kernel functions reconstruct the patterns, for $h \to \infty$ they average over all N patterns [37].

To increase the flexibility, parameterized kernel density functions can be employed that allow a greater flexibility to adapt to local data space characteristics. Bishop [12] states valid combinations of kernel functions, e.g., a weighted sum of the Gaussian and the Epanechnikov kernel

$$K_H(\mathbf{z}) = \alpha K_G(\mathbf{z}) + (1 - \alpha) K_E(\mathbf{z}), \tag{4.30}$$

with $\alpha \in [0, 1]$. Such a hybrid kernel function can *morph* between both kernel density functions. The parameter can be adjusted during training of the functional model.

4.9.3 Kernel Bandwidths

The results of kernel density estimators significantly depend on the choice of proper kernel parameters, e.g., kernel bandwidths. Figure 4.7 shows the influence of bandwidth parameter h on the Parzen-estimate of a sample data set with the Epanechnikov kernel. The sample data set consists of twenty 2-dimensional random vectors drawn from the Gaussian distribution with mean $\mu = (0.0, 0.0)^T$ and $\sigma = (1.0, 1.0)^T$. The figures show that the bandwidth parameter has a significant influence on the density function. For the small value $h = 0.1$, the estimate generates small *bumps* at the locations of the patterns. For the larger bandwidth $h = 2.0$, the Gaussian distribution can be recognized. Too large bandwidths lead to *oversmoothing*, i.e., averaging of the data sampled without reflecting any structure in the data.

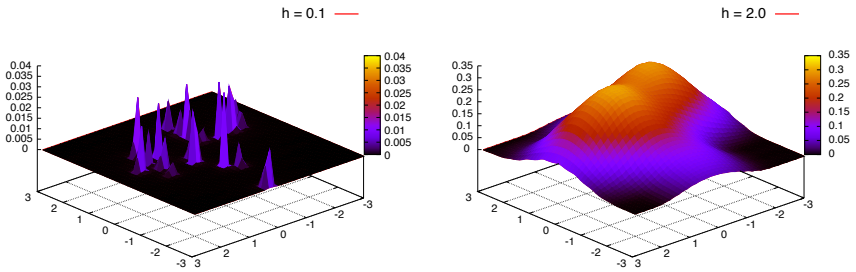
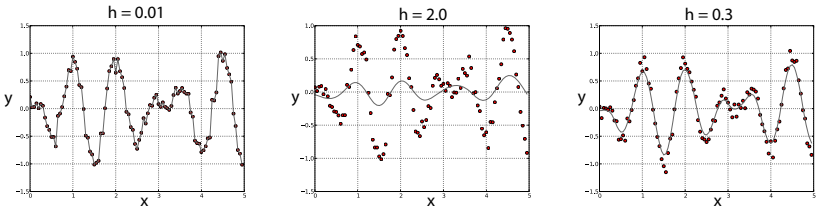


Fig. 4.7 Comparison of Epanechnikov kernel with two different bandwidth settings: (a) $h = 0.1$ and (b) $h = 2.0$ [66]

Figure 4.8 shows the influence of h on learning a noisy trigonometric function. Small values lead to an overfitted prediction function, while high values result in an overgeneralization. LOO-CV is a technique to regularize the model. Various bandwidth selection methods are known in literature. A simple and good working choice is the Silverman's rule of thumb that recommends to set the bandwidth to

$$\sigma = \hat{\sigma} c \mu^{-1/5} \tag{4.31}$$

with μ solutions, the sample standard deviation $\hat{\sigma}$ and $c = 1.06$ for a Gaussian distribution.



(a) (b) (c)

Fig. 4.8 Influence of bandwidth h on the smoothness of the regression model on a noisy trigonometric function: (a) low values ($h = 0.01$) lead to an overfitted model, while (b) high values ($h = 2.0$) overgeneralize. A smooth regression function can be achieved with (c) the choice of $h = 0.3$.

4.9.4 UKR Optimization

To avoid overfitting in UKR, penalty of extension in latent space (cf. Equation 4.21) is employed. An alternative is restriction of latent space to $[0, 1]^q$. Otherwise, optimization would move the latent points infinitely apart from each other, which only results in an overfitted reconstruction [53]. As minimizing of Equation 4.21 is a hard optimization problem, Klanke and Ritter [53] have introduced an optimization scheme consisting of half a dozen advanced steps. The scheme employs PCA and multiple LLE solutions for initialization. In short, the following optimization scheme is applied:

1. Initialization of $m + 1$ candidate solutions: m solutions from LLE, one solution from PCA,
2. selection of the best initial solution w.r.t. cross-validation error, or
3. search for optimal scale factors that scale the best LLE solution to an UKR solution w.r.t. cross-validation error,
4. selection of the most promising solution w.r.t. cross-validation error,
5. cross-validation error minimization:

- if the best solution has been generated by PCA: search for optimal regularization parameters η (with the homotopy method, cf. [53]),
- if the best solution stems from LLE: cross-validation error minimization with the homotopy method / resilient backpropagation (RPROP) by Riedmiller and Brown [91],

6. final density threshold selection.

The *spectral initialization* with LLE is an often employed procedure to generate satisfying initial UKR models. The UKR-based optimization process plays the role of post-optimization of LLE solutions.

4.9.5 Algorithmic Variants

In the following, we give a short overview of UKR variants. A *landmark variant* by Klanke [52] is based on the idea of finding representative codebook vectors instead of employing the complete set of patterns, which can make the kernel density computations slow in case of large data sets. The landmark variant can reduce the computational effort during training of the model, for sampling from the manifold, and for projecting new data. A set of landmark points $\hat{\mathbf{Y}} = \{\hat{\mathbf{y}}_1, \ldots, \hat{\mathbf{y}}_{\hat{N}}\}$ with $\hat{N} < N$ is selected from $\mathbf{y}_1, \ldots, \mathbf{y}_N$. Then, $\mathbf{Y}$ is reconstructed from a UKR model utilizing only $\hat{\mathbf{Y}}$ in the regression function

$$\mathbf{f}_{\hat{\mathbf{X}}}(\mathbf{x}) = \hat{\mathbf{Y}}\mathbf{b}(\hat{\mathbf{x}}). \tag{4.32}$$

Landmark variants are an effective way to reduce the computational complexity of machine learning methods by concentrating on a subset of patterns.

A *feature space variant* of UKR introduced by Klanke [52] extends UKR by a kernel for computation of the DSRE in a feature space. In experiments, the approach turned out to achieve lower errors than the native approach. Kernels for feature space map the patterns from data space to a space of higher dimensionality. In this space, non-linearities are softened. We introduce such a kernel approach for UNN in Chapter 7.

LOO-CV is an *extreme* variant of cross-validation and well appropriate for small data sets, cf. Section 2.5. LOO-CV is based on leaving out each pattern as validation set, i.e. setting $n = N$. It is known to yield an unbiased estimate of the prediction error. Often, the drawback of LOO-CV is that it can be computationally inefficient. But for UKR, LOO-CV can be implemented very efficiently by setting the diagonal entries of $\mathbf{X}$ to *zero* and normalizing the columns before applying Equation 4.26. Leave-c-out cross-validation with $c = 3$ and $c = 5$ have been observed to produce lower variances, but may overestimate [52].

In [62, 63] we present an evolutionary UKR variant and employ the covariance matrix adaptation evolution strategy (CMA-ES) (cf. Chapter 6) to solve two steps of the UKR optimization framework with the objective to replace the complicated optimization scheme by the shorter framework:

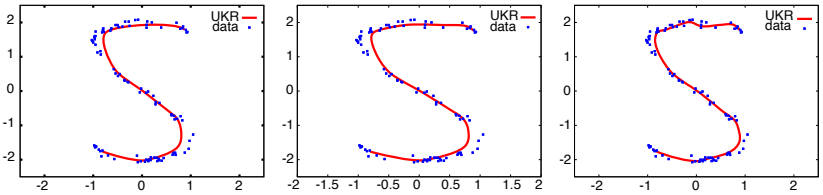


Fig. 4.9 Evolutionary UKR on 2-dimensional noisy $\mathcal{S}$ data set. The figures show the results of evolutionary optimization runs with three different ratios of optimization steps spent on scaling and final cross-validation error minimization, i.e., 0/4, 2/2, 3/1 [66]. The model on the right is slightly overfitted.

1. Initialization of m candidate LLE solutions,
2. selection of the best initial solution w.r.t. cross-validation error,
3. search for optimal scale factors with the CMA-ES and
4. cross-validation error minimization with the CMA-ES.

A positive effect of the employment of an evolutionary scheme is that arbitrary, also non-differentiable kernel functions can be used, as the derivatives do not have to be computed using blackbox optimization. Figure 4.9 shows the evolved UKR manifold of a 2-dimensional $\mathcal{S}$-structure consisting of $N = 100$ patterns with noise for three different ratios of optimization steps of scaling factors and cross-validation error minimization [62]. We employed Huber's loss function [44] for training the evolutionary UKR model.

4.9.6 Further Regression Methods

Carreira-Perpiñán and Lu [15] argue that training parametric methods can accelerate learning, e.g., unsupervised regression based on radial basis function networks (RBFs) [99], Gaussian processes [71], and neural networks [102]. They usually do not require to take into account all N patterns in each learning or prediction step. An unsupervised regression approach employing neural networks has been introduced by Tan and Mavrovouniotis [102]. It is based on a three-layer auto-associative network, which is trained with gradient descent on the latent variables and neural weights. A further variant is unsupervised local polynomial regression (ULPR) that employs polynomial regression, which is very related to kernel regression, in the UR framework [52]. Local polynomial regression fits multiple simple functions $\mathbf{f}_1(\cdot), \ldots, \mathbf{f}_N(\cdot)$ to each pattern that are weighted with kernel density function $K(\cdot)$:

$$E(\mathbf{X}, \mathbf{f}_1, \ldots, \mathbf{f}_N) = \sum_{i=1}^{N} K(\mathbf{x} - \mathbf{x}_i) \|\mathbf{y}_i - \mathbf{f}_i(\mathbf{x}_i)\|^2 \tag{4.33}$$

Polynomials are more flexible than the kernel density weight of unsupervised kernel regression, and ULPR leads to better embeddings. But fitting a polynomial at each point is computationally very expensive.

Carreira-Perpiñán and Lu [15] extend the unsupervised regression optimization problem to a more general formulation, which takes into account the mapping backwards from latent space to data space. The unsupervised regression problem is formulated as minimization problem in the following form

$$\min_{\mathbf{X},\mathbf{f_X},\mathbf{F_Y}} E(\mathbf{X},\mathbf{f_X},\mathbf{F_Y}) = E(\mathbf{X},\mathbf{f_X}) + E(\mathbf{X},\mathbf{F_Y}) \tag{4.34}$$

with

$$E(\mathbf{X},\mathbf{f_X}) = \|\mathbf{Y} - \mathbf{f_X}(\mathbf{X})\|_F^2 + \lambda_\mathbf{f}\|\mathbf{f_X}\|, \tag{4.35}$$

and

$$E(\mathbf{X},\mathbf{F_Y}) = \|\mathbf{X} - \mathbf{F_Y}(\mathbf{Y})\|_F^2 + \lambda_\mathbf{F}\|\mathbf{F_Y}\|. \tag{4.36}$$

with parameters $\lambda_\mathbf{f}, \lambda_\mathbf{F} \in \mathbb{R}$. Further, $\mathbf{X} \in \mathbb{R}^{d\times N}$ is finite-dimensional, while $\mathbf{f_X}$ and $\mathbf{F_Y}$ belong to appropriate infinite-dimensional function spaces. Here, $\|\cdot\|$ is also used as functional norm for $\mathbf{f}$ and $\mathbf{F}$, while $E(\mathbf{X},\mathbf{f_X})$ and $E(\mathbf{X},\mathbf{F_Y})$ are formulated as regularized least-squares regression problem. $E(\mathbf{X},\mathbf{f_X})$ and $E(\mathbf{X},\mathbf{F_Y})$ are competing terms of separating and clustering $\mathbf{X}$. $E(\mathbf{X},\mathbf{f_X})$ separates points in $\mathbf{X}$ from each other so that $\mathbf{f_X}$ can more easily interpolate and reduce the error $\|\mathbf{Y} - \mathbf{f_X}(\mathbf{X})\|_F^2$. Along the way, $E(\mathbf{X},\mathbf{F_Y})$ drives the latent points from $\mathbf{X}$ to $\mathbf{0}$ so that $\mathbf{F}$ can smoothly interpolate $(\mathbf{X},\mathbf{Y})$ by $\mathbf{F_Y} = \mathbf{0}$. Optimization of both objectives often results in good mappings, while the concentration on one of the two terms turns out to be disadvantageous in experimental studies. Carreira-Perpiñán and Lu proposed an alternating scheme of adaptation and projection. In the adaptation step, $\mathbf{X}$ is kept constant and a solution for Equation 4.34 is computed. This is performed with radial basis function expansions at each of the patterns and a translation of the optimization problem to a system of linear equations. In the projection step, $\mathbf{f_X}$ and $\mathbf{F_Y}$ are fixed, while Equations 4.35 and 4.36 are minimized. This requires the solution of a non-convex optimization problem, which can be quite difficult to optimize.

4.10 Quality Measures for Latent Embeddings

The idea of dimensionality reduction is to compute a low-dimensional representation of the high-dimensional patterns that preserve most of the information. To evaluate dimensionality reduction results, various quality measures have been introduced. In the following, we present and discuss three quality criteria. The co-ranking matrix measure will guide as evaluation criterion in the experimental parts of this work.

4.10.1 Quantization Error

The quantization error is a simple quality measure for vector quantization and related methods [8], e.g. SOMs and the neural gas. It measures the average distance between each pattern $\mathbf{y}_i \in \mathbb{R}^d$ with $i = 1, \ldots, N$ and its closest codebook vector $\mathbf{c}_{i^*}$, which is known as best matching unit

$$E_Q = \frac{1}{N} \sum_{i=1}^{N} \| \mathbf{y}_i - \mathbf{c}_{i^*} \|^2. \tag{4.37}$$

The quantization error is only an applicable quality measure for approaches that employ codebook vectors. The measure is neglecting that codebook vectors might lie at locations, where not patterns are located. In this case, only few codebook vectors might be distributed in data space leading to a comparatively low error, while others are far away from the main clusters of patterns.

4.10.2 Topographic Error

The topographic error by Kiviluoto [51] is also designed for methods that employ codebook vectors. It determines the best $\mathbf{c}_{i^{*1}}$ and the second best $\mathbf{c}_{i^{*2}}$ matching units for each pattern $\mathbf{y}_i$ and counts every case, in which both are not neighbored on the topographic map

$$E_T = \frac{1}{N} \sum_{i=1}^{N} \mathcal{I}(\mathbf{c}_{i^{*1}}, \mathbf{c}_{i^{*2}}) \tag{4.38}$$

with $\mathcal{I}(\mathbf{c}_{i^{*1}}, \mathbf{c}_{i^{*2}}) = 1$, if $\mathbf{c}_{i^{*1}}$ and $\mathbf{c}_{i^{*2}}$ are not neighbored. This measure takes into account that neighbored positions in latent space should correspond to neighbored positions in data space. The concept is already quite close to the concept of the co-ranking matrix that will be introduced in the next section. Similar to the quantization error, problems may occur, if codebook vectors are not well distributed in data space. For example, an extreme case is that for most patterns only two best matching units are neighbored, while all other best matching units are far away. This bad embedding would result in a low error.

4.10.3 Co-ranking Matrix

Lee and Verleysen [73] introduced a measure for the evaluation of latent embeddings called co-ranking matrix. It is based on the comparison of ranks w.r.t. distance-based sorting of patterns in data space and in latent space. Let $\mathbf{y}_i$ and $\mathbf{y}_j$ be two patterns in data space with corresponding latent positions $\mathbf{x}_i$ and $\mathbf{x}_j$ in latent space. The rank ρ_{ij} of $\mathbf{y}_j$ with respect to $\mathbf{y}_i$ in data space is defined by

$$\rho_{ij} = |\{k \mid \|\mathbf{y}_i - \mathbf{y}_k\|^2 \leq \|\mathbf{y}_i - \mathbf{y}_j\|^2 \text{ and } 1 \leq k < j \leq N\}|. \tag{4.39}$$

With the help of parameter k, the set of indices of patterns is counted with a smaller distance to $\mathbf{y}_i$ than $\mathbf{y}_j$. An equivalent definition specifies the ranks in latent space that we denote as r_{ij}. Now, we can define a co-ranking matrix $\mathbf{Q}$ that explicitly states the deviations of ranks in data and latent space

$$q_{kl} = |\{(i,j) \mid \rho_{ij} = k \text{ and } r_{ij} = l\}|. \tag{4.40}$$

In this matrix, rank errors correspond to off-diagonal entries. A point $\mathbf{y}_j$ with lower rank w.r.t. a point $\mathbf{y}_i$ in latent space, i.e., $\rho_{ij} > r_{ij}$, is called *intrusion*. In the case $\rho_{ij} < r_{ij}$, it is called *extrusion*, see Figure 4.10.

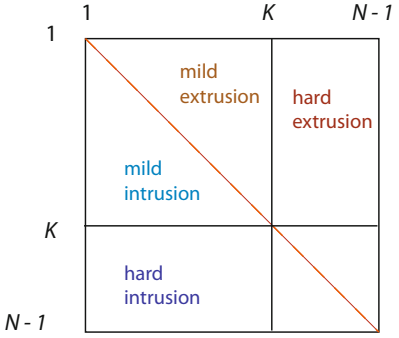


Fig. 4.10 Illustration of co-ranking concept oriented to the work of Lee and Verleysen [73]. Points $\mathbf{y}_j$ with lower rank w.r.t. a point $\mathbf{y}_i$ in latent space ($\rho_{ij} > r_{ij}$) are called *intrusions*. In turn, points with $\rho_{ij} < r_{ij}$ are called *extrusions*.

From the co-ranking matrix, the following quality measure can be derived that counts the number of proper ranks within a neighborhood of size K

$$E_{NX}(K) = \frac{1}{KN} \sum_{k=1}^{K} \sum_{l=1}^{K} q_{kl} \tag{4.41}$$

This term restricts the measure to neighborhoods of size K. High values for E_{NX} show that the high-dimensional neighborhood relations are preserved in latent space. A perfect embedding achieves a value of one, i.e., all K-nearest neighbors of N patterns in data space have their latent representations in the set of K-nearest neighbors in latent space. Extensions of the measure have been presented, e.g., weighting the rank errors and defining a tolerance level. A discussion can be found by Lueks *et al.* [75].

4.11 Conclusions

Dimensionality reduction is the task of learning a mapping from high-dimensional data space to a latent space with lower dimensions, while preserving topological information. A broad range of problems can be solved with dimensionality reduction methods, e.g., feature preprocessing, fault detection, monitoring, and visualization. We have shortly revisited prominent approaches, from clustering with K-means and vector quantization with SOMs to linear dimensionality reduction with PCA and non-linear embeddings with ISOMAP and LLE. Further, evaluation criteria for embeddings have been presented. The quantization error and the topographic error have been introduced for methods employing codebook vectors. The co-ranking matrix measure is appropriate for algorithms that compute point-wise embeddings.

The unsupervised counterparts of regression methods are powerful manifold learning techniques. They are based on optimizing latent variables w.r.t. the regression based reconstruction of high-dimensional patterns. This objective guides the optimization process and will also be the basis of the unsupervised nearest neighbors optimization process that will be introduced in the following chapter. UKR is a famous representative of this class of methods. UKR solutions are usually initialized with PCA or LLE solutions and post-processed with gradient descent in latent space w.r.t. unsupervised regression. We have shown how evolutionary methods can be used to optimize UKR manifolds employing LOO-CV and robust loss functions on a simple test data set. Although many interesting UKR variants have been presented in the past, unsupervised regression has not become as popular as similar methods, e.g., PCA, LLE, or ISOMAP. Only few applications based on unsupervised regression and UKR have been introduced in the last years, e.g., learning of intrinsic structures of robot arm motions [100]. Besides kernel regression, other methods have been employed in the unsupervised regression framework, from local polynomial regression to feedforward networks.

Part II

Unsupervised Nearest Neighbors

5

Latent Sorting

5.1 Introduction

In this chapter, we introduce the first variant of unsupervised nearest neighbors for embedding patterns in discrete latent topologies. For this sake, we first introduce the basic principle of fitting nearest neighbor regression to the unsupervised regression framework of the last chapter. The nearest neighbor principle is a simple and efficient approach. With UNN, we present an iterative method for the construction of low-dimensional embeddings, which allows to cope with large data sets. To accelerate the method, a greedy variant restricts the search process to the latent neighborhoods of the closest embedded patterns. All presented methods will be analyzed experimentally. In the remainder of this book, various optimization strategies for UNN will be introduced, and the approach will be extended step by step.

In practical scenarios, data sets are noisy and entries are missing. Besides modeling of the data mining process, selection of relevant features, and tasks like normalization, the practitioner often has to preprocess the data. Failures and environmental conditions can lead to noise and missing entries. We introduce and analyze methods to cope with noisy and missing data for UNN. To cope with noise, we employ the ϵ-insensitive loss. Parameter ϵ allows to adapt the level of noise to ignore. Experiments will show that the search for proper parameters can improve the UNN learning results. To cope with missing data, we introduce two approaches:

- Repair-and-embed first repairs incomplete patterns iteratively with KNN regression and then embeds the repaired patterns employing UNN. For the imputation process, the completed patterns are used to predict the missing values iteratively.
- Embed-and-repair first embeds incomplete patterns ignoring the features with missing data. After embedding, the gaps are filled so that the embedded patterns optimally fit into the latent positions minimizing the DSRE.

O. Kramer: *Dimensionality Reduction with Unsupervised Nearest Neighb.,* ISRL 51, pp. 55–73.
DOI: 10.1007/978-3-642-38652-7_5

With the help of experiments, we will carefully analyze the missing data strategies w.r.t. an increasing rate of missing values considering the imputation performance and a comparison of the DSRE with complete data.

5.2 Unsupervised Nearest Neighbors

First, we introduce the iterative UNN optimization framework presenting the notation for iteratively growing latent and pattern matrices.

5.2.1 Iterative Unsupervised Regression

We seek for a manifold that minimizes Equation 4.20. Unsupervised nearest neighbors is an approach that constructs the manifold by iteratively adding locally optimal latent points w.r.t. a growing pattern set. For the iterative procedure, we define a notation for growing latent matrices $\overline{\mathbf{X}} \in \mathbb{R}^{q \times n}$, and pattern matrices $\overline{\mathbf{Y}} \in \mathbb{R}^{d \times n}$ for number $1 \leq n \leq N$ of currently embedded patterns.

Let $\mathbf{Y}$ be the complete pattern matrix with an arbitrary order of patterns that can randomly be changed at initialization. At the beginning, for the first pattern $\mathbf{y}_1$, an arbitrary grid position is chosen, e.g., $\mathbf{x}_1 = [\mathbf{0}]$. The latent matrix becomes $\overline{\mathbf{X}} = [\mathbf{x}_1]$, and the corresponding pattern matrix becomes $\overline{\mathbf{Y}} = [\mathbf{y}_1]$.

Let $\mathbf{y}_1, \ldots, \mathbf{y}_n$ be the sequence of already considered patterns with associated embeddings $\mathbf{x}_1, \ldots, \mathbf{x}_n$. For the next pattern $\mathbf{y}_i$ with $i = n + 1 \leq N$, UNN generates a set of κ latent position candidates $\mathbf{x}_1^*, \ldots, \mathbf{x}_\kappa^*$, e.g., by testing nodes on a q-dimensional lattice structure or by generating randomly sampled points. The candidate latent point is chosen that minimizes the contribution to the DSRE

$$\mathbf{x}_i = \arg \min_{\mathbf{x}=\mathbf{x}_1^*,\ldots,\mathbf{x}_\kappa^*} e_{\overline{\mathbf{X}}}(\mathbf{x}), \tag{5.1}$$

which is defined as

$$e_{\overline{\mathbf{X}}}(\mathbf{x}) = \|\mathbf{f}_{\overline{\mathbf{X}}}(\mathbf{x}) - \mathbf{y}_i\|^2. \tag{5.2}$$

We can also choose the latent position that minimizes the DSRE of the complete manifold

$$\mathbf{x}_i = \arg \min_{\mathbf{x}=\mathbf{x}_1^*,\ldots,\mathbf{x}_\kappa^*} E(\overline{\mathbf{X}}) + \lambda \|\overline{\mathbf{X}}\|_F^2 \tag{5.3}$$

with

$$E(\overline{\mathbf{X}}) = \|\mathbf{f}_{\overline{\mathbf{X}}}(\overline{\mathbf{X}}) - \overline{\mathbf{Y}}\|_F^2, \tag{5.4}$$

and parameter $\lambda \in \mathbb{R}^+$. Here, $\mathbf{f}_{\overline{\mathbf{X}}}(\cdot) : \mathbb{R}^q \to \mathbb{R}^d$ is the K-nearest neighbor (KNN) regression model defined as

$$\mathbf{f}_{\overline{\mathbf{X}}}(\mathbf{x}) = \frac{1}{K} \sum_{j \in \mathcal{N}_K(\mathbf{x}, \overline{\mathbf{X}})} \mathbf{y}_j \tag{5.5}$$

with the set of indices $\mathcal{N}_K(\mathbf{x}, \overline{\mathbf{X}})$ of the K-nearest latent points to latent position $\mathbf{x}$ given a pattern matrix $\overline{\mathbf{X}} = [\mathbf{x}_j]_{j=1}^n$. In the end, we seek for an optimal complete latent matrix $\mathbf{X}^* = \overline{\mathbf{X}}^*$ with $n = N$.

For KNN, neighborhoods are invariant w.r.t. scaling of latent vectors. Hence, the optimal solution is not unique. For practical purposes, e.g., due to the limitation of the size of numbers on machines, it might be reasonable to restrict continuous KNN latent spaces, e.g., to $\mathbf{x}_i \in [0, 1]^q$. The proposed strategies for generation of latent points automatically restrict extension in latent space. In the following section, discrete latent space topologies are used that do not require further regularization.

Optimal latent points $\mathbf{X}^*$ and patterns $\mathbf{Y}$ yield mappings in both directions, i.e., from latent space to data space $\mathbf{f}_{\mathbf{X}^*}(\cdot)$ and from data space to latent space

$$\mathbf{F}_{\mathbf{X}^*}(\mathbf{y}) = \frac{1}{K} \sum_{j \in \mathcal{N}_K(\mathbf{y}, \mathbf{Y})} \mathbf{x}_j^*. \tag{5.6}$$

For KNN, not the absolute latent positions are relevant, but the relative positions that define the *neighborhood relations*. This perspective reduces the problem to a combinatorial search in the space of neighborhoods $\mathcal{N}_K(\mathbf{x}_i, \mathbf{X})$ with $i = 1, \ldots, N$ that can be solved by testing all combinations of K-element subsets of N elements, i.e., all $\binom{N}{K}$ combinations. The problem is still difficult to solve, in particular for large numbers of patterns.

5.2.2 Latent Sorting

For generation of latent positions we assume that latent points lie on a lattice structure in the following. For $q = 1$, finding appropriate latent positions on a line is similar to finding the best sorting of patterns. Latent sorting works as follows. For the first pattern $\mathbf{y}_1$, an arbitrary grid position can be chosen, e.g., $\mathbf{x}_1 = [1]$. The latent matrix is $\overline{\mathbf{X}} = [\mathbf{x}_1]$, and the corresponding pattern matrix is $\overline{\mathbf{Y}} = [\mathbf{y}_1]$.

Let $\mathbf{y}_1, \ldots, \mathbf{y}_n$ be the sequence of already considered patterns with associated embeddings $\mathbf{x}_1, \ldots, \mathbf{x}_n$. For the next pattern $\mathbf{y}_i$ with $i = n + 1 \leq N$, UNN generates $n + 1$ candidate latent positions $0.5, 1.5, \ldots, n + 0.5$. The optimal latent position $\mathbf{x}^* = i^*$ is chosen that minimizes Equation 5.2, or Equation 5.4 respectively. Algorithm 3 shows the algorithm in pseudocode. After choosing the optimal latent position i^*, all latent points get the latent position that corresponds to their rank in an increasing order $1, \ldots, n+1$. Finally, the latent matrix is extended[1] by the novel latent position $\overline{\mathbf{X}} = [\overline{\mathbf{X}}, i^*]$, the pattern $\mathbf{y}_i$ is added to the pattern matrix $\overline{\mathbf{Y}} = [\overline{\mathbf{Y}}, \mathbf{y}_i]$.

Figure 5.1 illustrates the $n + 1$ possible embeddings of a pattern into an existing order of points in latent space (yellow circles) for $K = 2$. The position

[1] The elements (vectors and matrices) within the parentheses $[\cdot]$ are concatenated to a matrix.

Algorithm 3. Latent Sorting

Require: $\mathbf{Y}$, K
1: $\overline{\mathbf{X}} = [\mathbf{x}_1]$, $\overline{\mathbf{Y}} = [\mathbf{y}_1]$
2: **for** $i = 2$ **to** N **do**
3: choose $\mathbf{y}_i$
4: **for** $i = 0.5$ **to** $n + 0.5$ **do**
5: test intermediate position i for $\mathbf{x}_i$ in latent space
6: **end for**
7: choose position $\mathbf{x}^* = i^*$ that minimizes $e_{\overline{\mathbf{X}}}(\mathbf{x})$ or $E(\overline{\mathbf{X}})$
8: regularize grid
9: $\overline{\mathbf{X}} = [\overline{\mathbf{X}}, \mathbf{x}^*]$, $\overline{\mathbf{Y}} = [\overline{\mathbf{Y}}, \mathbf{y}_i]$
10: **end for**

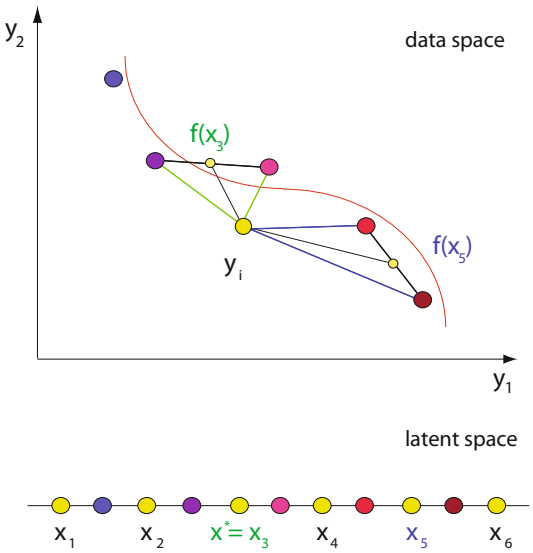


Fig. 5.1 Illustration of UNN latent sorting when embedding a pattern $\mathbf{y}_i$ in a discrete latent space topology w.r.t. the DSRE testing $n + 1$ positions

of element $\mathbf{x}_3$ results in a lower DSRE than the position of $\mathbf{x}_5$, as the mean of the patterns of the two nearest neighbors of $\mathbf{x}_3$ is closer to $\mathbf{y}_i$ than the mean of the patterns of the two nearest neighbors of $\mathbf{x}_5$.

The embedding of N patterns takes $(N+1)\cdot(N+2)/2$ DSRE computations, i.e., UNN takes $O(N^2)$ time. In 1-dimensional latent space, it is easily possible to save the K-nearest neighbors in latent space in a list, so that the search for indices can be accelerated.

A greedy variant UNN$_g$ [55] only tests the neighbored positions in latent space of the nearest embedded point of pattern $\mathbf{y}_i$

$$\mathbf{y}^* = \arg \min_{\mathbf{y}=\overline{\mathbf{y}}_1,\ldots,\overline{\mathbf{y}}_n} \|\mathbf{y}_i - \mathbf{y}\|^2 \tag{5.7}$$

in data space with corresponding latent position $\mathbf{x}^*$. Only the neighbored latent positions of $\mathbf{x}^*$, i.e., $\mathbf{x}^* - 0.5$ and $\mathbf{x}^* + 0.5$ are tested. The latent position i^* is chosen that minimizes $E(\overline{\mathbf{X}})$. All other steps are the same as in the standard UNN variant. Figure 5.2 illustrates the embedding of a 2-dimensional pattern $\mathbf{y}_i$ (yellow) left or right of the nearest pattern $\mathbf{y}^*$ in data space. The position with the lowest DSRE is chosen.

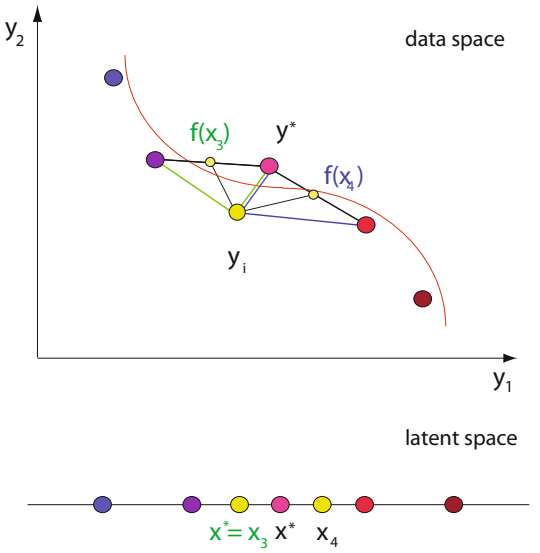


Fig. 5.2 UNN$_g$ is testing the neighbored positions of the nearest pattern $\mathbf{y}^*$ in data space

UNN$_g$ computes the nearest embedded point $\mathbf{y}^*$ in $O(N)$ for each pattern. Further, for each pattern, two DSRE computations are necessary. Hence, also UNN$_g$ takes $O(N^2)$ time for the complete embedding. But if DSRE computations take significantly longer than nearest neighbor searches in data space, UNN$_g$ is a recommendable variant.

5.3 Experimental Analysis of UNN

In the following, we present an experimental evaluation of UNN regression on the test data sets S, USPS *Digits* [46], and the EFIGI data set of galaxy images (cf. Appendix A).

5.3.1 S and USPS

The 3D-S variant without a hole (3D-S) consists of $N = 500$ patterns. Figure 5.3(a) shows the order of elements of the 3D-S data set at the beginning: all latent embeddings are unsorted. The other parts of Figure 5.3 show embeddings of UNN with different neighborhood sizes, i.e., $K = 2, 10, 200$. In case of too large neighborhood settings, UNN cannot distinguish between patterns neighbored on the surface of the S-manifold and patterns being closer in data space.

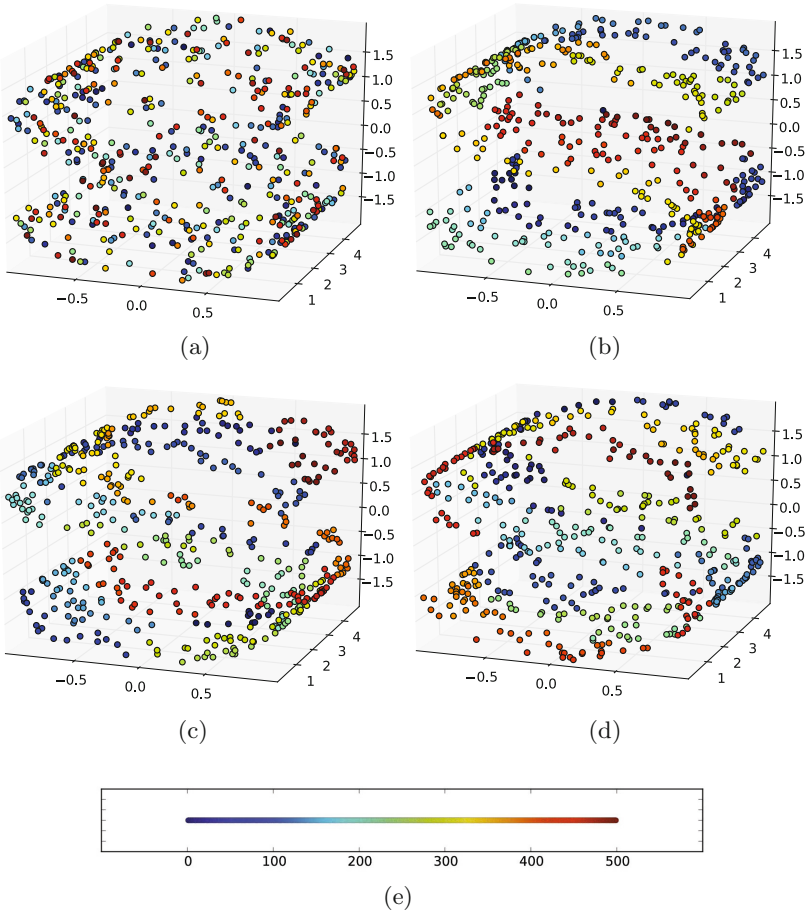


Fig. 5.3 UNN Embeddings: (a) 3D-S data set at the beginning and learning results with various neighborhood sizes, i.e., (b) $K = 2$, (c) $K = 10$, and (d) $K = 200$. Figure (e) shows the corresponding latent space consisting of $N = 500$ latent points.

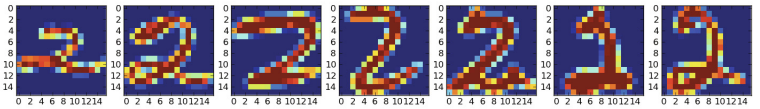


Fig. 5.4 Latent sorting of 100 digits ('2's) from the *Digits* data set with UNN$_g$. Images assigned to every 14th embedded latent point are shown. Similar digits are neighbored in latent space.

Figure 5.4 shows the embedding of 100 patterns of 256-dimensional (16 x 16 pixels) images of handwritten digits (2's). We embed a 1-dimensional manifold and show the images that are assigned to every 14th latent point. We can observe that neighbored digits are similar to each other, while digits that are dissimilar are further away from each other in latent space.

Figure 5.5 shows intermediate solutions of the optimization process of UNN$_g$ regression on 3D-S with $N = 600$. It shows intermediate solutions every 100 iterations, starting from (a) 100 embedded patterns to (f) the full embedding. The figures show that the assignment to different colors at different locations takes place from the very beginning. This process continues iteratively until all points are embedded. The complete embedding looks similar to the solution presented in Figure 5.3(b).

5.3.2 DSRE Comparison

In the following, we compare the DSRE achieved by both strategies with the initial DSRE and the DSRE achieved by LLE on all test problems. For the USPS *Digits* data set, we choose number '7'. Table 5.1 shows the experimental results of three settings for the neighborhood size K. The lowest DSRE on each problem is highlighted with bold figures. After application of the iterative strategies, the DSRE is significantly lower than initially. Increasing K results in higher DSRE. With exception of LLE with $K = 10$ on the 2D-S data set, the UNN strategy always achieves the best results. UNN achieves lower DSRE than UNN$_g$ with exception of 2D-S and $K = 10$. The win in accuracy of UNN over UNN$_g$ has to be paid with a constant runtime factor that plays an important role in case of large sets of high-dimensional data. UNN$_g$ and LLE perform similarly, i.e., UNN$_g$ is better than LLE in five cases, while LLE shows superior results in seven cases.

5.3.3 Sorting of Galaxies

In the following, we test the UNN variants on real-world data from astronomy, i.e., images of galaxies. Galaxies are massive, gravitationally bound systems of stars, gas and dust. Their numbers of stars typically varies in the range of 10^7 to 10^{14}. Edwin Hubble introduced a morphological classification scheme,

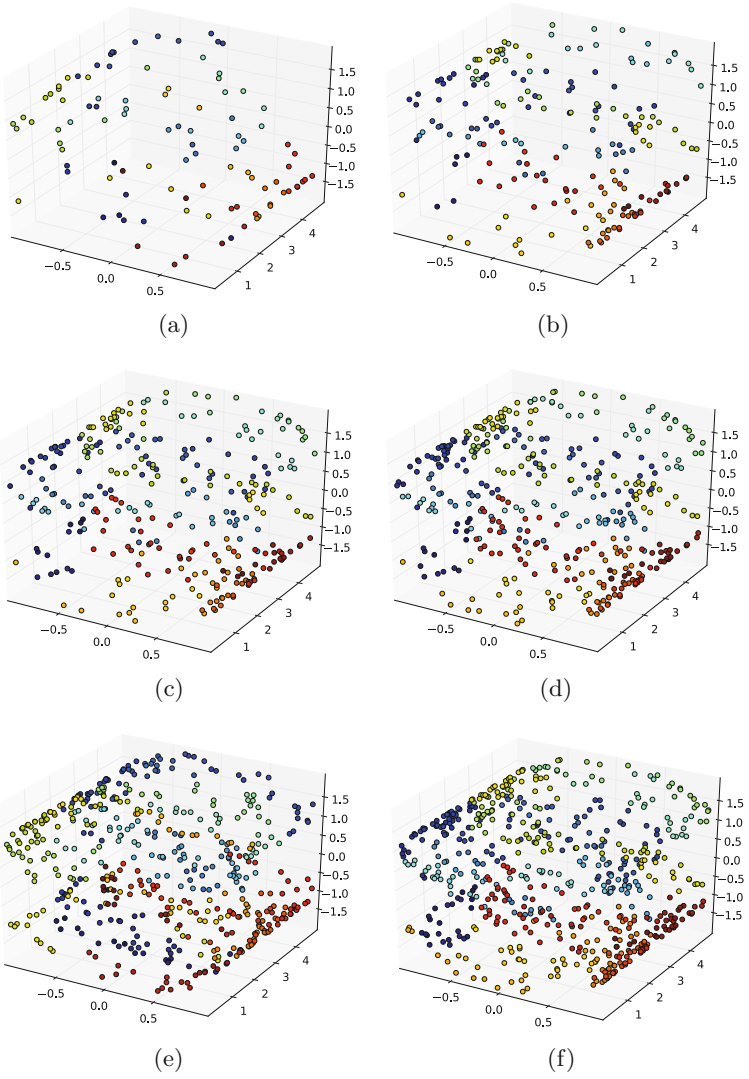


Fig. 5.5 Construction of 3D-S solution with UNN_g after (a) 100, (b) 200, (c) 300, (d) 400, (e) 500, and (f) 600 iterations

which became famous as Hubble sequence [43]. Neighbored classes in this diagram represent galaxies with similar shape and is thus similar to a topographic map. Hubble's classification scheme differentiates between three main classes: (1) elliptical galaxies, (2) spiral galaxies, and (3) lenticular galaxies,

Table 5.1 Comparison of DSRE for initial data set and after embedding with strategy UNN and UNN_g

	2D-S			3D-S		
K	2	5	10	2	5	10
init	201.6	290.0	309.2	691.3	904.5	945.80
UNN	**19.6**	**27.1**	66.3	**101.9**	**126.7**	**263.39**
UNN_g	29.2	70.1	64.7	140.4	244.4	296.5
LLE	25.5	37.7	**40.6**	135.0	514.3	583.6
	3D-S_h			*Digits* (7)		
K	2	5	10	2	5	10
init	577.0	727.6	810.7	196.6	248.2	265.2
UNN	**80.7**	**108.1**	**216.4**	**139.0**	**179.3**	**216.6**
UNN_g	101.8	204.4	346.8	145.3	195.4	222.1
LLE	94.9	198.9	387.4	147.8	198.1	217.8

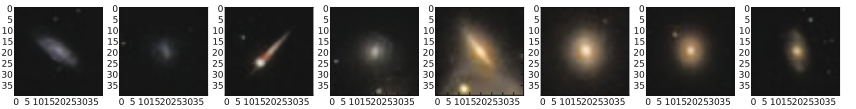


Fig. 5.6 Latent sorting of galaxies with UNN_g: galaxies of similar classes from the Hubble sequence are neighbored in latent space

see [50]. In our experiment, we employ images of galaxies from the *Sloan Digital Sky Survey (SDSS)*, a collection of millions of astronomical objects [2]. Figure 5.6 shows the UNN_g embedding of 100 images of galaxies from the SDSS database. Each image is a vector of 40×40 RGB values, i.e., the data space dimensionality is $d = 4,800$. The figure shows every 12th galaxy. We can observe that galaxies, which belong to one class according to Hubble's classification scheme, are neighbored on the low-dimensional manifold. Elliptical galaxies start from the left, while lenticular shapes are placed on the right hand side, a sorting that is similar to the Hubble taxonomy.

5.4 Robust ϵ-Insensitive Loss

Loss functions have an important part to play in machine learning, as they define the error and thus the design objective. In case of noisy data sets, overfitting effects may occur. It may be difficult to decide, if the curvature of the data manifold is substantial or noise. With the design of a loss function, the emphasis of outliers can be controlled. First, the residuals are computed. In case of unsupervised regression, the error is computed in two steps:

1. The distance function $\delta : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ maps the difference between the prediction $f(\mathbf{x})$ and the desired output value $\mathbf{y}$ to a value according to the distance w.r.t. a certain measure δ (e.g. the Minkowski metric).
2. The loss function $L : \mathbb{R} \to \mathbb{R}$ maps the residuals to the learning error. With the design of the loss function, the influence of residuals can be controlled.

The loss function should be chosen according to the requirements of the data mining model. Often, high residuals are penalized more than low residuals (e.g., with a quadratic loss function). We will concentrate on the ϵ-insensitive loss in the following. Let r be the residual. The L_1 loss is defined as $L_1(r) = \sum_{i=1}^{N} |r|$, while the L_2 loss quadratically penalizes with $L_2(r) = \sum_{i=1}^{N} r^2$. We will use the L_2 loss for the final DSRE study, but concentrate on the ϵ-insensitive loss L_ϵ during training of the UNN model. Loss L_ϵ is defined as

$$L_\epsilon(r) = \begin{cases} 0 & \text{if } |r| < \epsilon \\ |r| - \epsilon & \text{if } |r| \geq \epsilon, \end{cases} \tag{5.8}$$

and allows to ignore errors beyond a level of ϵ. Thus, L_ϵ avoids to overfit to curvatures of the data that may only be caused by noise effects.

5.5 Experimental Analysis of Robust Loss Functions

In the following, we experimentally analyze the influence of loss functions on the UNN embedding result. We evaluate the final embedding by (1) measuring the final L_2-based DSRE, (2) by visualizing the embedding with colored points, and (3) by showing the latent order of the high-dimensional patterns. We concentrate on two data sets, i.e., a 3D-S data set with noise and the *Digits* data set [46].

5.5.1 3D-S

In the first experiment, we analyze the behavior on the 3D-S data set. Noise is modeled by multiplication of each pattern of the 3D-S with a random value drawn from the Gaussian distribution[2]:

$$\mathbf{y}' \sim \mathcal{N}(1, \sigma) \cdot \mathbf{y}. \tag{5.9}$$

Table 5.2 shows the experimental results of UNN and UNN$_g$ concentrating on the ϵ-insensitive loss for $K = 5$ and Euclidean distance. The two left columns show the results on the data set without noise, i.e. $\sigma = 0.0$, the two right columns show the results for a noise magnitude of $\sigma = 5.0$.

[2] $\mathcal{N}(m, \sigma^2)$ represents a Gaussian distributed number with mean m and standard deviation σ.

Table 5.2 Influence of ϵ-insensitive loss on final DSRE of UNN on the 3D-S data set with and without noise

	$\sigma = 0.0$		$\sigma = 5.0$	
ϵ	UNN	UNN_g	UNN	UNN_g
0.2	**47.4**	77.4	79.1	85.6
0.4	48.1	77.4	79.3	85.6
0.6	51.8	76.3	78.7	85.6
0.8	50.9	76.3	**77.2**	84.4
1.0	64.0	76.4	79.4	84.2
2.0	96.0	68.3	119.6	82.0
3.0	138.4	**50.6**	163.7	**80.5**
4.0	139.1	50.6	168.8	82.1
5.0	139.1	50.6	169.0	83.2

At first, we concentrate on the experiments without noise. We can observe that the DSRE achieved by UNN is minimal for the lowest ϵ. Without noise for UNN ignoring residuals is disadvantageous: all intermediate positions are tested and a good local optimum can be reached. Further, for UNN_g lower DSRE values are achieved with increasing ϵ, to a limit of $\epsilon = 3.0$. We can conclude that local optima of UNN_g can be avoided by ignoring small residuals. The best DSRE of UNN_g is worse than the best of UNN.

For the experiments with noise of magnitude $\sigma = 5.0$, we can observe local DSRE minima: for $\epsilon = 0.8$ in case of UNN and $\epsilon = 3.0$ in case of UNN_g. For UNN, local optima caused by noise can be avoided by ignoring residuals. Furthermore, for UNN_g we observe the optimum at the same level of ϵ.

Figures 5.7(a) and 5.7(b) show a visualization of the embeddings of UNN and UNN_g without noise and the settings $\epsilon = 0.2$ and $\epsilon = 3.0$ corresponding to the settings of Table 5.2 shown in bold. Similar colors correspond to neighbored embeddings in latent space. The visualization shows that in both cases neighbored points in data space have similar colors, i.e., correspond to neighbored latent points. UNN achieves a lower DSRE than UNN_g. This is difficult to recognize in the visualization. Only some blue points in the embedding of UNN_g seem to be distributed in the upper and lower part of the S-structure, which may represent a local optimum. Figures 5.7(c) and 5.7(d) show the visualization of the UNN embeddings on the noisy 3D-S. The structure of the 3-dimensional S is obviously disturbed. However, neighbored parts in data space are assigned to similar colors. Again, the UNN embedding seems to be slightly better than the UNN_g embedding. Blue points can again be observed at different parts of the structure representing local optima.

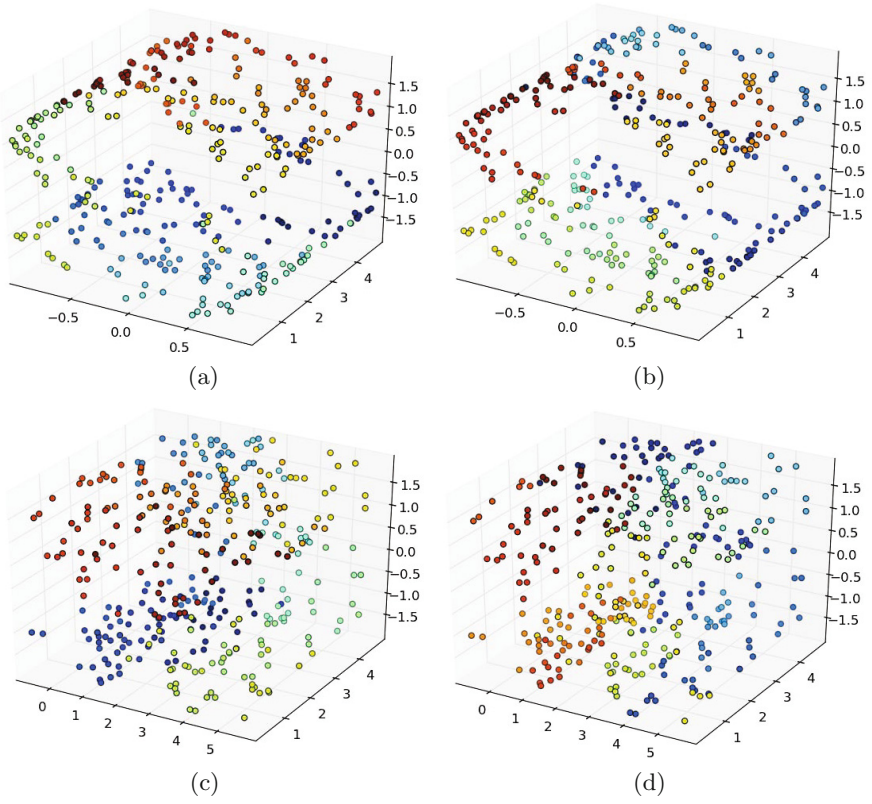


Fig. 5.7 Visualization of the best embeddings (lowest DSRE, bold values in Table 5.2) on the 3D-S data set of (a) UNN without noise, (b) UNN$_g$ without noise, (c) UNN with noise of magnitude $\sigma = 5.0$, and (d) UNN$_g$ with the same noise level

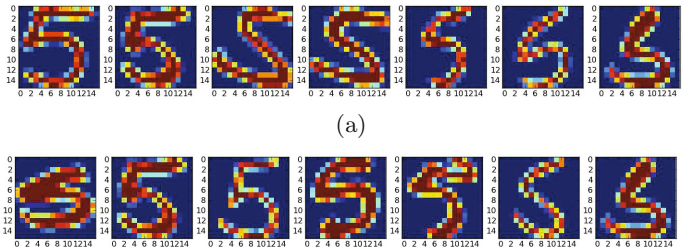
5.5.2 Handwritten Digits

To demonstrate the effect of the ϵ-insensitive loss for data spaces with higher dimensions, we employ the USPS handwritten *Digits* data set with $d = 256$. Table 5.3 shows the final DSRE w.r.t. the L_2-loss after training with the ϵ-insensitive loss with various parameterizations for ϵ. We used the setting $K = 10$ and $p = 10.0$ for the Minkowski metric. The results for *Digit* '5' show that a minimal DSRE has been achieved for $\epsilon = 3.0$ in case of UNN and $\epsilon = 5.0$ for UNN$_g$ with a minimum of $R = 429.75561$ was found for $\epsilon = 4.7$. Obviously, both methods can profit from the use of the ϵ-insensitive loss. For *Digit* '7' and UNN, ignoring small residuals does not improve the learning results, while UNN$_g$ with $\epsilon = 4.0$ achieves the best embedding.

Table 5.3 Influence of ϵ-insensitive loss on the final DSRE of UNN employing the *Digits* data set

	Digits (5)		*Digits* (7)	
ϵ	UNN	UNN_g	UNN	UNN_g
0.0	423.8	440.2	**225.4**	222.8
0.5	423.8	440.2	**225.4**	222.8
1.0	423.8	440.2	**225.4**	222.8
2.0	423.8	440.2	225.6	222.8
3.0	**423.5**	440.2	238.1	221.0
4.0	441.3	440.2	262.1	**218.2**
5.0	488.7	**432.3**	264.8	221.4
6.0	496.9	434.2	265.6	220.8

Figure 5.8 shows two UNN_g embeddings of the handwritten *Digits* data set, i.e., for $\epsilon = 2.0$ and for $\epsilon = 20.0$. For both settings, similar digits are neighbored in latent space. But we can observe that for $\epsilon = 20.0$, a larger variety of '5's in the data set is covered.



(a)

(b)

Fig. 5.8 Comparison of UNN_g embedding of '5's from the handwritten *Digits* data set. The figures show every 14th pattern of the latent sorting of 100 digits for (a) $\epsilon = 2.0$ and (b) $\epsilon = 20.0$.

5.6 Missing Data

In practical applications, data sets are often incomplete. Failures of sensors, matching of databases with disjunct feature sets or conditions, where data can get lost (e.g., in outer space due to X-ray) are typical examples for practical scenarios with incomplete patterns. However, it might be desirable to compute a latent embedding of incomplete high-dimensional data. Objective of this section is to introduce strategies that allow UNN to cope with missing data. The question arises, if the embedding approach can exploit useful structural information to reconstruct the missing entries.

The problem of missing data in feature vectors can be treated in various kinds of ways. Two strategies are usually employed to handle incomplete data sets:

- elimination of patterns with missing values and
- imputation methods that fill the gaps.

A simple method is to eliminate all patterns with missing entries. But this may introduce a bias, as the fact that features are missing can be caused by systematic errors. Further, the elimination of patterns from small data sets deteriorates the chance to learn good models. A class of methods to handle incomplete data are imputation methods, i.e., filling the gaps based on learning from complete patterns. Incomplete data can be filled based on simple statistical parameters like the median and the mean of the available entries. Unfortunately, this method may also introduce a bias. Other imputation methods are based on regression approaches, similar to the repair step we apply in this chapter. For this sake, a training set consisting of the complete patterns is used, missing entries are the missing labels that have to be predicted, while the entries of the complete patterns of the corresponding dimensions are employed as labels.

In case the distribution of missingness is conditionally independent of the missing values given the observed data, the data is called *missing at random*, i.e., entries are missing randomly with uniform distribution, in contrast to *missing not at random*, where dependencies exist. Schafer and Graham [97] have reviewed methods to handle them. In case of sparse data sets, joint densities can be computed in a probabilistic framework [32].

If possible, the method can directly deal with missing data. Our embed-and-repair method that will be introduced in Section 5.8 belongs to this class. For SVM classification, such an approach has been introduced by Chechik *et al.* [18]. It alters the SVM margin interpretation to directly deal with incomplete patterns. But the method is best suited for features that are absent than those that are not missing at random. An extension has been introduced by Dick *et al.* [22]. The approach by Williams *et al.* [112] employs logistic regression for classification of incomplete data and performs an analytic integration with an estimated conditional density function instead of imputation. The approach does not only take into account the complete patterns, but also the incomplete patterns in a semi-supervised kind of way.

5.7 Repair-and-Embed

Let $\mathbf{Y}$ be the matrix of high-dimensional patterns. In the missing data scenario, we assume that some patterns in $\mathbf{Y}$ are incomplete, i.e., it holds $\exists y'_{ij}$ with $y'_{ij} = n.a.$. Let $\tilde{\mathbf{Y}}$ be the matrix of complete patterns, i.e., it holds $\nexists y_{ij}$ with $y_{ij} = n.a.$. In the following, we illustrate the imputation for one missing entry, but the approach can easily be extended to the multiple case.

To complete y'_{ij}, repair-and-embed trains a regression model $\tilde{\mathbf{f}}$ based on $\tilde{\mathbf{Y}}$ and first fills the missing entries of patterns $[\mathbf{Y}]_j$ with a minimal number of missing entries. Let y_{ij} be the entry to complete. We can employ matrix $\tilde{\mathbf{Y}}^T_{-i}$ as training pattern matrix[3], while $\tilde{\mathbf{y}}_i = y_{i1}, \ldots, y_{i\tilde{N}}$ comprises the corresponding labels. Entry y'_{ij} is estimated with $\tilde{\mathbf{f}}$ leading to the complete vector $[\mathbf{Y}]_j = \tilde{\mathbf{y}}_j$ that can be embedded as usual with UNN. Algorithm 4 shows the pseudocode of repair-and-embed. As KNN regression is a non-parametric method, no training is necessary, only K has to be chosen carefully. After the pattern has been completed, the next pattern with minimal number of missing entries is chosen, and the process is repeated until all patterns are complete. Then, data set $\tilde{\mathbf{Y}}$ can be embedded as usual with UNN.

Algorithm 4. Repair-and-Embed

Require: $\mathbf{Y}$, K
1: **repeat**
2: choose $[\mathbf{Y}]_j$ with minimal number of missing entries
3: $\tilde{\mathbf{Y}}^T_{-i}$ is training pattern matrix
4: predict y_{ij} with KNN regression model $\tilde{\mathbf{f}}$
5: add y_{ij} to $\mathbf{Y}$, and set $\tilde{\mathbf{Y}} = [\tilde{\mathbf{Y}}, \mathbf{y}_j]$
6: **until** $\tilde{\mathbf{Y}}$ is complete
7: embed $\tilde{\mathbf{Y}}$ with UNN

5.8 Embed-and-Repair

The second variant for embedding incomplete data is based on embedding a vector $\mathbf{y}_j$ with a missing entry y_{ij} at dimension i ignoring the i-th component during the computation of the DSRE, i.e., minimizing

$$E_{-i}(\overline{\mathbf{X}}) = \frac{1}{N} \| \mathbf{f}_{\overline{\mathbf{X}}}(\overline{\mathbf{X}})_{-i} - \overline{\mathbf{Y}}_{-i} \|^2_F. \tag{5.10}$$

Algorithm 5 shows the pseudocode of the embed-and-repair approach. The algorithm starts iteratively with the vector $\mathbf{y}_j = [\mathbf{Y}]_j$ with increasing number of missing values. Starting the dimensionality reduction with complete patterns is reasonable to get as close as possible to the structure of the complete embedding. Embed-and-repair is a greedy approach that only considers the locally best embedding w.r.t. the available information. Embedded patterns can be completed to take part in the remaining embedding process. The gaps are closed with entries that ensure that the embedding is minimal w.r.t. $e_{\overline{\mathbf{X}}}(\mathbf{x})$. This is the average of the K-nearest points for dimension i, i.e., the nearest neighbors estimation $y_{ij} = \mathbf{f}_{\overline{\mathbf{X}}}(\mathbf{x}_j)_i$, see Equation 2.4.

[3] $\tilde{\mathbf{Y}}_{-i} = [(\mathbf{y}_j)_{-i}]_{j=1}^{\tilde{N}}$ with $(\mathbf{y}_j)_{-i} = (y)_j$ and $j = 1, \ldots, d$, $j \neq i$. $\tilde{N}$ is the number of complete patterns.

Algorithm 5. Embed-and-Repair

Require: $\mathbf{Y}$, K

repeat
  choose $\mathbf{y}_j = [\overline{\mathbf{Y}}]_j$ with minimal number of missing entries, y_{ij} is missing
  embed $\mathbf{y}_j$ with UNN minimizing $E_{-i}(\overline{\mathbf{X}}) \rightarrow \mathbf{x}_j$
  complete $\mathbf{y}_j$ with KNN based on $\overline{\mathbf{X}} \rightarrow \mathbf{y}_j$
  add $\mathbf{x}_j$ to $\overline{\mathbf{X}}$ with UNN
  $\overline{\mathbf{X}} = [\overline{\mathbf{X}}, \mathbf{x}_j]$, $\overline{\mathbf{Y}} = [\overline{\mathbf{Y}}, \mathbf{y}_j]$
until all patterns embedded

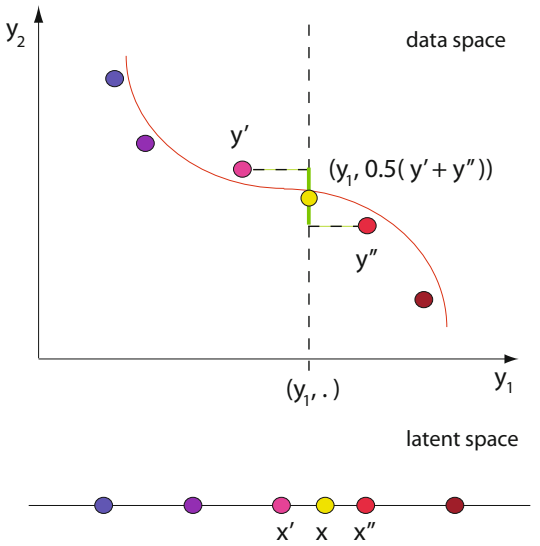


Fig. 5.9 Embed-and-repair. The incomplete pattern $\mathbf{y} = (y_1, \cdot)$ is embedded at position x leading to the lowest DSRE w.r.t. the first dimension, i.e., between x' and x''. Then, gap y_2 is filled with KNN and $K = 2$.

Figure 5.9 illustrates the embed-and-repair strategy for neighborhood size $K = 2$. Pattern $\mathbf{y} = (y_1, \cdot)$ is incomplete. It is embedded at the position, where it leads to the lowest DSRE w.r.t. the first dimension: between x' and x''. Then, the gap is filled with the mean of the second dimension of $\mathbf{y}'$ and $\mathbf{y}''$ yielding $\mathbf{y} = (y_1, 0.5 \cdot (y' + y''))$. The difference between KNN imputation and embed-and-repair imputation is that the embed-and-repair KNN prediction is based on neighborhoods in latent space. Hence, it is a dimensionality reduction-oriented imputation method based on characteristics introduced by UNN regression.

5.9 Experimental Analysis of Missing Data Methods

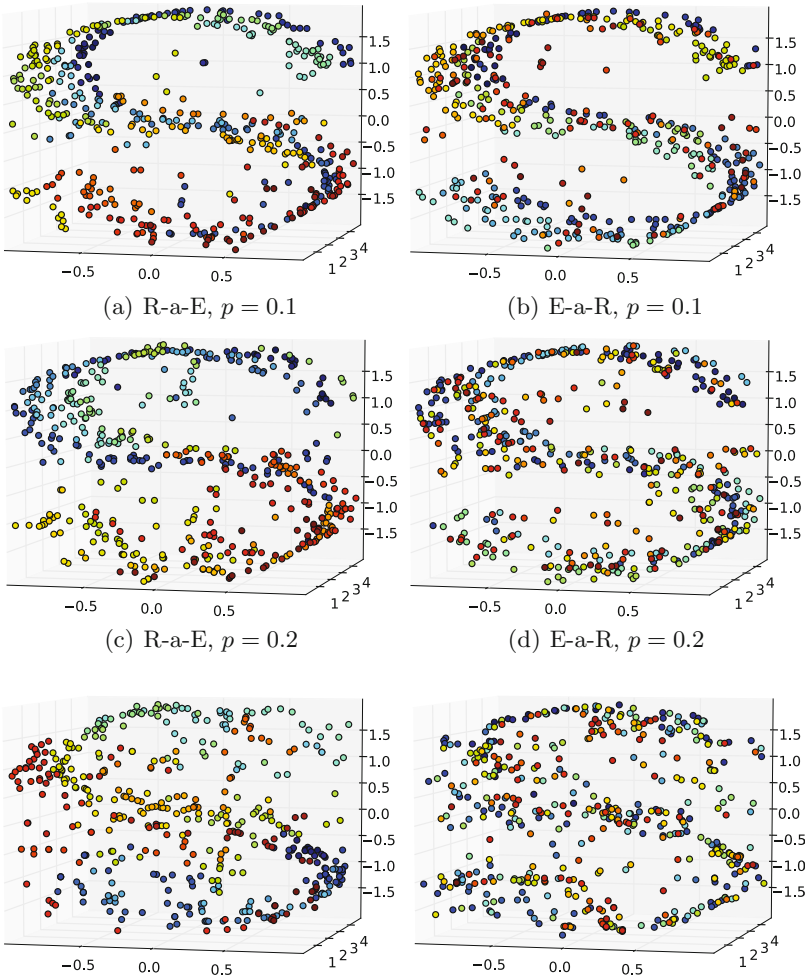
In the following, we describe the experimental setup for the comparison between both approaches. We generate a missing data matrix $\mathbf{Y}$ by removing entries y_{ij} from a complete data matrix $\mathbf{Y}^+ = [\mathbf{y}_j^+]_{j=1}^N \in \mathbb{R}^{d \times N}$ at random with uniform probability p, i.e., each entry y_{ij} is set to $n.a.$ with probability p. We experimentally compare the DSRE for embedding $\mathbf{Y}^+$ and $\mathbf{Y}$. Further, we analyze the imputation error $E_{imp} = \sum_{j=1}^N \|\mathbf{y}_j^+ - \tilde{\mathbf{y}}_j\|^2$, which is the deviation from the original complete patterns $\mathbf{y}_j^+$ and the repaired counterparts $\tilde{\mathbf{y}}_j$.

Table 5.4 shows the experimental results for increasing data missing rates p on the data set 3D-S. The experimental results show that UNN with repair-and-embed achieves the lowest DSRE on both data sets. The results are very close to the DSRE achieved on the data set without missing values (complete). UNN with embed-and-repair achieves the lowest imputation error E_{imp} in seven of the eight cases, but much worse results for the DSRE. While the DSRE results are still satisfactory with 1% of incomplete data, the approach fails for higher missing rates. Obviously, it is difficult to first determine the manifold structure from data sets with a high rate of missing values.

Figure 5.10 shows the embeddings of UNN on the 3D-S data set for increasing missing rates p. The left plots show the embeddings of repair-and-embed, the right plots the corresponding embeddings of embed-and-repair. The patterns are colorized w.r.t. their latent colors, i.e., patterns with similar colors are neighbored in latent space showing that the embedding has been successful. Figure 5.10(a) shows the embedding with a low missing rate of $p = 0.1$. The 3D-S is almost completely reconstructed, while the colors of the embedding show that a reasonable learning process took place. The higher the rate of incomplete data (cf. Figures (c) and (e)) the worse are the embeddings, i.e., different colors are neighbored. Higher missing rates can also be recognized by deviations from the S-structure.

Table 5.4 Comparison of imputation error E_{imp} and DSRE between UNN with repair-and-embed (R-a-E) and UNN with embed-and-repair (E-a-R) on 3D-S and 3D-S$_h$ w.r.t. increasing data missing rate p

		R-a-E		E-a-R		complete
data	p	E_{imp}	DSRE	E_{imp}	DSRE	UNN
3D-S	0.01	0.0507	**147.2**	**0.0269**	165.39	142.8
	0.1	0.3129	**143.8**	**0.2884**	265.2	142.8
	0.2	0.6454	**149.0**	**0.6146**	369.2	142.8
	0.3	0.9557	**152.7**	**0.9265**	452.3	142.8
3D-S$_h$	0.01	**0.0235**	**104.2**	0.0309	119.7	105.5
	0.1	0.2671	**101.9**	**0.2595**	217.6	105.5
	0.2	0.5509	**122.2**	**0.5007**	296.8	105.5
	0.3	0.8226	**129.5**	**0.5285**	301.8	105.5



(a) R-a-E, $p = 0.1$ (b) E-a-R, $p = 0.1$

(c) R-a-E, $p = 0.2$ (d) E-a-R, $p = 0.2$

(e) R-a-E, $p = 0.4$ (f) E-a-R, $p = 0.4$

Fig. 5.10 UNN embeddings of 3D-S with missing data for missing rates $p = 0.1, 0.2$, and $p = 0.4$ for repair-and-embed (left column) and for embed-and-repair (right column)

UNN with embed-and-repair shows a comparatively good embedding for the low missing rate of $p = 0.1$. But the colors show that the embeddings are worse for higher missing rates. For example, Figure 5.10(f) shows that the embedding of UNN with repair-and-embed and data missing rate $p = 0.4$ leads to comparably bad results, which is consistent with the results of Table 5.4.

5.10 Conclusions

With UNN regression, we have fitted a simple and fast regression technique into the unsupervised setting for dimensionality reduction. The two iterative UNN strategies are efficient methods to embed high-dimensional data into discrete 1-dimensional latent spaces taking $O(N^2)$ time. The speedup is achieved by restricting the number of possible solutions employing discrete latent positions and applying a fast iterative solution construction scheme. Both methods turned out to be efficient on artificial test problems in the experimental part. UNN achieves lower DSRE values than UNN_g. However, UNN_g is faster because of the multiplicative constants of UNN. In practical scenarios, data sets often suffer from noise and missing entries. We have introduced strategies to make UNN more robust. The ϵ-insensitive loss has been employed for noisy patterns and has experimentally shown to yield better embeddings for properly chosen values of ϵ. Besides the presence of noise, incomplete data is a frequent problem. We introduced an iterative variant that takes into account the predicted values for completion of entries of patterns with an increasing data missing rate. First embedding patterns at locations with the lowest DSRE and then repairing the entries employing the neighbors in latent space is an approach that makes use of the intrinsic structure UNN regression assumes for imputation. This leads to comparatively good pattern reconstructions, but worse embeddings than UNN with repair-and-embed.

6

Metaheuristics

6.1 Introduction

The UNN optimization problem is particularly difficult to solve due to local optima. The two iterative strategies UNN and UNN$_g$ of the last chapters allow the fast construction of a solution. In this chapter, we want to answer the question, if *exhaustive* evolutionary search allows to come closer to the optimal embedding. We compare a discrete evolutionary approach based on stochastic swaps to a continuous evolutionary variant that is based on evolution strategies, i.e., the covariance matrix adaptation variant CMA-ES. The continuous variant is the first step to embeddings into continuous latent spaces. The evolutionary optimization process requires to model a candidate solution, which is a matrix of latent vectors $\mathbf{X} = [\mathbf{x}_i]_{i=1}^N \in \mathbb{R}^{q \times N}$. For one fitness evaluation $\overline{f}(\mathbf{x})$, the DSRE of the complete embedding is computed (cf. Equation 5.4). For the continuous variant, two kinds of regularization approaches will be compared: restriction of latent space to a hypercube $[0, 1]^q$ and penalizing extension with a regularization term $\lambda \|\mathbf{X}\|_F^2$.

In many applications, it might be desirable to embed high-dimensional patterns into free continuous latent spaces. The CMA-ES approach is the first step into this direction. We address the problem of an increasing solution space sizes, when increasing the number of patterns with an iterative swarm-inspired approach [59]. Swarm intelligence describes self-organizing processes of groups of individuals in nature and in artificial systems. Particle swarm optimization (PSO) and ant colony optimization are the two most famous and successful algorithms in this research field. A swarm approach for unsupervised nearest neighbors is presented that allows free embeddings of patterns in latent spaces of arbitrary dimensionality, i.e., $1 \leq q < d$. The approach combines the iterative construction of solutions, similar to the previously introduced UNN algorithms, with PSO-like search steps. In the experimental part, we show that the swarm approach is an effective embedding strategy.

O. Kramer: *Dimensionality Reduction with Unsupervised Nearest Neighb.,* ISRL 51, pp. 75–91.
DOI: 10.1007/978-3-642-38652-7_6

6.2 Evolutionary Algorithms

Evolutionary search is based on a population $\mathcal{P} = \{\mathbf{x}_1, \ldots, \mathbf{x}_\lambda\}$ of λ candidate solutions called individuals that are approximations of a problem solution [64]. In continuous solution spaces, the individuals are often real-valued vectors $\mathbf{x} \in \mathbb{R}^q$. The population is iteratively subject to random changes with recombination, mutation, and selection of the best candidate solutions. Algorithm 6 shows the pseudocode of a general evolutionary algorithm. After initialization and fitness evaluation of the first population $\mathcal{P}$, the optimization process repeats three steps: (1) the recombination operator selects ρ parents and combines their parts to λ new solutions, (2) the mutation operator adds random noise to the λ preliminary candidate solutions. Their quality is called fitness $\overline{f}(\mathbf{x})$. The fitness of new offspring solutions is evaluated. All individuals of a generation are put into population $\mathcal{P}'$. In Step (3), μ individuals are selected from $\mathcal{P}'$ and form the new parental population $\mathcal{P}$ of the following generation. The process is repeated until a termination condition is fulfilled. Typical termination conditions are that a certain solution quality or an upper bound on the number of generations is reached.

Algorithm 6. Evolutionary Algorithm

Require: $\overline{f}, \mu, \lambda, \rho$
1: initialize solutions $\mathbf{x}_i$ of population $\mathcal{P}$
2: evaluate fitness $\overline{f}(\mathbf{x}_i)$ of solutions in $\mathcal{P}$
3: **repeat**
4: **for** $i = 1$ **to** λ **do**
5: select ρ parents from $\mathcal{P}$
6: create $\mathbf{x}_i$ by recombination
7: mutate $\mathbf{x}_i$
8: evaluate $\overline{f}(\mathbf{x}_i)$
9: add $\mathbf{x}_i$ to $\mathcal{P}'$
10: **end for**
11: select μ parents from $\mathcal{P}' \rightarrow \mathcal{P}$
12: **until** termination condition

In the following, we will give a short introduction to evolutionary optimization methods and go deeper into evolution strategies that have proven well in practical optimization scenarios. The genetic operators intermediate and dominant recombination as well as Gaussian mutation are introduced.

6.2.1 Recombination

In biological systems, recombination (also known as crossover) mixes the genetic material of two parents. Most evolutionary algorithms use a recombination operator and combine the information of two or more individuals

$\mathbf{x}_1, \ldots, \mathbf{x}_\rho$ to a new offspring solution. Hence, the offspring carries parts of the genetic material of its parents. Many recombination operators are restricted to two parents, but multi-parent recombination variants have been presented in the past that combine information of ρ parents. The use of recombination is discussed controversially within the building block hypothesis by Goldberg [35, 42] and the genetic repair effect by Beyer [10].

Typical recombination operators in evolution strategies are dominant and intermediate recombination. Dominant recombination randomly combines the genes of all parents and is applicable to continuous and discrete representations. If we consider parents of the form $\mathbf{x}_k = ((x_1)_k, \ldots, (x_q)_k)^T$ with $1 \leq k \leq \rho$, dominant recombination creates the offspring vector $\mathbf{x}' = (x'_1, \ldots, x'_q)^T$ by choosing the i-th component x'_i at random

$$x'_i = (x_i)_k, \quad k \in \text{ random } \{1, \ldots, \rho\}. \tag{6.1}$$

Intermediate recombination is appropriate for integer and real-valued solution spaces. Given ρ parents $\mathbf{x}_1, \ldots, \mathbf{x}_\rho$ each component of the offspring vector $\mathbf{x}'$ is the arithmetic mean of the components of all ρ parents. Thus, an offspring solution carries the characteristics of its parents

$$x'_i = \frac{1}{\rho} \sum_{k=1}^{\rho} (x_i)_k. \tag{6.2}$$

Integer representations may require rounding procedures for valid solutions.

6.2.2 Mutation

Mutation is the main source of evolutionary changes. According to Beyer and Schwefel [11], a mutation operator is supposed to fulfill three conditions. First, from each point in the solution space each other point must be reachable. Second, in unconstrained solution spaces a bias is disadvantageous, because the direction to the optimum is unknown, and third, the mutation strength should be adjustable, in order to adapt exploration and exploitation to local solution space conditions.

In the following, we concentrate on the famous Gaussian mutation operator for optimization in $\mathbb{R}^N$. Solutions are vectors of real values $\mathbf{x} = (x_1, \ldots, x_N)^T \in \mathbb{R}^N$. Random numbers based on the Gaussian distribution $\mathcal{N}(0, 1)$ fulfill these conditions in continuous domains[1]. With the Gaussian distribution, many natural and artificial processes can be described. The idea is to mutate each individual applying the mutation operator

$$\mathbf{x}' = \mathbf{x} + \mathbf{z}, \tag{6.3}$$

[1] $\mathcal{N}(m, \sigma^2)$ represents a randomly drawn Gaussian distributed number with expectation value m and standard deviation σ.

with a mutation vector $\mathbf{z} \in \mathbb{R}^N$ based on sampling from the Gaussian distribution

$$\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I}) = (\mathcal{N}(0, \sigma^2), \ldots, \mathcal{N}(0, \sigma^2))^T \sim \sigma\mathcal{N}(\mathbf{0}, \mathbf{I}) \tag{6.4}$$

with identity matrix $\mathbf{I}$. The standard deviation σ plays the role of the mutation strength and is also known as step size. The isotropic Gaussian mutation with only one step size uses the same standard deviation for each component x_i. Of course, the step size σ can be kept constant, but the convergence to the optimum can be improved by adapting σ according to local solution space characteristics. In case of high success rates, i.e., a high number of offspring solutions being better than their parents, big step sizes are advantageous, in order to explore the solution space as fast as possible. This is often reasonable at the beginning of the search. In case of low success rates, smaller step sizes are appropriate. This is often adequate in later phases of the search during convergence to the optimum, i.e., when approximating solutions should not be destroyed. An example for an adaptive control of step sizes is the 1/5-th success rule by Rechenberg [90] that increases the step sizes, if the success rate is over 1/5-th, and decreases it, if the success rate is lower.

6.2.3 Selection

The counterpart of the variation operators mutation and recombination is selection. Selection gives the evolutionary search a direction. Based on their fitness, a subset of the population is selected, while the rest is rejected. The elitist selection strategies comma and plus selection choose the μ best solutions and are usually applied for survivor selection. Both operators can easily be implemented by sorting the population with respect to the individuals' fitness. Plus selection (written as $(\mu + \lambda)$-EA) selects the μ best solutions from the union $\mathcal{P} \cup \mathcal{P}'$ of the last parental population $\mathcal{P}$ and the current offspring population $\mathcal{P}'$. In contrast, comma selection (written as (μ, λ)-EA) selects exclusively from the offspring population neglecting the parental population, even if individuals have a superior fitness. Forgetting superior solutions may seem to be disadvantageous. But good solutions may be only local optima within a certain vicinity, and the evolutionary process may fail to leave them without the ability to forget.

6.2.4 CMA-ES

As optimization approach, we employ the CMA-ES by Hansen and Ostermeier [86]. The CMA-ES belongs to the class of evolution strategies [11]. In each generation, a population of λ points $\mathbf{x}_i$, $i = 1, \ldots, \lambda$ is produced with the multivariate normal distribution

$$\mathbf{x}_i = \mathbf{m} + \mathbf{z} \tag{6.5}$$

for $i = 1, \ldots, \lambda$ with $\mathbf{m}$ defining the mean of the Gaussian distribution generated by the CMA-ES and mutation

$$\mathbf{z} \sim \sigma \cdot \mathcal{N}_i(0, \mathbf{C}) \tag{6.6}$$

with σ defining the step sizes and with covariance matrix $\mathbf{C}$. Individuals are ranked according to their fitness $\overline{f}$

$$\overline{f}(\mathbf{x}_{1:\lambda}) \leq \ldots \leq \overline{f}(\mathbf{x}_{\mu:\lambda}) \leq \ldots \leq \overline{f}(\mathbf{x}_{\lambda:\lambda}), \tag{6.7}$$

following the notation that $\mathbf{x}_{i:\lambda}$ is the i-th best of λ individuals. The mean $\mathbf{m}$ is updated with the μ best solutions in each generation

$$\mathbf{m} = \sum_{i=1}^{\mu} \omega_i \mathbf{x}_{i:\lambda} \tag{6.8}$$

with positive and normalized weights ω_i. Core of the CMA-ES is the update of covariance matrix $\mathbf{C}$, which is adapted to local fitness conditions. The update rules for $\mathbf{C}$, $\mathbf{m}$, and σ can be found in Hansen *et al.* [86]. The population sizes are chosen as $\lambda = 4N$ and $\mu = \lambda/2$.

6.3 Combinatorial Perspective

First, we analyze the combinatorial problem formulation based on embeddings on a lattice structure. The number of possible K-nearest neighborhoods and consequently of different fitness values is $\binom{N}{K}$. For large N, this is still an intractable number of solutions and makes it impractical to test all possible latent variable neighborhoods. In the following, we present an evolutionary discrete search strategy that randomly selects two points on a latent grid and swaps their positions, see Algorithm 7 for the corresponding pseudocode. The candidate solution is an assignment of patterns to latent positions on a latent lattice.

Algorithm 7. $(1+1)$-EA FOR UNN

Require: $\mathbf{Y}$
1: initialization $\mathbf{X} = [\mathbf{x}_i = i]_{i=1}^{N}$
2: **repeat**
3: choose latent points $\mathbf{x}_i, \mathbf{x}_j$ with $i, j \in \{1, \ldots, N\}$ with $i \neq j$
4: change $\mathbf{X}$ to $\mathbf{X}'$ by swapping $\mathbf{x}_i$ and $\mathbf{x}_j$
5: replace $\mathbf{X}$ by $\mathbf{X}'$ if $E(\mathbf{X}') \leq E(\mathbf{X})$
6: **until** termination condition
7: **return** embedding $\mathbf{X}$

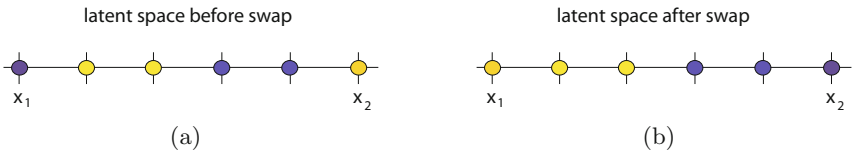


Fig. 6.1 Grid in latent space with $q = 1$ (a) before and (b) after a swap that leads to a lower DSRE with $K = 2$. Similar colors correspond to low distances in data space.

Let $\mathbf{X} = [\mathbf{x}_i = i]_{i=1}^{N}$ be an initial solution, i.e., an initial assignment of the high-dimensional patterns $\mathbf{y}_i$ to positions $\mathbf{x}_i$, $i = 1, \ldots, N$ in latent space. The $(1+1)$-EA randomly selects two positions $\mathbf{x}_i, \mathbf{x}_j$, with $i, j \in \{1, \ldots, N\}$ with $i \neq j$ and swaps them, iff the DSRE is decreased (see Line 5). Otherwise, the swap is rejected, and the latent points are reset to their original positions. The search terminates, if the DSRE does not change for κ iterations. Figure 6.1 shows a 1-dimensional example grid. Assume the left blue latent point $\mathbf{x}_1$ and the right yellow latent point $\mathbf{x}_2$ have randomly been selected to swap their positions. The DSRE is computed for the old and the novel neighborhood. The swap is accepted, as similar patterns in data space, illustrated by the same colors, are neighbored in latent space.

If we define the swap process as mutation operator $\mathrm{SWP}(\mathbf{x})$, we can state a weak convergence result for a slightly modified $(1+1)^*$-EA that selects the worse solution with a positive probability. The proof based on Rudolph's Markov chain results for evolutionary algorithms [95] is sketched in the following. The $(1+1)^*$-EA with swap mutation (SWP) converges with probability 1 to the global optimum after a finite number of iterations. To show this, we have to ensure that assumptions (A_1) to (A_3) of Theorem 5.4 are valid [95]. The recombination condition (A_1) must guarantee that there is a positive probability to take a solution $\mathbf{x}_t$ into the offspring population with the recombination operator. As the $(1+1)^*$-EA does not use recombination, the solution $\mathbf{x}_t$ is used for mutation in iteration t with probability 1. Assumption (A_2) must guarantee that for every pair $(\mathbf{x}, \tilde{\mathbf{x}}_t)$ of candidates in solution space, there exists a finite path $\mathbf{x} = \mathbf{x}_1, \mathbf{x}_2, \ldots, \tilde{\mathbf{x}}_t = \mathbf{x}_l$ of pairwise distinct solutions the evolutionary process can mutate to with a positive probability. The probability to choose indices i, j is lower bounded by $1/N^2$. Hence, the probability to mutate from $\mathbf{x}_1$ to $\mathbf{x}_l$ is lower bounded by $1/N^{2(l-1)} \geq 0$. Last, for all candidate solutions $\mathbf{x}_t$, there is a positive probability of being selected into the following generation.

6.4 Continuous Perspective

The continuous perspective allows an infinite number of possible latent positions, which are only restricted by the regularization mechanism. A candidate solution is represented as matrix $\mathbf{X} = [\mathbf{x}_i]_{i=1}^N \in \mathbb{R}^{q \times N}$ of latent positions. It is a vector of scalars for $q = 1$. For one fitness evaluation $\overline{f}(\mathbf{x})$, the DSRE of the complete embedding (cf. Equation 5.4) is computed. Similar to UKR, we regularize the continuous UNN model to avoid complex models and overfitting. An unconstrained UNN regression formulation would allow the latent points to move infinitely apart from each other, although only a restricted number of neighborhoods is possible. From a practical optimization perspective, a restriction of the search space is recommendable to avoid an infinite number of redundant solutions. Two kinds of regularization approaches will be compared: restriction of latent space to a hypercube $[0, 1]^q$ and penalizing extension with a summand $\lambda \|\mathbf{X}\|_F^2$.

6.4.1 Restriction of Latent Space

First, we restrict the latent space to the unit hypercube $\mathbf{x} \in [0, 1]^q$, and the optimization problem becomes

$$\min \overline{f}(\mathbf{X}) = E(\mathbf{X}) \text{ subject to } x_{ij} \in [0, 1]. \tag{6.9}$$

The constraint forces the latent points to stay in the unit hypercube in latent space. To handle the interval constraint, we penalize deviations from the interval employing a quadratic penalty

$$p(\mathbf{X}) = \sum_{i,j} \epsilon_{ij} \text{ with } \epsilon = \begin{cases} (x_{ij} - 1)^2 & \text{if } x_{ij} > 1 \\ x_{ij}^2 & \text{if } x_{ij} < 0 \\ 0 & \text{else} \end{cases}. \tag{6.10}$$

In Section 6.5, the latent space restriction approach will be compared to the penalty approach experimentally.

6.4.2 Penalizing Extension

The second and very related variant for regularization of UNN we employ is penalizing extension in latent space. The optimization problem becomes

$$\min \overline{f}(\mathbf{X}) = E(\mathbf{X}) + \lambda \|\mathbf{X}\|_F^2. \tag{6.11}$$

The penalty limits the sum of lengths of all latent variables. This technique has already been introduced for regularization of UKR models (cf. Section 4.21).

To understand the influence of parameter λ, we test various settings for regularization parameter λ and two neighborhood sizes K on the 3D-S data

Table 6.1 Analysis of regularization parameter λ for two neighborhood sizes $K = 2$ and $K = 10$ on the 3D-S data set with two data set sizes $N = 30$ and 50

N	K	init	10^{-2}	10^{-1}	1.0	10^{1}	10^{2}
30	2	34.82	**18.64**	21.76	32.06	36.23	36.25
30	10	51.39	**36.99**	38.95	51.33	51.14	50.33
50	2	60.50	**39.51**	52.01	56.01	63.63	58.08
50	10	86.17	**67.44**	75.14	81.19	84.91	82.10

set. We repeat each experiment 25 times. Table 6.1 shows the median DSRE of the experiments. The CMA-ES terminates after $1,000$ generations. It can be observed that the DSRE can be reduced significantly in comparison to the initial state for small settings of λ. For large settings, the optimization process first primarily concentrates on reduction of extension and neglects optimization of the DSRE in the $1,000$ generations the CMA-ES employs. Hence, we set $\lambda = 10^{-2}$ in the following experimental analyses.

6.5 Experimental Analysis of Evolutionary Variants

In the following, we analyze the evolutionary variants experimentally concentrating on a comparison between the introduced variants and an analysis of the problem dimensionality.

6.5.1 Comparison

Table 6.2 shows the experimental results of 25 CMA-ES runs on the data sets 3D-S and 3D-S_h with $N = 30$ patterns. The figures show the resulting DSRE at the beginning (init), in comparison to UNN_g, the CMA-ES with both regularization strategies and the $(1 + 1)$-EA that works on lattice representations. For comparison to one of the state-of-the-art methods in dimensionality reduction, we employ LLE [94].

The experimental results show that UNN_g achieves a lower DSRE than LLE for $K \geq 5$. This result shows the strength of the iterative heuristic. Concerning the evolutionary optimization approaches, we can observe that the (1+1)-EA achieves a lower DSRE than UNN_g. The continuous approaches and UNN_g achieve similar results: On 3D-S with $K = 2$, the penalized variant, for $K = 5$ the constrained variant and for $K = 10$ both variants achieve a lower DSRE than UNN_g. On 3D-S_h, UNN_g, and the continuous variants achieve similar results. The evolutionary variants have a budget of $1,000$ fitness evaluations, i.e., $1,000N$ complete DSRE evaluations ($30,000$ for $N = 30$), while UNN_g solves the problem with a budget of $0.5N \cdot (N + 1)$ DSRE computations (465 for $N = 30$). At the same time, the $(1 + 1)$-EA shows

Table 6.2 Comparison between UNN_g, the evolutionary approaches, and LLE for $q = 1$ w.r.t. DSRE

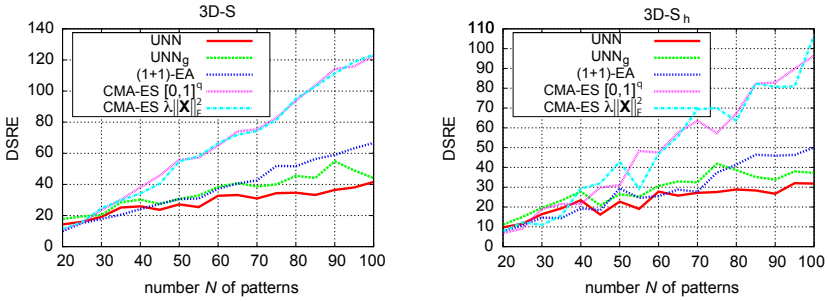
	3D-S			3D-S_h		
K	2	5	10	2	5	10
init	34.8 ±0.0	46.8 ±0.0	51.3 ±0.0	28.3 ±0.0	40.5 ±0.0	41.2 ±0.0
UNN_g	23.4 ±0.0	31.3 ±0.0	43.3 ±0.0	15.6 ±0.0	20.8 ±0.0	28.3 ±0.0
CMA, $[0, 1]^q$	24.5 ±11.6	27.6 ±8.4	36.3 ±15.0	24.0 ±11.7	20.4 ±20.4	29.1 ±15.4
CMA, $\lambda\|\mathbf{X}\|_F^2$	22.2 ± 6.1	31.6 ± 10.8	41.4 ± 8.3	14.7 ± 10.9	24.5 ± 11.8	33.8 ± 21.3
$(1 + 1)$-EA	$\mathbf{13.3} \pm 1.3$	$\mathbf{24.4} \pm 2.5$	$\mathbf{31.1} \pm 1.7$	10.8 ± 0.4	$\mathbf{17.6} \pm 1.4$	$\mathbf{24.7} \pm 0.4$
LLE	13.7 ±0.0	34.1 ±0.0	49.6 ±0.0	**10.4** ±0.0	23.6 ±0.0	31.2 ±0.0

better results than the CMA-ES. On both data sets, the restricted CMA-ES shows better results than the penalized CMA-ES in two of the three cases. But due to the high standard deviations, a statistical significance cannot be reported. In contrast, the $(1 + 1)$-EA is quite robust with small standard deviations.

6.5.2 Curse of Dimensionality

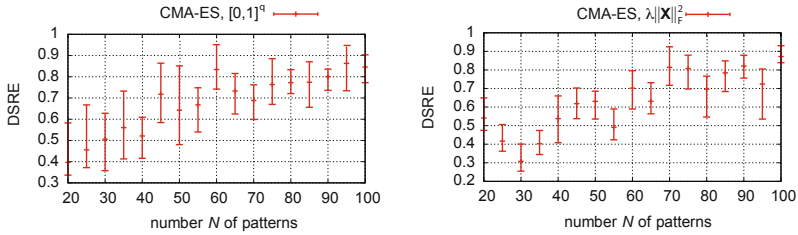
The number of patterns defines the dimensionality of the UNN optimization problem. The question arises, how the dimensionality influences the success of the evolutionary optimizers. We try to answer this question in the following by showing the fitness development depending on the number of patterns. Figure 6.2(a) shows the DSRE for UNN, UNN_g, both CMA variants, and the $(1 + 1)$-EA on the 3D-S data set (without hole). We can observe that the stochastic variants are slightly better at the beginning of the optimization process, but fail for higher dimensions. The continuous variants do not scale well, while the $(1 + 1)$-EA is still able to approximate the optimum. But the heuristics UNN and UNN_g scale much better, in particular UNN turns out to be the best optimizer for larger problem sizes. Figure 6.2(b) shows the corresponding outcome of the experiments for the 3D-S_h data set with hole, where similar observations can be made. Here, it is interesting that the two continuous variants show a very similar behavior.

For these two cases, we show the relative fitness improvement with variances in Figure 6.3. For the 3D-S_h data set with hole, the mean relative DSRE improvement E_s/E_e is shown, with E_s being the DSRE at the start (initialization) and the DSRE E_e in the end (after termination). The relative error is plotted with variance for the constrained (left) and the penalized (right) CMA-ES-based optimization. The relative improvement gets worse with increasing problem size. We can observe that the worst runs on larger problem sizes (as of $N = 50$) show very bad results, with almost no improvement. The constrained variant shows higher variances for small data sets.



(a) (b)

Fig. 6.2 Curse of dimensionality for evolutionary UNN variants on (a) 3D-S and (b) 3D-S$_h$



(a) (b)

Fig. 6.3 Relative error E_s/E_e depending on the number of patterns for the constrained CMA-ES ($[0,1]^q$, left) and the regularized ($\lambda\|\mathbf{X}\|_F^2$, right) on the 3D-S$_h$ data set

6.6 Swarm Intelligence

In nature, systems can be observed, in which large numbers of comparatively simple units organize in groups. This form of collective organization is known as swarm intelligence. The disadvantage of simple behaviors is compensated by the large number of acting units and their massive parallelism. Swarms consist of a large number of simple entities that cooperate to act goal-oriented. Besides the effect of masses, two basic principles are the driving forces of the problem solving capabilities of swarms:

- Emergence: Goal-directed behavior is achieved in an emergent kind of way. The group of individuals allows the swarm to show behaviors that are more sophisticated than the abilities of a single individual. Learning of collaboration and the division of tasks into subtasks belong to the most important principles of swarm intelligence.
- Stigmergy: The environment is used as memory and as means of communication between individuals of the swarm. The individuals, also called particles, usually have a very restricted memory. The information

exchange is based on elements in the environment, e.g., on pheromones in the case of ants. Pheromones allow to encode location- and time-relevant information using surfaces to store their traces.

With these two principles, swarms are able to solve comparatively complex tasks. Ant colony optimization algorithms [24] belong to the most prominent variants of swarm algorithms, mostly for solving combinatorial optimization tasks. For the dimensionality reduction method, we use an approach based on PSO mechanisms.

6.6.1 Particle Swarm Optimization

Similar to evolutionary algorithms, PSO is a population approach with stochastic components. Introduced by Kennedy and Eberhart [49], it is inspired by the movement of natural swarms and flocks. The algorithm utilizes μ particles with a position $\mathbf{x}$ that corresponds to the objective variables and a velocity $\mathbf{v}$, which is similar to the mutation strengths in evolutionary computation. The principle of PSO is based on the idea that the particles move in solution space, influencing each other with stochastic changes, while previous successful solutions act as attractors.

Algorithm 8. Particle Swarm Optimization

Require: κ
1: initialize particles $\mathbf{x}_1, \ldots, \mathbf{x}_\mu$
2: **repeat**
3: **for** $i = 1$ **to** μ **do**
4: compute $\tilde{\mathbf{x}}$ and $\mathbf{x}^*$
5: update velocity $\hat{\mathbf{v}}$
6: update position $\hat{\mathbf{x}}$
7: compute fitness $\overline{f}(\hat{\mathbf{x}})$
8: **end for**
9: **until** termination condition

In each iteration, the position of particle $\mathbf{x}$ is updated by adding a velocity $\hat{\mathbf{v}}$

$$\hat{\mathbf{x}} = \mathbf{x} + \hat{\mathbf{v}}, \tag{6.12}$$

which is updated as follows

$$\hat{\mathbf{v}} = \mathbf{v} + c_1 r_1 (\tilde{\mathbf{x}} - \mathbf{x}) + c_2 r_2 (\mathbf{x}^* - \mathbf{x}), \tag{6.13}$$

where $\tilde{\mathbf{x}}$ is the best previous position of the particle and $\mathbf{x}^*$ is the best position of the swarm. The weights $c_1, c_2 \in [0, 1]$ are acceleration coefficients that determine the bias of the particle towards its own and the swarm history and can be used to control exploitation and exploration of the search space. The recommendation given by Kennedy and Eberhart is to set both parameters to 0.5 [49]. The stochastic components r_1 and r_2 are uniformly drawn at random from the interval $[0, 1]$.

6.6.2 Swarm-Based Dimensionality Reduction

Various methods for PSO- and ant colony optimization-based clustering have been presented. A survey can be found in the book by Abraham *et al.* [3]. Kao and Cheng [48] have introduced an ant colony optimization algorithm for clustering that employs pheromones and distances between elements as heuristic clustering information. The combination of population-based search and stochastic elements allows to overcome local optima and find optimal clustering results. Further methods for swarm-based clustering can be found in the book by Abraham *et al.* [3]. O'Neill and Brabazon [85] have introduced a hybrid approach of PSO and SOMs by Kohonen [54] that control the weights of the map employing a PSO-similar update rule. Also ant colony optimization has been employed to improve the topographic SOM mapping [41].

6.7 Iterative Particle Swarm Embeddings

There are two reasons to employ a metaheuristic search method to solve the UNN optimization problem. First, the optimization problem is highly multimodal (cf. Chapter 7), second, $E(\mathbf{X})$ is not differentiable due to the employment of KNN.

6.7.1 PSEA Approach

As a consequence of the difficult optimization problem, the PSO metaheuristic is employed. The particle swarm embedding algorithm (PSEA) is based on the following two ideas:

1. *Iteratively* construct an embedding $\mathbf{X}$ to cope with the large number of free parameters, and
2. perform PSO-like blackbox search steps in each iteration to place each latent point at an optimal position.

The iterative procedure turned out to be a fruitful part of the discrete latent approaches of UNN in the previous chapters. From this perspective, the combination with a continuous heuristic operator seems to be a natural enhancement. The approach is described in the following, see Algorithm 9.

The first pattern $\mathbf{y}_1$ is embedded at an arbitrary initial position, e.g., $\overline{\mathbf{X}} = [\mathbf{0}]$, and the iterative pattern matrix is $\overline{\mathbf{Y}} = [\mathbf{y}_1]$. Let $\mathbf{y}_1, \ldots, \mathbf{y}_n$ be the sequence of already considered patterns with associated embeddings $\mathbf{x}_1, \ldots, \mathbf{x}_n$. For the next pattern $\mathbf{y}_i$ with $i = n + 1 \leq N$, the PSEA seeks the optimal position $\mathbf{x}$ to embed the particle. A loop of PSO-like steps is repeated for κ iterations

$$\hat{\mathbf{x}} = \mathbf{x} + \hat{\mathbf{v}} \tag{6.14}$$

with velocity

$$\hat{\mathbf{v}} = \mathbf{v} + c_1 r_1 (\tilde{\mathbf{x}} - \mathbf{x}) + c_2 r_2 (\mathbf{x}^* - \mathbf{x}). \tag{6.15}$$

Algorithm 9. Particle Swarm Embedding Algorithm

Require: $\mathbf{Y}$, K, κ
1: $\overline{\mathbf{X}} = [\mathbf{0}]$, $\overline{\mathbf{Y}} = [\mathbf{y}_1]$
2: **for** $i = 2$ **to** N **do**
3: choose $\mathbf{y}_i$
4: look for closest pattern $\mathbf{y}^*$ with latent position $\mathbf{x}^*$
5: **for** $j = 1$ **to** κ **do**
6: update velocity (cf. Equation 6.15)
7: update latent position (cf. Equation 6.14)
8: evaluate $e_{\overline{X}}(\hat{\mathbf{x}})$
9: update best position $\tilde{\mathbf{x}}$
10: **end for**
11: $\overline{\mathbf{X}} = [\overline{\mathbf{X}}, \tilde{\mathbf{x}}]$, $\overline{\mathbf{Y}} = [\overline{\mathbf{Y}}, \mathbf{y}_i]$
12: **end for**

Here, $\tilde{\mathbf{x}}$ is the best latent position

$$\tilde{\mathbf{x}} = \arg\min_{\mathbf{x}} e_{\overline{\mathbf{X}}}(\mathbf{x}) \tag{6.16}$$

the corresponding particle has found in the past optimization steps, and latent position $\mathbf{x}^*$ belongs to the closest embedded pattern

$$\mathbf{y}^* = \arg\min_{\mathbf{y}=\overline{\mathbf{y}}_1,\ldots,\overline{\mathbf{y}}_n} \|\mathbf{y}_i - \mathbf{y}\|^2. \tag{6.17}$$

The parameters $c_1, c_2 \in [0, 1]$ are constants that define the orientation to the best latent particle and the closest already embedded one. Variables $r_1, r_2 \in [0, 1]$ are uniform random values. Figure 6.4 illustrates the particle swarm embedding step. The new candidate latent point $\hat{\mathbf{x}}$ is generated with velocity $\hat{\mathbf{v}}$ and the two scaled vectors.

The approach does not evolve all latent positions at once. As the problem to minimize $E(\mathbf{X})$ scales linearly with the number of patterns N, which might be a very large number in practical applications, the *iterative* solution construction is the key concept for efficiently learning the manifold. A slower PSEA variant employs the overall DSRE (cf. Equation 5.4) for each latent position.

6.7.2 Runtime

The embedding has a runtime complexity of $O(N^2)$, but can be accelerated with *k-d* trees [9] and balltrees in data and latent space, (cf. Section 2.7). Employing space partitioning data structures in data and latent space allows the PSEA to construct a solution in $O(N \log N)$ time, if we assume that the PSO-based search in each step takes constant time. The search for the closest pattern $\mathbf{y}^*$ to $\mathbf{y}_i$ takes $O(\log N)$ employing a *k-d* tree/balltree in data space. This runtime can often not be achieved in high-dimensional spaces.

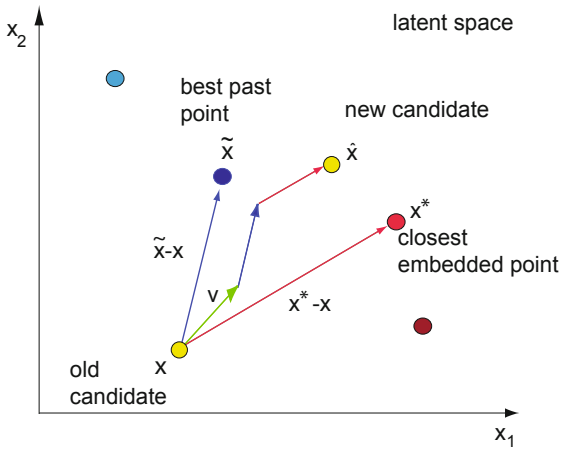


Fig. 6.4 Illustration of particle swarm embeddings: The new candidate latent point $\hat{\mathbf{x}}$ is generated with velocity $\hat{\mathbf{v}}$ and the two scaled vectors $\tilde{\mathbf{x}} - \mathbf{x}$ and $\mathbf{x}^* - \mathbf{x}$

The search for the optimal embedding takes $\kappa \cdot K$-neighborhood computations in latent space, i.e. $O(\kappa \cdot K \cdot \log N) \in O(\log N)$. Insertion of $\tilde{\mathbf{x}}$ to the latent space k-d tree/balltree and $\mathbf{y}_i$ to the data space k-d tree/balltree each take $O(\log N)$. Hence, the overall runtime of the approach can be accelerated to $O(N \log N)$.

6.8 Experimental Analysis of PSEA

In the following, we analyze the results of the novel PSEA experimentally. To evaluate the quality of the embeddings, we employ the normalized DSRE[2] and the co-ranking matrix measure (cf. Section 4.10.3).

6.8.1 Neighborhood Sizes

First, we analyze the influence of neighborhood size K on the results of PSEA, LLE, and ISOMAP on two test data sets, i.e., *Digits* and *Boston*. For PSEA, we choose the following settings. The particle swarm embedding process runs $\kappa = 50$ iterations. The initial velocity is randomly generated with a Gaussian distribution $\mathbf{v}_0 \sim \mathcal{N}(0, 1)$, the initial position starts from the latent position of the closest embedded point $\mathbf{x}_0 = \mathbf{x}^*$. The constants are both set to $c_1 = c_2 = 0.5$.

[2] The normalized DSRE is $E_N(\mathbf{X}) = \frac{1}{N} E(\mathbf{X})$.

Table 6.3 Comparison of DSRE and E_{NX} with PSEA (mean values of 25 runs with standard deviation), LLE, and ISOMAP on the two test data sets *Digits* and *Boston* with $N = 300$ and $\kappa = 50$

Digits	PSEA		ISOMAP		LLE	
K	DSRE	E_{NX}	DSRE	E_{NX}	DSRE	E_{NX}
5	$\mathbf{0.94 \pm 0.01}$	$\mathbf{0.49 \pm 0.01}$	*1.00*	*0.45*	1.23	0.30
10	1.20 ± 0.02	0.42 ± 0.01	**1.03**	**0.54**	*1.08*	*0.50*
15	1.32 ± 0.05	0.42 ± 0.03	**1.13**	**0.57**	*1.16*	*0.52*
30	1.50 ± 0.04	0.42 ± 0.03	**1.28**	**0.64**	*1.42*	*0.51*
Boston	PSEA		ISOMAP		LLE	
K	DSRE	E_{NX}	DSRE	E_{NX}	DSRE	E_{NX}
5	*1.44 ± 0.19*	*0.53 ± 0.02*	**1.05**	**0.67**	2.56	0.35
10	*2.21 ± 0.18*	*0.43 ± 0.01*	**1.38**	**0.65**	2.21	0.42
15	2.69 ± 0.19	0.39 ± 0.03	**1.50**	**0.70**	*2.28*	*0.54*
30	3.44 ± 0.15	0.34 ± 0.01	**2.05**	**0.74**	*2.33*	*0.72*

Table 6.3 shows the experimental results w.r.t. the DSRE and E_{NX} for the settings $K = 5, 10, 15$ and 30. Each PSEA experiment has been repeated 25 times. The best results, i.e., the smallest DSRE and largest E_{NX} values are shown in bold, the second best are shown in italic numbers. The results illustrate that a small DSRE correlates with a large E_{NX}. The DSRE is increasing with the neighborhood size. PSEA achieves the best results of all methods in case of small neighborhood sizes $K = 5$ and $K = 10$ on both data sets. In case of larger neighborhoods, ISOMAP shows better results, but PSEA still computes competitive embeddings and achieves the second best results in half of the cases. LLE and ISOMAP win in performance for larger neighborhoods. The results of LLE are worse than the results of PSEA in three of the four cases, in particular E_{NX} tends to be much worse. Surprising is the bad result of ISOMAP on the *Boston* data set for $K = 10$.

Our experiments with varying data set sizes have shown that ISOMAP and LLE do not scale well in terms of runtime with an increasing number of patterns. The runtime of the PSEA scales slower with the number of patterns. This can be a major advantage of PSEA in comparison to the other methods in large-scale data mining scenarios.

6.8.2 Visual Comparison of Embeddings

In Figure 6.5, we compare PSEA variants with varying neighborhood sizes for $N = 750$ patterns. The figures show that reasonable embeddings have been computed for all neighborhood sizes. The plots show the latent positions of embedded digits in a 2-dimensional latent space. Known labels, which are not

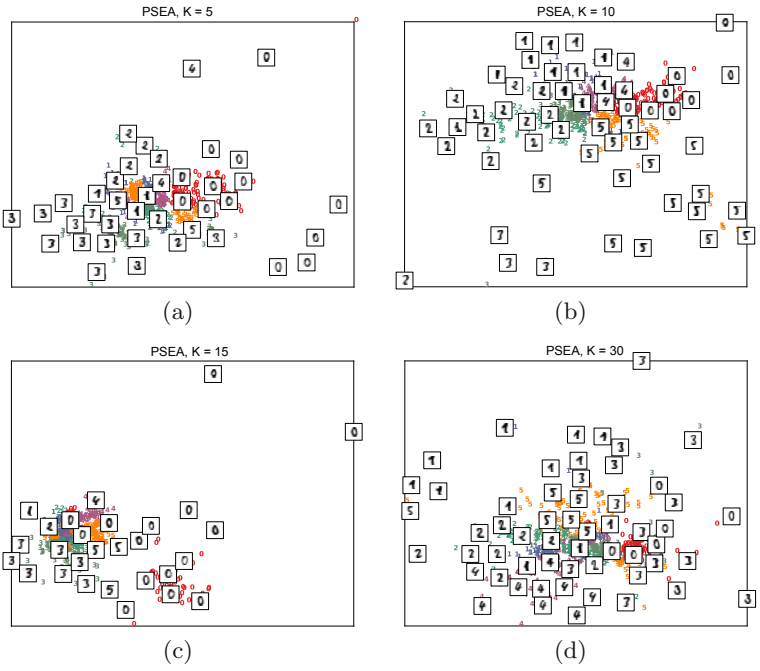
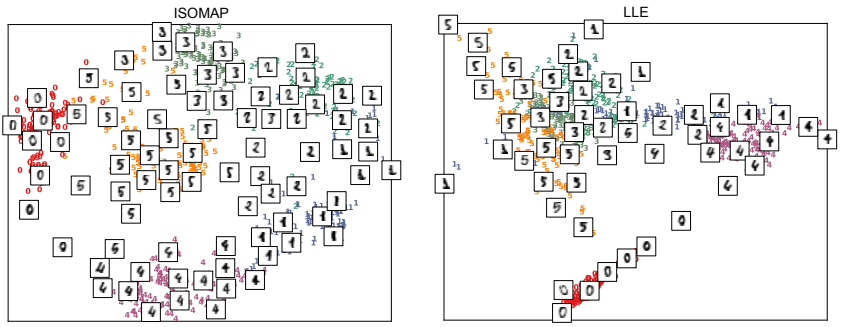


Fig. 6.5 Comparison of embeddings of 750 patterns and six classes of the *Digits* data set. PSEA results for (a) $K = 5$, (b) $K = 10$, (c) $K = 15$, and (d) $K = 30$.

used for the dimensionality reduction process, are used to visualize the learning results plotting the corresponding colored digits. Further, some latent positions are highlighted with small windows to illustrate how neighbored patterns look like. The learning results show that similar digits are mapped to neighbored latent areas. Digits belonging to the same class have the same color and lie closely together. For $K = 15$, two outliers let the rest of the manifold be plotted closely together.

Figure 6.6 shows the embeddings of ISOMAP and LLE on the same data set. Both embeddings also separate the classes and fulfill topological requirements like neighborhood preservation. ISOMAP distributes the latent points circularly in latent space, which leads to better shapes than LLE. The plots show that the PSEA results share characteristics with the ISOMAP result. It even distributes the latent points better than LLE.



(a) (b)

Fig. 6.6 Comparison of embeddings of 750 patterns and 6 classes of the *Digits* data set for (a) ISOMAP embeddings and (b) LLE embeddings with $K = 30$

6.9 Conclusions

In this chapter, we have introduced metaheuristics to compute UNN embeddings. Evolutionary algorithms are strong blackbox optimization methods, but often restricted in the problem dimensionality they are able to handle. The analysis confirms that the evolutionary variants achieve higher accuracies than their heuristic counterparts on small data sets and consequently low problem dimensions. The high accuracy has to be paid with a slow approximation speed. But for data sets larger than $N = 50$, the optimization problem becomes difficult to solve, in particular for the continuous variants. The (1+1)-EA scales slightly better w.r.t. the data set sizes, as the restriction to a latent lattice structure decreases the solution space size. In comparison, the presented UNN heuristics UNN and UNN_g for discrete latent topologies are much faster, in particular for high dimensions. As a conclusion, we can recommend the evolutionary approaches for small data sets, but the iterative heuristics for larger numbers of patterns.

The CMA-ES variants are the first approaches not based on discrete latent grids. Embeddings in continuous latent spaces without a lattice structure offer a greater flexibility for reflecting the structure of high-dimensional data. But the optimization problem is highly multimodal. Further, the optimization problem of placing all latent variables at once scales with the number of patterns and becomes impractical for large data sets. We have introduced an optimization approach that is based on the hybridization of iteratively constructing a solution and PSO-like optimization in each iteration. The approach belongs to the first particle swarm-based dimensionality reduction algorithms. The experimental results turned out to be competitive to embeddings of established methods like ISOMAP and LLE. The experiments have shown that the PSEA embeddings fulfill conditions like neighborhood and distance preservation. PSEA turns out to be a fast and robust algorithm for the embedding of high-dimensional patterns into continuous latent spaces.

7

Kernel and Submanifold Learning

In this chapter, we introduce an extension of UNN for point-wise embeddings into continuous latent spaces of arbitrary dimensionality with stochastic sampling. Distances in data space are employed as variances for Gaussian sampling in latent space. Neighborhoods are preserved with the nearest neighbors data space reconstruction error. Similar to the previous UNN methods, also this approach is an iterative method that constructs an embedding by selecting the latent position with the lowest error. Further, we introduce kernel functions for computing the DSRE in a feature space that allows to better handle non-linearities and high-dimensional data spaces. Experimental studies show that kernel unsupervised nearest neighbors (KUNN) is an efficient method for embedding high-dimensional patterns.

Further, a variant of kernel UNN for continuous latent spaces is introduced that allows to embed patterns in different manifolds with independent parameterizations called submanifolds. The problem to simultaneously assign patterns to models and learn the embeddings is known as submanifold learning or manifold clustering. The problem can be very challenging, as the manifolds may lie closely to each other and can have different dimensions and arbitrary curvature. The UNN-based submanifold learning approach (SL-UNN) that is introduced in this chapter combines a fast constructive K-means variant with UNN. The resulting speedy approach depends on only few parameters, i.e., a distance threshold to allow the definition of new submanifolds and the usual UNN parameters. Extensions of SL-UNN automatically determine the latent dimensions of each submanifold based on the DSRE.

7.1 Gaussian Embeddings

The idea of UNN with stochastic embeddings is to randomly generate points near the closest embedded points in latent space and choose the position that achieves the lowest DSRE. Algorithm 10 shows the pseudocode of UNN

O. Kramer: *Dimensionality Reduction with Unsupervised Nearest Neighb.*, ISRL 51, pp. 93–111.
DOI: 10.1007/978-3-642-38652-7_7

Algorithm 10. Stochastic Embeddings

Require: $\mathbf{Y}$, K, κ
1: $\overline{\mathbf{X}} = [\mathbf{0}]$, $\overline{\mathbf{Y}} = [\mathbf{y}_1]$
2: **for** $i = 2$ **to** N **do**
3: choose $\mathbf{y}_i$
4: look for closest pattern $\mathbf{y}^*$ with latent position $\mathbf{x}^*$
5: **for** $l = 1$ **to** κ **do**
6: $\mathbf{x}_l^* \sim \sigma \cdot \mathcal{N}(\mathbf{x}^*, 1)$ with $\sigma = \|\mathbf{y}_i - \mathbf{y}^*\|^2$
7: **end for**
8: choose $\mathbf{x}_i = \arg\min_{\mathbf{x}=\mathbf{x}_1^*,\ldots,\mathbf{x}_\kappa^*} e_{\overline{\mathbf{X}}}(\mathbf{x})$
9: $\overline{\mathbf{X}} = [\overline{\mathbf{X}}, \mathbf{x}_i]$, $\overline{\mathbf{Y}} = [\overline{\mathbf{Y}}, \mathbf{y}_i]$
10: **end for**

with stochastic embeddings. Later, we extend the concept of neighborhood relations in data space to neighborhoods in kernel-induced feature spaces.

The idea of Gaussian sampling in latent space works as follows. For the first pattern $\mathbf{y}_1$, the latent position can arbitrarily be chosen, e.g., as vector of zeros $\mathbf{x}_1 = \mathbf{0}$. Latent matrix is $\overline{\mathbf{X}} = [\mathbf{x}_1]$, and the corresponding pattern matrix is $\overline{\mathbf{Y}} = [\mathbf{y}_1]$.

Let $\mathbf{y}_1, \ldots, \mathbf{y}_n$ be the sequence of already considered patterns with associated embeddings $\mathbf{x}_1, \ldots, \mathbf{x}_n$. For the next pattern $\mathbf{y}_i$ with $i = n + 1 \leq N$, UNN searches for the closest already embedded pattern

$$\mathbf{y}^* = \arg \min_{\mathbf{y}=\mathbf{y}_1,\ldots,\mathbf{y}_n} \|\mathbf{y}_i - \mathbf{y}\|^2 \tag{7.1}$$

of pattern matrix $\overline{\mathbf{Y}} = [\mathbf{y}_j]_{j=1}^n \in \mathbb{R}^{d \times N}$. Based on the latent position $\mathbf{x}^*$ belonging to pattern $\mathbf{y}^*$, κ candidate latent positions $\mathbf{x}_1^*, \ldots, \mathbf{x}_\kappa^*$ are generated with the Gaussian distribution

$$\mathbf{x}_l^* = \mathbf{x}^* + \mathbf{z}_l \tag{7.2}$$

for $l = 1, \ldots, \kappa$ with

$$\mathbf{z}_l \sim \sigma \mathcal{N}(0, 1), \tag{7.3}$$

and $\sigma = \|\mathbf{y}_i - \mathbf{y}^*\|^2$. The distance σ between pattern $\mathbf{y}_i$ that has to be embedded, and the closest embedded pattern $\mathbf{y}^*$ is employed as scaling factor of the Gaussian-based latent point sampling representing distance preservation of data space in latent space. The candidate latent point is chosen that minimizes the DSRE

$$\mathbf{x}_i = \arg \min_{\mathbf{x}=\mathbf{x}_1^*,\ldots,\mathbf{x}_\kappa^*} e_{\overline{\mathbf{X}}}(\mathbf{x}), \tag{7.4}$$

see Equation 5.2 for the definition of $e_{\overline{\mathbf{X}}}(\mathbf{x})$. The embedding process has a runtime complexity of $O(N^2)$. If space partitioning data structures like k-d-trees allow the acceleration of neighborhood requests to $O(\log N)$, UNN can be accelerated to $O(N \log N)$.

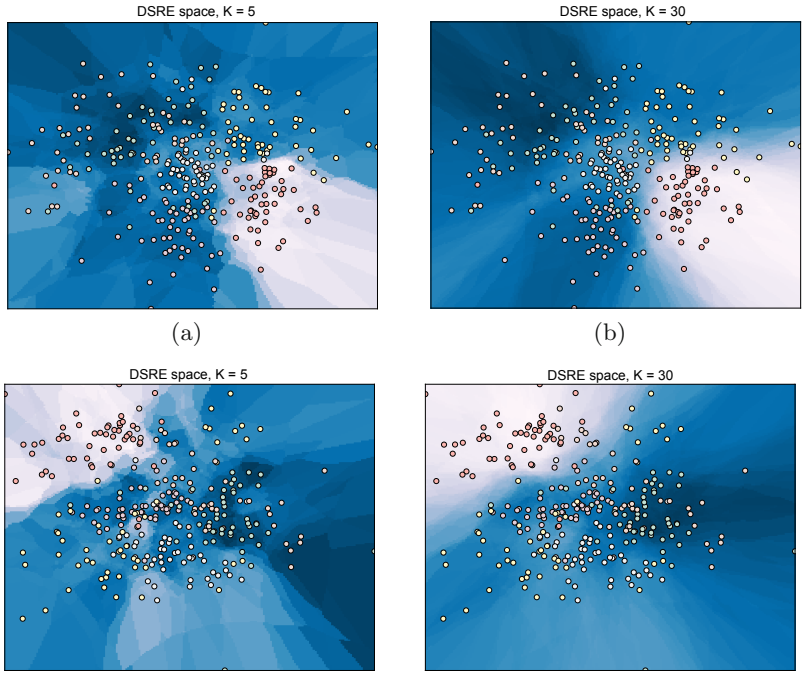


Fig. 7.1 Visualization of DSRE space $e_{\mathbf{X}}(\cdot)$ w.r.t. the first embedded pattern $\mathbf{y}_1$ in the center of the plot for two runs (upper row and lower row) of UNN with $N = 300$ and neighborhood sizes $K = 5$ (left) and $K = 30$ (right)

To illustrate the search for optimal latent positions, we visualize the DSRE space in Figure 7.1. The plots show the DSRE w.r.t. the first pattern $\mathbf{y}_1$ after embedding of $N = 300$ patterns with two seeds for generation of random numbers employing UNN with Gaussian embeddings. Figure 7.1(a) shows a run with the first seed, employing neighborhood size $K = 5$, Figure 7.1(b) shows the same situation with neighborhoods size $K = 30$. Figures 7.1(c) and 7.1(d) show the corresponding results for the second run. Bright areas represent parts of latent space with low errors, while dark colors represent a large DSRE. The figures show that the DSRE space for embedding *one* pattern is multimodal, in particular for small K. For increasing neighborhood sizes, the DSRE space becomes *less multimodal*, as the number of different K-neighborhoods decreases, and there are larger areas with similar fitness, i.e., plateaus. Such search spaces overlap in case of many patterns leading to a highly multimodal situation that is hard to optimize.

7.2 Kernel UNN

Kernel functions have become very popular in the last decade and are important ingredients of many state-of-the-art methods in machine learning. The motivation for kernel functions is to cope with non-linearities in data space. We extend the stochastic iterative embedding algorithm to the kernel variant KUNN.

7.2.1 Kernel Functions

Kernel methods take advantage of an interesting property of a reproducing kernel Hilbert space [4]. A kernel is a real-valued function of two input space elements that corresponds to a scalar product of its arguments mapped to some metric feature space. The *kernel trick* is the effect that all operations in a feature space of higher dimensions can be expressed by scalar products that can efficiently be computed. A kernel function $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ induces a feature mapping $\phi : \mathbb{R}^d \to \mathcal{H}$ into some potentially high-dimensional feature space $\mathcal{H}$ such that

$$k(\mathbf{y}, \mathbf{y}') = \langle \phi(\mathbf{y}), \phi(\mathbf{y}') \rangle. \tag{7.5}$$

Often employed kernel functions are linear, polynomial, and Gaussian kernels. The linear kernel is based on the inner product

$$k(\mathbf{y}, \mathbf{y}') = \langle \mathbf{y}, \mathbf{y}' \rangle, \tag{7.6}$$

which is *one* in case of identity of both vectors and *zero* in case they are orthogonal. The polynomial kernel

$$k(\mathbf{y}, \mathbf{y}') = \langle \mathbf{y}, \mathbf{y}' \rangle^p \tag{7.7}$$

employs a polynomial function with $p \in \mathbb{N}$. For example, for $p = 2$ and $d = 2$, the mapping $\phi : \mathbb{R}^2 \to \mathcal{H} = \mathbb{R}^3$ into higher dimensions can explicitly be demonstrated

$$\begin{aligned} k(\mathbf{y}, \mathbf{y}') &= \langle \mathbf{y}, \mathbf{y}' \rangle^2 = (y_1 y_1' + y_2 y_2')^2 && (7.8)\\ &= y_1^2 y_1'^2 + y_2^2 y_2'^2 + 2 y_1 y_1' y_2 y_2' && (7.9)\\ &= \langle (y_1^2, y_2^2, \sqrt{2} y_1 y_2), (y_1'^2, y_2'^2, \sqrt{2} y_1' y_2') \rangle && (7.10)\\ &= \langle \phi(\mathbf{y}), \phi(\mathbf{y}') \rangle. && (7.11)\\ & && (7.12) \end{aligned}$$

An often employed kernel function, in particular for support vector classification and regression [98, 101], is the radial basis function kernel (RBF-kernel)

$$k(\mathbf{y}, \mathbf{y}') = \exp\left(-\gamma \|\mathbf{y} - \mathbf{y}'\|^2\right) \tag{7.13}$$

with $\gamma > 0$. The parameter is often chosen as $\gamma = 1/\sigma^2$ with bandwidth σ. The hyperbolic tangent is another kernel function, which is used less often. It is defined as

$$k(\mathbf{y}, \mathbf{y}') = \tanh(\alpha \cdot \langle \mathbf{y}, \mathbf{y}' \rangle + \beta) \tag{7.14}$$

with $\alpha, \beta \in \mathbb{R}$, $\alpha > 0$ and $\beta < 0$. In the following, we employ kernel functions for the DSRE computation of UNN. The kernel functions will be experimentally analyzed in Section 7.3.

7.2.2 Kernelization of DSRE

In UNN, the employment of kernel functions for computation of the DSRE allows to capture non-linear structures in data space corresponding to *non-linear Voronoi boundaries*. For each pattern $\mathbf{y}_i$ with $i = 2, \ldots N$ that has to be embedded, and its mapping $\phi(\mathbf{y}_i)$ into feature space, we look for the closest embedded pattern $\phi(\mathbf{y}^*)$ in feature space. Similarity can directly be expressed with kernel function $k(\cdot, \cdot) : \mathbb{R}^d \times \mathbb{R}^d \to \mathcal{H}$

$$\mathbf{y}^* = \arg \max_{\mathbf{y} = \overline{\mathbf{y}}_1, \ldots, \overline{\mathbf{y}}_n} k(\mathbf{y}_i, \mathbf{y}). \tag{7.15}$$

With a kernel, the DSRE $e_{\overline{\mathbf{X}}}(\mathbf{x})$ is computed in feature space as follows:

$$\begin{aligned} e_{\overline{\mathbf{X}}}(\mathbf{x}) &= \|\phi(\mathbf{f}_{\overline{\mathbf{X}}}(\mathbf{x})) - \phi(\mathbf{y})\|^2 && (7.16)\\ &= \langle \phi(\mathbf{f}_{\overline{\mathbf{X}}}(\mathbf{x})), \phi(\mathbf{f}_{\overline{\mathbf{X}}}(\mathbf{x})) \rangle - 2\langle \phi(\mathbf{f}_{\overline{\mathbf{X}}}(\mathbf{x})), \phi(\mathbf{y}) \rangle + \langle \phi(\mathbf{y}), \phi(\mathbf{y}) \rangle && (7.17)\\ &= k(\mathbf{f}_{\overline{\mathbf{X}}}(\mathbf{x}), \mathbf{f}_{\overline{\mathbf{X}}}(\mathbf{x})) - 2k(\mathbf{f}_{\overline{\mathbf{X}}}(\mathbf{x}), \mathbf{y}) + k(\mathbf{y}, \mathbf{y}) && (7.18) \end{aligned}$$

The kernel DSRE is the basis of the embeddings of the experimental part.

7.3 Experimental Analysis of KUNN

In the following experimental study, we compare the behavior of the introduced methods on selected artificial data sets. Besides visualization, we compare the experimental results w.r.t. the DSRE and the co-ranking matrix measure E_{NX} by Lee and Verleysen [73], cf. Section 4.10.3.

7.3.1 RBF-Kernel

One of the most frequently employed kernel function is the RBF-kernel. We explore the influence of kernel bandwidth γ of the RBF-kernel in the following experimental analysis. Table 7.1 shows the normalized DSRE[1] and E_{NX} of KUNN with RBF-kernel for three settings of γ, i.e., $\gamma = 1.0, 10^{-4}$, and 10^{-8}. The figures show the mean DSRE and the corresponding standard deviation of 25 runs. The choice of γ has a significant influence on the learning result. The best embeddings w.r.t. the DSRE have been achieved for $\gamma = 10^{-8}$ in case of data set *Digits* and also in case of *Boston*. As of $\gamma = 10^{-9}$, the achieved

[1] Similar to Chapter 6, the normalized 'kernel-free' DSRE is $E_N(\mathbf{X}) = \frac{1}{N} E(\mathbf{X})$.

Table 7.1 Analysis of kernel bandwidth γ of KUNN with RBF-kernel on the *Digits* data set with $N = 300$ patterns and settings $K = 10$, $\kappa = 30$. The best results are shown in bold, the second best in italic figures.

data	*Digits*		*Boston*	
γ	DSRE	E_{NX}	DSRE	E_{NX}
1.0	1.30 ± 0.01	0.30 ± 0.01	2.43 ± 0.10	0.29 ± 0.01
10^{-4}	*1.11 ± 0.02*	*0.45 ± 0.01*	*2.15 ± 0.12*	*0.42 ± 0.01*
10^{-8}	**0.91 ± 0.01**	**0.50 ± 0.01**	**1.14 ± 0.15**	**0.62 ± 0.04**

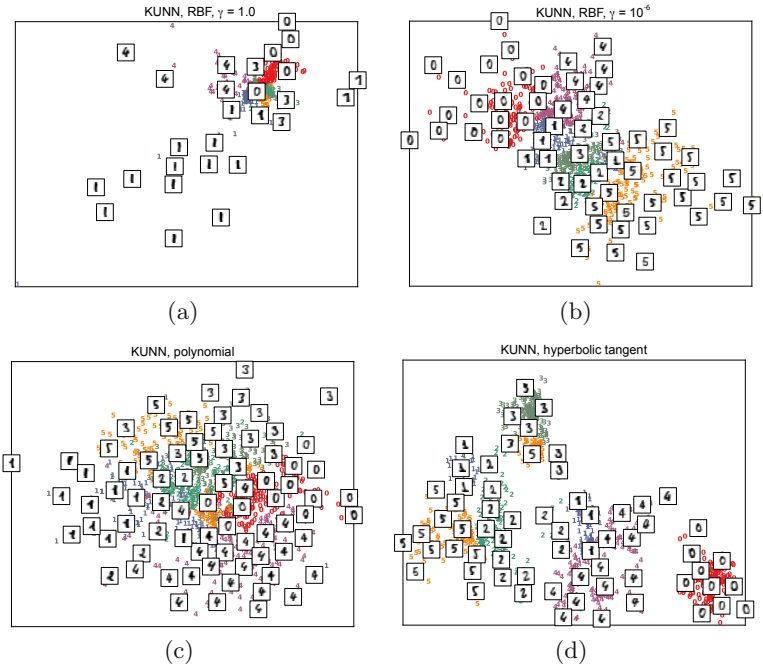


Fig. 7.2 Comparison between embeddings of KUNN with RBF-kernel and parameter settings (a) $\gamma = 1.0$, (b) $\gamma = 10^{-6}$, (c) a polynomial (p=4), and (d) a hyperbolic tangent kernel ($\alpha = 10^{-6}$ and $\beta = -10^{-2}$) on the *Digits* data set with $K = 10$, $\kappa = 30$, and $N = 1,000$

DSRE varies only after the tenth decimal place. Figures 7.2(a) and 7.2(b) show exemplary embeddings for the bandwidth settings $\gamma = 1.0$ and $\gamma = 10^{-6}$ on the *Digits* data set. The distribution of the majority of latent points is comparatively narrow for $\gamma = 1.0$ due to outliers, while for $\gamma = 10^{-6}$ the manifold becomes broader and well distributed. The plots confirm the choice for γ determined in Table 7.1, i.e., the tendency towards smaller settings.

7.3.2 Kernel Function Comparison

The influence of the kernel function type on the embedding result is analyzed in the following. The comparison includes the linear kernel, the polynomial kernel, and the hyperbolic tangent kernel. Table 7.2 shows an experimental comparison of the three kernel functions on *Digits* and *Boston* w.r.t. DSRE and E_{NX}. The linear kernel is parameter-free. For the polynomial kernel, we choose $p = 2$ and for the hyperbolic tangent kernel we choose the settings $\alpha = 10^{-6}$ and $\beta = -10^{-2}$. We can observe that the polynomial kernel achieves better results than the linear kernel in minimizing the DSRE and maximizing the co-ranking matrix measure E_{NX} on the *Digits* data set and vice versa on the *Boston* data set. But the co-ranking matrix value is comparatively bad. Only the fraction of about 0.3 of the data space neighborhood is maintained in latent space. On the contrary, the hyperbolic tangent kernel shows surprisingly good results that outperform the RBF-kernel on both data sets. Figures 7.2(c) and 7.2(d) show a visualization of exemplary embeddings of the polynomial kernel ($p = 4$) and the tangent kernel ($\alpha = 10^{-6}$ and $\beta = -10^{-2}$). In particular, the hyperbolic tangent is able to separate the different classes. The groups of latent points with same colors that are well separated from each other demonstrate the quality the learning result.

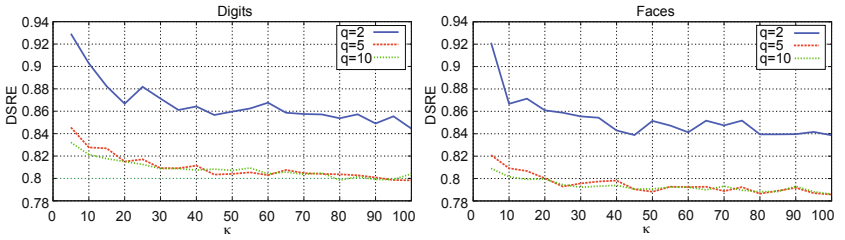
Table 7.2 Comparison between linear, polynomial, and hyperbolic tangent kernels on the *Digits* and the *Boston* data set with $N = 300$, $K = 5$, and $\kappa = 30$

kernel	linear		polynomial		tangent	
data	DSRE	E_{NX}	DSRE	E_{NX}	DSRE	E_{NX}
Digits	1.14 ± 0.03	0.31 ± 0.01	*1.12 ± 0.01*	*0.32 ± 0.01*	$\mathbf{0.85 \pm 0.02}$	$\mathbf{0.56 \pm 0.01}$
Boston	*1.87 ± 0.04*	*0.31 ± 0.01*	1.90 ± 0.08	0.31 ± 0.01	$\mathbf{1.00 \pm 0.07}$	$\mathbf{0.68 \pm 0.01}$

7.3.3 Analysis of Latent Space Dimensionality

An interesting question is the influence of latent space dimensions on the DSRE. A hypothesis is that neighborhood relations can better be reflected with larger latent dimensionality q. However, a larger q requires a larger κ to sample in all directions and to allow testing a sufficient number of latent space positions. Figure 7.3 shows the DSRE with different latent dimensions q depending on the number κ of latent space samples (a) on the *Digits* data set and (b) on the *Faces* data set. The curves show the mean DSRE of 25 runs.

The plots show that the DSRE is decreasing with increasing dimensionality q and confirm the hypotheses. A significant decrease of DSRE can be observed from $q = 2$ to $q = 5$, which can be explained with the higher flexibility to place latent points in higher dimensions. There are more possibilities to place latent points forming the same neighborhoods. But to exploit the larger degree



(a) (b)

Fig. 7.3 Analysis of DSRE depending on latent space dimensionality q and number κ of samples tested in each iteration on (a) the *Digits* and (b) the *Faces* data set

of freedom, it is necessary to invest more search, and the DSRE is further decreasing with the number κ of samples keeping q constant. The error can be decreased significantly less from $q = 5$ to $q = 10$. Obviously, five dimensions offer a sufficient degree of freedom for flexible embeddings. Further, a larger number of sampled latent positions is necessary to decrease the error, an effect caused by the *curse of dimensionality* effect (cf. Section 2.6).

7.3.4 Comparison between KUNN, LLE and ISOMAP

Last, we compare UNN with stochastic embeddings and KUNN with kernel functions to ISOMAP and LLE w.r.t. neighborhood size K. The embeddings of the *Digits* data set of UNN without kernels and KUNN are shown in Figures 7.4 (a) and (b). Different classes are separated, and similar digits are neighbored. KUNN achieves a better separation of different classes than UNN without kernels. In comparison to the LLE result, the embeddings are smoother and better distributed. Also ISOMAP computes a smooth embedding with similar patterns lying close to each other in latent space.

To evaluate the embeddings quantitatively, we again employ the DSRE and E_{NX}. The experimental results can be found in Table 7.3. From the analysis of different kernels in Section 7.3.2, we choose the hyperbolic tangent kernel with parameters $\alpha = 10^{-6}$ and $\beta = -10^{-2}$. Again, UNN and KUNN have been run 25 times, and the corresponding mean values and standard deviations are shown. Comparing UNN to KUNN, we can observe that the employment of a kernel function significantly improves the embedding results. KUNN achieves better DSRE results and neighborhood preserving values for measure E_{NX}. In general, a small DSRE is strongly correlated to a large E_{NX} result. When we compare the UNN variants to ISOMAP and LLE, we can observe that KUNN is competitive on the *Digits* data set for small neighborhood sizes, i.e., $K = 5, 10$, while ISOMAP is superior in two cases

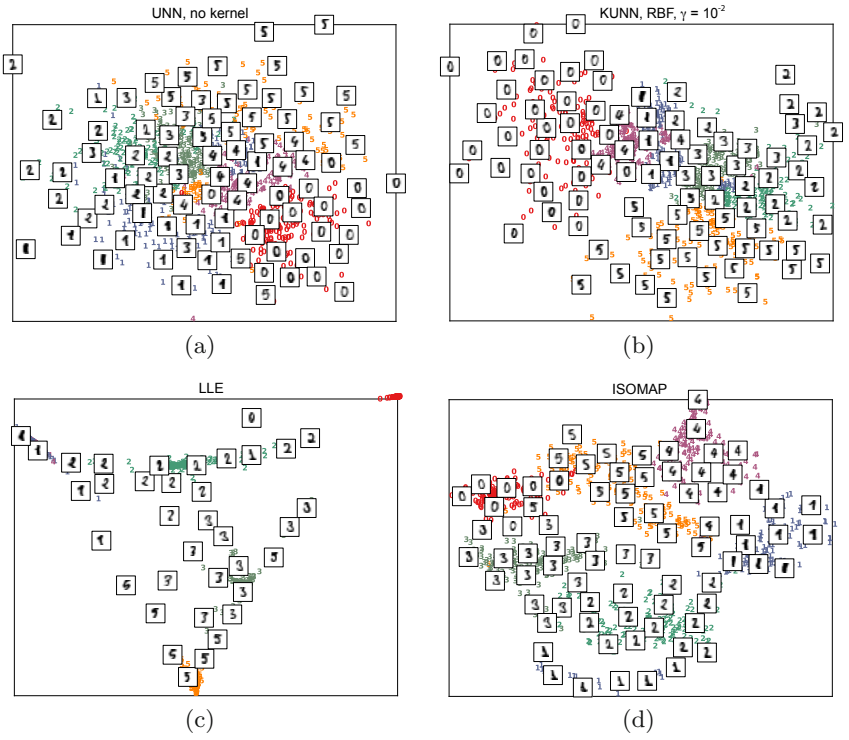


Fig. 7.4 Embeddings of (a) UNN, (b) KUNN employing an RBF-kernel with $\gamma = 10^{-2}$, (c) LLE, and (d) ISOMAP on the *Digits* data set ($K = 10$, $\kappa = 30$, $N = 1,000$)

on *Boston*. With one exception (*Boston* and $K = 30$), KUNN is superior to LLE w.r.t. both measures.

In a last experiment, we compare ISOMAP to KUNN with RBF-kernel on the ISOMAP-*Faces* data set that is employed in the original ISOMAP article [103]. This data set contains images of a statue with different poses and lights. Figure 7.5 shows the results of (a) ISOMAP with $K = 50$ and (b) KUNN using the RBF-kernel with setting $\gamma = 10^{-4}$ and neighborhood size $K = 5$. Both approaches compute topology preserving embeddings: similar poses and lights of the statue are neighbored in latent space. The embedding is competitive to the learning result of ISOMAP.

Table 7.3 Comparison of DSRE and E_{NX} between UNN, KUNN, LLE, and ISOMAP on the two test data sets *Digits* and *Boston*, each with $N = 300$ patterns. ISOMAP and KUNN achieve the lowest DSRE and show the best ability to preserve neighborhoods in latent space ($\kappa = 30$, 25 repetitions for UNN and KUNN)

Digits	UNN		KUNN		ISOMAP		LLE	
K	DSRE	E_{NX}	DSRE	E_{NX}	DSRE	E_{NX}	DSRE	E_{NX}
5	1.14 ± 0.02	0.31 ± 0.01	$\mathbf{0.86 \pm 0.01}$	$\mathbf{0.55 \pm 0.01}$	*1.00*	*0.45*	1.23	0.30
10	1.27 ± 0.03	0.31 ± 0.01	$\mathbf{1.03 \pm 0.01}$	0.53 ± 0.01	**1.03**	**0.54**	1.08	0.50
30	1.52 ± 0.01	0.40 ± 0.01	1.33 ± 0.02	0.57 ± 0.01	**1.28**	**0.64**	1.42	0.51
Bost.	UNN		KUNN		ISOMAP		LLE	
K	DSRE	E_{NX}	DSRE	E_{NX}	DSRE	E_{NX}	DSRE	E_{NX}
5	1.94 ± 0.11	0.31 ± 0.01	$\mathbf{1.00 \pm 0.07}$	$\mathbf{0.68 \pm 0.01}$	*1.05*	*0.67*	2.56	0.35
10	2.32 ± 0.03	0.30 ± 0.02	1.57 ± 0.08	0.62 ± 0.01	**1.38**	**0.65**	2.21	0.42
30	3.30 ± 0.04	0.37 ± 0.01	2.85 ± 0.18	0.56 ± 0.02	**2.05**	**0.74**	*2.33*	*0.72*

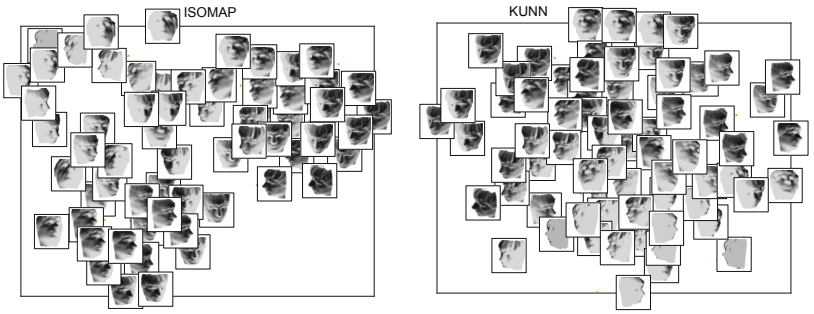


Fig. 7.5 Comparison between embeddings of the ISOMAP-*Faces* data set of (a) ISOMAP with $K = 50$ and (b) KUNN with $K = 5$ and RBF-kernel with $\gamma = 10^{-4}$

7.4 Manifold Clustering

To allow the simultaneous assignment to clusters and learning of submanifolds, we introduce a submanifold variant of UNN in the following. First, we characterize the manifold clustering problem and present related work.

7.4.1 Problem Definition

Given a matrix of N patterns $\mathbf{Y} = [\mathbf{y}_i]_{i=1}^N \in \mathbb{R}^{d \times N}$ lying in τ different manifolds $\{\mathcal{M}_j\}_{j=1}^\tau$ with intrinsic dimensions $\{d_j\}_{j=1}^\tau$, the submanifold learning problem is to simultaneously assign the patterns to clusters and solve the

manifold learning problem independently within each cluster. The problem is difficult, as first clusters have to be identified, second low-dimensional representations of the patterns have to be learned, and third in each cluster possibly varying parameters have to be chosen. Vidal [107] summarizes challenges of submanifold learning. The coupling between segmentation of patterns and model estimation is the most difficult challenge. A known segmentation would simplify the model estimation process, as it would be clear, which patterns belong to which manifold. In turn, a known distribution would simplify the segmentation process, as the distributions define the manifolds.

In general, the distribution within the clusters is unknown. Closeness and intersections between submanifolds extremely complicate the segmentation and the model estimation process. The perspective of the data space as collection of submanifolds with varying characteristics is similar to the concept of local models that allow separate parameterizations. In [67], we presented a kernel regression approach with multiple local models and independent kernel parameters.

If it is possible to sufficiently separate manifolds with clustering techniques, dimensionality reduction methods like ISOMAP and LLE with appropriately chosen neighborhood sizes are a possible way to solve submanifold learning tasks. Our approach is based on a similar idea, i.e., combining iterative clustering with the UNN strategy. After the presentation of related work, we introduce an iterative K-means variant that is the basis of the iterative submanifold learning method.

7.4.2 Manifold Clustering Approaches

Numerous submanifold learning algorithms have been presented in the past. Algebraic methods are based on matrix factorization [20, 31] and polynomial algebra, e.g., fitting polynomials to the submanifolds [108]. Iterative methods are often extensions of K-means, alternately fitting PCA-models to each submanifold and then assigning each pattern to its closest submanifold. K-planes [13] and K-subspaces [105] are instances of these algorithms. To handle noise, statistical models like mixtures of probabilistic PCA [104] assume that data within submanifolds are generated with independent Gaussian distributions employing the maximum likelihood principle. The assumption of Gaussian mixtures is the basis of the agglomerative lossy compression approach [76], which minimizes coding length required for fitting the Gaussians.

Many approaches are based on spectral clustering computing an affinity matrix with entries that measure the similarity between patterns. The definition of the affinity matrix is the key problem in applying spectral clustering to the submanifold learning problem. Local subspace affinity [114], spectral local best-fit flats [115], and locally manifold clustering [34] are based on choosing a pattern and its q-nearest neighbors. An affine subspace is fitted

to each group, and a pairwise affinity is computed by comparing the subspaces. Sparse subspace clustering [26] is a variant that restricts the choice to q *sparse* neighbors. A further method is sparse manifold clustering and embedding [27] that searches for neighborhood weights solving a sparse optimization problem. The technique is based on selecting close patterns lying in the same manifold.

Further, evolutionary approaches have been introduced for submanifold learning that concentrate on the evolutionary blackbox choice of attributes [83, 106]. The approach by Vahdat *et al.* [106] employs multi-objective search to identify a set of candidate clusters balancing intra-cluster distance and connectedness of clusters. For a deeper introduction to submanifold learning approaches, we refer to Vidal [107] and Luxburg [109].

7.5 Constructive K-Means

A fast variant of K-means is an iterative algorithm that does not require the initial specification of the number of clusters. To determine if new patterns belong to a novel cluster, a distance threshold $\zeta \in \mathbb{R}^+$ must be defined.

Algorithm 11. Constructive K-Means

Require: $\mathbf{Y}$, ζ
1: $K = 1$
2: $\mathbf{c}_1 = \mathbf{y}_1$, $\mathcal{I}(\mathbf{y}_1, \mathbf{c}_1) = 1$
3: **for** $i = 2$ **to** N **do**
4: choose $\mathbf{y}_i$
5: look for closest codebook vector $\mathbf{c}^*$
6: **if** $\|\mathbf{y}_i - \mathbf{c}^*\|^2 > \zeta$ **then**
7: $K = K + 1$
8: $\mathbf{c}_K = \mathbf{y}_i$
9: $\mathcal{I}(\mathbf{y}_i, \mathbf{c}_K) = 1$
10: **else**
11: $\mathcal{I}(\mathbf{y}_i, \mathbf{c}^*) = 1$
12: update $\mathbf{c}^*$
13: **end if**
14: **end for**

K-means successively repeats the two steps of (1) assigning each pattern to its closest codebook vector representing a cluster center and (2) recomputing the codebook vectors of all patterns assigned to the corresponding cluster.

The idea of the fast constructive K-means variant is to iteratively assign each pattern to the cluster of the closest codebook vector and then to update the center, see Algorithm 11. The first pattern $\mathbf{y}_1$ is the first cluster center $\mathbf{c}_1 = \mathbf{y}_1$, and we set $\mathcal{I}(\mathbf{y}_1, \mathbf{c}_1) = 1$.

Let n be the number of patterns that have been assigned to clusters. For all remaining patterns $\mathbf{y}_i$ with $i = n+1 \leq N$, the algorithm looks for the closest codebook vector $\mathbf{c}^*$. If the distance to pattern $\mathbf{y}_i$ exceeds a threshold $\|\mathbf{y}_i - \mathbf{c}^*\|^2 > \zeta$, it becomes a new cluster center. We increase the number of clusters $K = K + 1$ and set $\mathbf{c}_K = \mathbf{y}_i$, as well as $\mathcal{I}(\mathbf{y}_i, \mathbf{c}_K) = 1$. Otherwise, pattern $\mathbf{y}_i$ is assigned to the cluster of codebook vector $\mathbf{c}^*$, i.e., we set $\mathcal{I}(\mathbf{y}_i, \mathbf{c}^*) = 1$. Further, the cluster has to be updated (compare to Equation 4.1)

$$\mathbf{c}^* = \frac{\sum_{i=1}^{n+1} \mathcal{I}(\mathbf{y}_i, \mathbf{c}^*) \cdot \mathbf{y}_i}{\sum_{i=1}^{n+1} \mathcal{I}(\mathbf{y}_i, \mathbf{c}^*)}. \tag{7.19}$$

This process is repeated until all patterns are assigned to clusters. Figure 7.6 illustrates constructive K-means. Patterns with the same color belong to the same cluster, white patterns are not assigned yet. Unassigned patterns are iteratively assigned to the closest codebook vector. Figure 7.6(a) shows the assignment of a pattern $\mathbf{y}_i$ that lies outside the ζ-ball of the two cluster centers $\mathbf{c}_1$ and $\mathbf{c}_2$. A novel cluster is started with center $\mathbf{c}_3 = \mathbf{y}_i$. Figure 7.6(b) shows the situation that pattern $\mathbf{y}_i$ lies within an ζ-ball around $\mathbf{c}_1$ and is consequently assigned to cluster A. Constructive K-means computes the clustering in $O(\tau \cdot N)$, if τ is the number of submanifolds. The result of this iterative method depends on the order of elements. The algorithm requires the specification of threshold ζ to determine, if it is reasonable to start a new cluster. The clustering result is very sensitive to this parameter. The experimental analysis in Section 7.7 will show that too large values result in too few clusters, while too small values generate too many small clusters. For $\zeta \leq \min \|\mathbf{y}_i - \mathbf{y}_j\|^2$ with $i, j = 1, \ldots, N$ and $i \neq j$, constructive K-means returns N submanifolds.

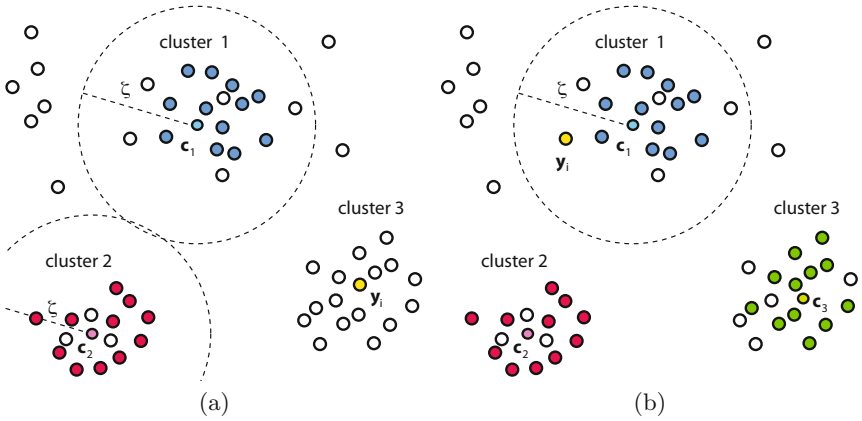


Fig. 7.6 In constructive K-means, two cases can occur when embedding pattern $\mathbf{y}_i$: (a) A novel cluster is started, if pattern $\mathbf{y}_i$ is further away than ζ from any cluster center or (b) pattern $\mathbf{y}_i$ is assigned to the cluster with the closest center

7.6 Submanifold Learning UNN

The submanifold learning approach SL-UNN is based on the idea to combine constructive K-means with UNN. The iterative construction scheme of K-means can easily be combined with the iterative UNN scheme. A pattern is assigned to a submanifold $\mathcal{M}_j$ and immediately embedded in $\mathcal{M}_j$. Algorithm 12 shows the pseudocode of SL-UNN.

The first pattern $\mathbf{y}_1$ is assigned to the first manifold and embedded at an initial latent position, i.e., $\mathcal{M}_1 = \{(\mathbf{0}, \mathbf{y}_1)\}$. Let n be the number of patterns that have been assigned to submanifolds. For all remaining patterns $\mathbf{y}_i$ with $i = n + 1 \leq N$, constructive K-means assigns $\mathbf{y}_i$ to a manifold $\mathcal{M}_j$, where the pattern is embedded with UNN. An arbitrary latent position $\mathbf{x}_i$ can be chosen, when a new manifold is started. In the experimental part, we will place the latent submanifold centers on a lattice with distances between the grid points that allow to distinguish between different manifolds.

The embedding is only based on the patterns that are assigned to $\mathcal{M}_j$, i.e., only the patterns of $\mathcal{M}_j$ are used for the neighborhood search and the DSRE computation, while patterns that are not part of the submanifold are neglected. For each manifold, separate parameterizations allow to improve the model estimations. In the experimental part, we will allow to use different kernel functions with separate parameters.

Algorithm 12. Submanifold Learning

Require: $\mathbf{Y}$, ζ
1: $K = 1$
2: $\mathbf{c}_1 = \mathbf{y}_1$
3: $\mathcal{M}_1 = \{(\mathbf{0}, \mathbf{y}_1)\}$
4: **for** $i = 2$ **to** N **do**
5: constructive K-means Steps 4 – 12 (j is index of last cluster)
6: embed $\mathbf{y}_i$ in $\mathcal{M}_j$ with UNN and $\kappa, \sigma_j^* \rightarrow \mathbf{x}_i$
7: $\mathcal{M}_j = \mathcal{M}_j \bigcup \{(\mathbf{x}_i, \mathbf{y}_i)\}$
8: **if** $|\mathcal{M}_j| = \vartheta$ **then**
9: choose parameters σ_j^* for manifold $\mathcal{M}_j$
10: **end if**
11: **end for**

The runtime complexity of the approach lies in the same class as UNN. SL-UNN iteratively assigns a pattern to the closest codebook vector, whose number is upper bounded by τ, in $O(\tau) \in O(1)$ steps and embeds a pattern in the assigned manifold $\mathcal{M}_j$ in $O(\log N)$ resulting in an overall runtime of $O(N \cdot \tau + N \log N) \in O(N \log N)$.

The parameters in each manifold can be adapted to allow better manifold learning results. For this purpose, we introduce the following approach. When ϑ patterns have been embedded in submanifold $\mathcal{M}_j$ with $j = 1, \ldots, \tau$, the

algorithm optimizes the corresponding settings, which can be dimensionalities $\{d_j\}_{j=1}^{\tau}$, kernel functions $\{k\}_{j=1}^{\tau}$, and corresponding kernel parameters. The two most important variants for the parameter adaptation process are: (1) grid search, the definition of parameter sets $\{\sigma\}_{f=1}^{g}$ that are successively tested and (2) optimization of parameters within defined bounds. We will employ the first variant in the following experimental analysis. The latter is recommendable for parameter sets that are too large to enumerate completely.

7.7 Experimental Analysis of SL-UNN

In this section, we analyze SL-UNN experimentally on the typical data sets of the previous experiments and compare it to KUNN without manifold clustering.

7.7.1 Digits

The first experimental part focuses on the visualization of the submanifold learning results on the *Digits* data set. We expect that SL-UNN identifies different clusters corresponding to the different classes of *Digits*, and embeds the patterns within each cluster. As first data set, we consider an example consisting of three classes. Figure 7.7 shows the results of SL-UNN with various settings for ζ on the *Digits* data set with $N = 539$ patterns and digits '0', '3', and '7'. We choose $\kappa = 20$ and neighborhood size $K = 10$. For a too large threshold $\zeta = 60.0$, only one manifold has been found. Hence, the result corresponds to standard UNN without clustering. For $\zeta = 50.0$, two manifolds have been identified. In particular, the '0's are separated from the rest of the patterns with exception of few outliers. For $\zeta = 47.0$, the number of manifolds is identical with the number of classes, i.e., $\tau = 3$. This is the optimal setting, which reflects the different classes and corresponding reasonable embeddings. However, we can observe false patterns in some manifolds. This is caused by errors of the constructive K-means clustering procedure. For too small threshold values, too many manifolds are found, e.g., $\tau = 14$ manifolds in case of $\zeta = 35.0$.

For a further data set composed of the *Digits* data set with six classes (digits from '0' to '5'), we can observe similar results, see Figure 7.8. For too large distance thresholds like $\zeta = 47.0$, only two clusters were found, while for $\zeta = 45.0$ already four clusters have been detected and even $\tau = 25$ in case of $\zeta = 35.0$. But with the setting $\zeta = 43.0$, SL-UNN identifies a number of manifolds that corresponds to the number of classes.

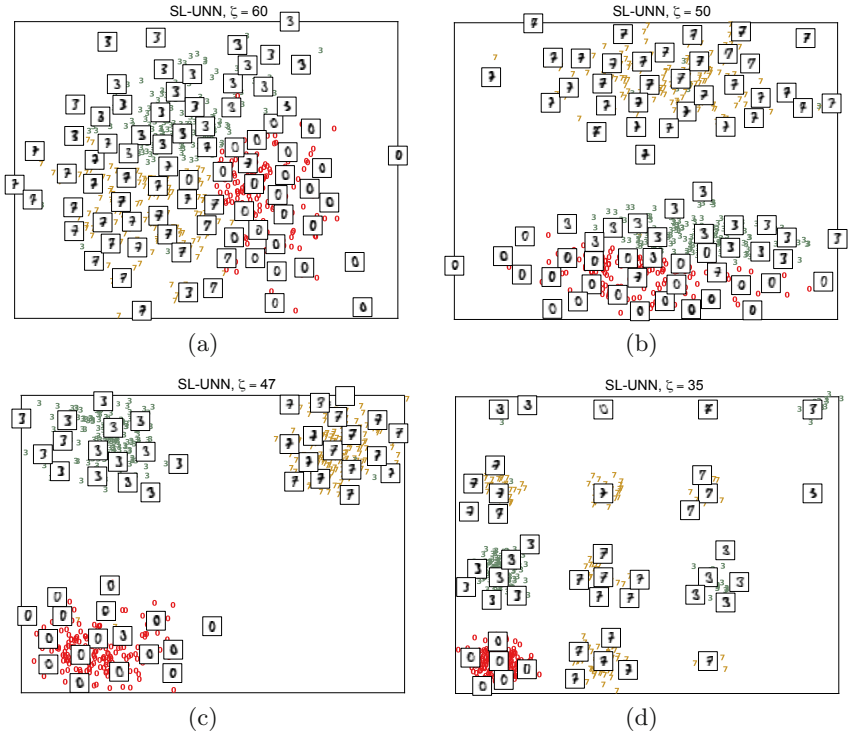


Fig. 7.7 Comparison of SL-UNN embeddings of the *Digits* data set ('0', '3', and '7') in 2-dimensional manifolds with varying ζ

7.7.2 DSRE in Submanifolds

In the following, we analyze the DSRE within the submanifolds. The parameters for the model of each manifold are adapted according to the following scheme after ϑ patterns have been assigned to manifold $\mathcal{M}_j$:

1. linear kernel,
2. polynomial kernel with $p = 2$,
3. RBF-kernel with parameters $\gamma = 10^l, 0 \leq l \leq 8$,
4. hyperbolic tangent kernel with $a = 10^{-l}$, $1 \leq l \leq 6$, and $b = -10^{-2}$.

For each manifold $\{\mathcal{M}_j\}_{j=1}^{\tau}$, the setting $\{\sigma_j^*\}_{j=1}^{\tau}$ that achieves a minimal DSRE with $\vartheta = 50$ patterns is chosen.

Table 7.4 shows the experimental comparison between SL-UNN with $\zeta = 43.0$ and KUNN without manifold clustering in 25 runs.[2] The results

[2] The figures show the average pattern-wise DSRE* $E(\mathbf{X}) = \frac{1}{N}\sum_{i=1}^{N}\sqrt{\|\mathbf{f}(\mathbf{x}_i) - \mathbf{y}_i\|^2}$. This measure is reasonable as the submanifolds contain less patterns than the overall manifold.

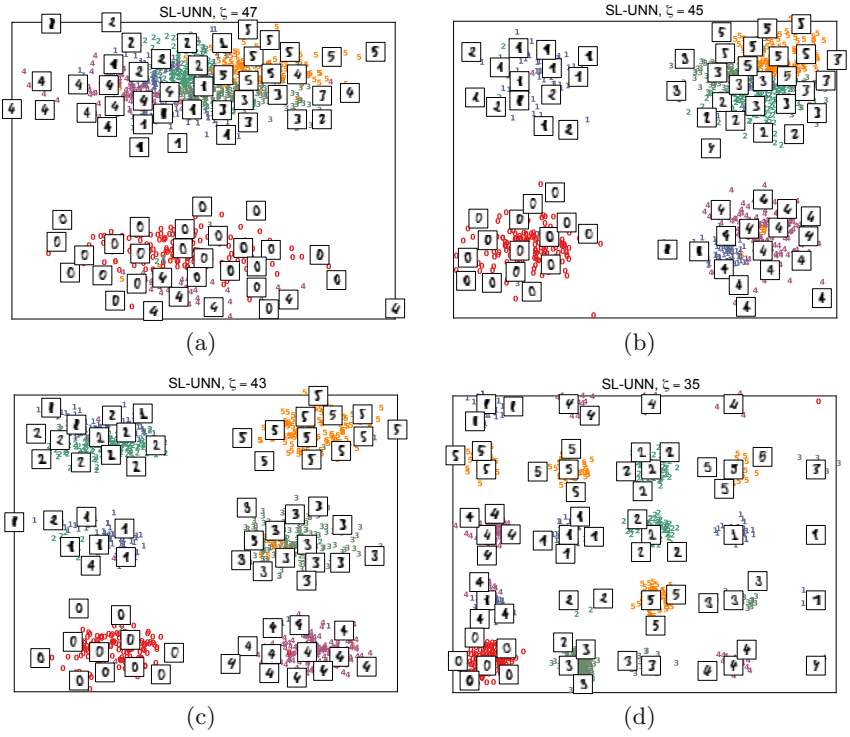


Fig. 7.8 Comparison of SL-UNN embeddings of the *Digits* data set with six classes ('0' to '5') in 2-dimensional manifolds with varying ζ

are compared to the DSRE of the submanifolds $\{\mathcal{M}_j\}_{j=1}^{\tau}$. In Table 7.4, the average DSRE* in each manifold is presented, as well as the DSRE* of the whole embedding. We compare SL-UNN with $\zeta = 43.0$ and $\kappa = 30$ on $N = 1,083$ patterns to KUNN on $N = 180$ patterns with one manifold ($\tau = 1$) corresponding to the average number of patterns in one manifold ($N = 180 \approx 1080/6$). Further, we compare to KUNN (1,083) with $\tau = 1$ employing all $N = 1,083$ patterns.

The results show that SL-UNN assigns the patterns to six submanifolds. In four of the six submanifolds, significantly lower errors are achieved than (1) the average DSRE* of the whole embedding and than (2) KUNN with $N = 180$ patterns (which corresponds to the average number of patterns in each submanifold). The overall DSRE* is slightly higher than the overall DSRE* of KUNN on the whole data set. The reason is that the submanifolds are placed on discrete grid positions not taking into account a sorting w.r.t. the DSRE. Hence, the neighborhood preservation for patterns in different submanifolds can be violated. A modification of SL-UNN could be to *sort*

Table 7.4 Analysis of overall DSRE* and submanifold DSRE* within $\{\mathcal{M}_j\}_{j=1}^{\tau}$ for the *Digits* data set with neighborhood sizes $K = 5, 10, 30$, $\kappa = 30$, and $\zeta = 43.0$

Digits	$\mathcal{M}_1$	$\mathcal{M}_2$	$\mathcal{M}_3$	$\mathcal{M}_4$	$\mathcal{M}_5$	$\mathcal{M}_6$	SL-UNN (1,083)	KUNN (180)	KUNN (1,083)
K	DSRE*						DSRE*	DSRE*	DSRE*
5	2.53	8.12	22.39	6.22	11.86	17.64	14.49 ± 0.12	16.16 ± 0.25	13.83 ± 0.11
10	3.06	9.96	27.41	7.43	13.97	20.46	17.51 ± 0.30	20.94 ± 0.50	16.48 ± 0.065
30	3.63	10.94	35.86	9.33	17.74	23.52	22.44 ± 0.34	28.76 ± 1.28	22.71 ± 1.39

the clusters w.r.t. the DSRE when a new cluster is started. SL-UNN chose the hyperbolic tangent kernel in four of the submanifolds, while the kernel function choice for the other two manifolds varied from RBF-kernel with different settings to the polynomial kernel. The KUNN variants chose the hyperbolic tangent kernel in each experiment.

7.7.3 ISOMAP-Faces

The embedding results of SL-UNN on the ISOMAP-*Faces* data set are shown in Figure 7.9. The figures show the embeddings of SL-UNN with two parameter settings, i.e., $\zeta = 20.0$ and $\zeta = 17.0$. In both experiments, the settings $\kappa = 20$ and neighborhood size $K = 5$ have been chosen. In contrast to the *Digits* data set, no labels are known in advance, and it is difficult to draw conclusions about the correct number of submanifolds. But we can observe that patterns embedded in each manifold have similar characteristics, i.e., similar poses and similar lights. Further, within the manifolds neighbored patterns are neighbored in latent space.

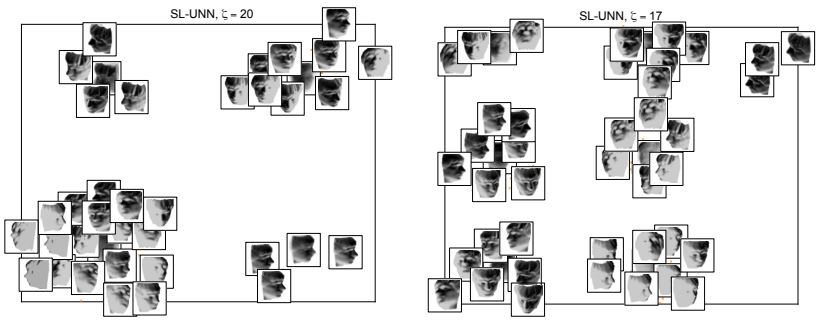


Fig. 7.9 SL-UNN embeddings of the ISOMAP-*Faces* data set with the two parameters $\zeta = 20.0$ and $\zeta = 17.0$

7.8 Conclusions

In this chapter, we introduced an effective approach for embedding patterns to latent spaces of arbitrary dimensionality with an iterative KNN-based strategy. Neighborhoods are preserved employing the KNN-based DSRE, while distances are preserved by Gaussian sampling in latent space with variances proportional to distances in data space. The approach is extended by the concept of kernel-induced feature spaces to handle non-linearities. Various kernel functions are employed, from linear to hyperbolic tangent kernels. The latter achieved surprisingly low DSRE and high co-ranking matrix values. The experiments have shown that KUNN is competitive with famous methods like ISOMAP and LLE. The DSRE decreases with increasing dimensionality of latent space and the number κ of search steps invested in each iteration. Employing various kernel functions for the DSRE turns out to improve the dimensionality reduction result significantly. While the runtime complexity of ISOMAP is $O(N^2 \log N)$ and LLE takes $O(N^2)$, KUNN is computing a manifold in $O(N^2)$ and can be accelerated to $O(N \log N)$ employing space partitioning data structures for the neighborhood queries in data and in latent space, e.g., k-d trees [9] and balltrees [84].

Patterns may lie in different submanifolds with varying characteristics. To allow the assignment of patterns to submanifolds, we have extended UNN by a constructive clustering approach. The novel algorithm called SL-UNN assigns patterns to submanifolds based on a simple yet efficient K-means variant and simultaneously embeds them with UNN. Submanifolds allow the management of separate parameterizations, e.g., kernel functions and kernel parameters in case of KUNN. The overall runtime complexity is still $O(N \log N)$, if space partitioning data structures offer neighborhood requests in $O(\log N)$. The experimental results have shown that SL-UNN achieves low learning errors in the majority of the submanifolds, as each local model can employ separate parameters. Further, a speedup can be observed, as the submanifolds have less patterns than the whole data set. If low errors are required in local neighborhoods, the employment of SL-UNN can be recommended. However, the overall DSRE could not be decreased. The clustering mechanism can be extended by further clustering and manifold criteria like density-based measures. The prize of higher clustering accuracies and possibly lower embedding errors might be paid with worse runtimes.

Part III

Conclusions

8

Summary and Outlook

8.1 Summary

Dimensionality reduction has an important part to play in a world with a steadily growing information infrastructure. Many dimensionality reduction methods have been introduced in the past. For large data sets, efficient methods are required. With UNN and its variants, we have introduced a fast and efficient dimensionality reduction method. All UNN variants compute an embedding in $O(N^2)$ and can be accelerated to $O(N \log N)$, when space partitioning data structures like k-d-trees and balltrees allow efficient neighborhood queries. UNN concentrates on maintaining neighborhoods and distances from the high-dimensional data space in latent space. The variants reach from sorting approaches in 1-dimensional latent spaces to submanifold learning in continuous latent spaces with separate parameterizations for each model. In the following, we summarize the most important results of this work.

8.1.1 From Nearest Neighbors to Dimensionality Reduction

In the first part of this book, we introduced foundations of machine learning and nearest neighbor methods. The choice of neighborhood sizes has an impact on the locality of KNN. Nearest neighbor methods are powerful regression approaches. Besides the introduction of machine learning concepts, a real-world application from the energy domain was introduced. An ensemble of KNN and SVMs has proven to be a strong classifier for nonintrusive appliance load monitoring. We gave an introduction to dimensionality reduction and a short overview of famous methods like SOMs, PCA, ISOMAP, and LLE. The last two methods are also non-linear dimensionality reduction algorithms and have been used as benchmark algorithms in the experimental comparisons. Finally, the framework of unsupervised regression was

O. Kramer: *Dimensionality Reduction with Unsupervised Nearest Neighb.*, ISRL 51, pp. 115–118.
DOI: 10.1007/978-3-642-38652-7_8

introduced, which is one basis of UNN. Important dimensionality reduction quality measures have been reviewed.

8.1.2 From Latent Sorting to Continuous Latent Spaces

UNN is efficiently sorting high-dimensional patterns w.r.t. topological characteristics like distance and neighborhood preservation. The iterative procedures allow the computation of robust unsupervised regression embeddings without initialization with other methods like PCA or LLE in the optimization scheme. The ϵ-insensitive loss allows to learn manifolds in noisy data sets. Missing entries can be completed with KNN preprocessing, or they can first be embedded incompletely and be repaired afterwards. We also analyzed evolutionary variants that achieve a lower learning error, but scale worse with the data set sizes than the iterative variants. For this sake, we compared a simple (1+1)-EA optimization scheme embedding on a lattice structure to an approach based on evolutionary strategies for continuous embeddings, i.e. the CMA-ES. As the continuous approach that optimizes all latent variables at once lacks from the curse of dimensionality, we introduced an iterative learning approach that combines a particle swarm-based method with iterative solution construction. The PSEA turned out to be a successful stochastic variant that allows geometrically free embeddings in arbitrary latent spaces.

8.1.3 Kernel and Submanifold Learning

Last, the particle swarm step of the PSEA is replaced by a Gaussian sampling procedure. Neighborhood preservation is enforced by the DSRE, while distance preservation is achieved by Gaussian sampling employing data space distances as variance. Kernel functions allow better embedding results for non-linear data sets. In particular, the hyperbolic tangent kernel turned out to improve KUNN significantly. A fast way for simultaneous clustering and submanifold learning is the combination of the constructive K-means variant with UNN. The clustering approach is sensitive to a distance threshold that has to be adapted to each data set. The learning results have shown the strengths of assigning patterns to separate manifolds. Various measures, from visualization to co-ranking matrix measures on typical machine learning data sets have been employed in the course of this work.

8.2 Outlook

The development of efficient dimensionality reduction methods will also play an important role in the future and will consequently be a fruitful research field in the next years. Due to the *no free lunch* theorem, the UNN variants

and their parameterizations may not be optimal for every data set. Specific data sets may afford the adaptation of parameters and possibly also of algorithmic mechanisms. For UNN, interesting research directions could be the employment of geodesic distances in data space, which are employed in LLE and ISOMAP. Geodesic distances allow to handle curvatures that follow nonlinear structures (cf. Section 4.6). For this sake, a neighborhood graph has to be computed.

The ensemble principle can also be employed to the iterative manifold construction scheme. The hybridization of multiple criteria of different regression methods allows to increase the flexibility of the embeddings. In discrete latent spaces, it is possible to use a majority vote. In case of free latent embeddings, the mean of the best positions can be used.

The iterative optimization scheme is the key for the success of UNN. But in case of differentiable error functions, further optimization approaches can be applied. A combination of iteratively constructing a solution and gradient descent in each step can be a fruitful direction. For certain problems, it may be reasonable to adapt kernel parameters of KUNN. Evolutionary algorithms can be successful kernel learning algorithms that are often faster than simple grid search.

In Chapter 4, we introduced related methods for unsupervised regression. In particular, UKR is a well analyzed and understood technique. But also UKR suffers from the curse of dimensionality problem and employs LLE and PCA for generating initial embeddings. UNN is can alternative initialization routine for UKR.

A lot of interesting application areas for UNN will be found in the future. An example shows Chapter 1, where we used SOMs for monitoring high-dimensional wind energy time series. Figure 8.1 shows the embeddings of 1,500 high-dimensional wind time series patterns to 2-dimensional latent

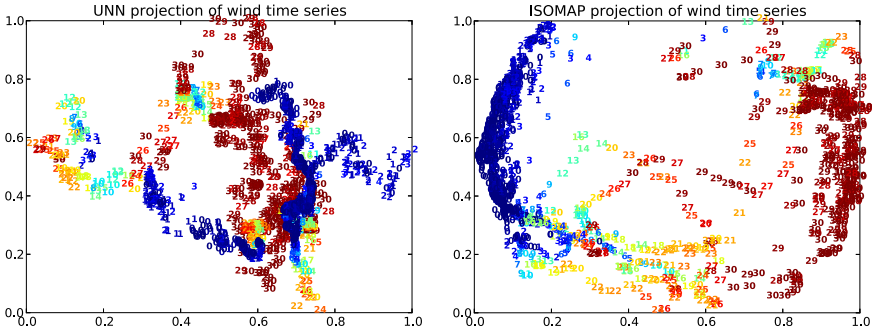


Fig. 8.1 Comparison between (a) UNN and (b) ISOMAP embeddings of high-dimensional wind time series

spaces with UNN and ISOMAP. Each latent point corresponds to a 50-dimensional pattern, i.e., the wind energy produced by 50 wind turbines of a wind park near Tehachapi. The colors are chosen w.r.t. the average wind speed at each step. The plots show that changing wind situations can be recognized as low-dimensional structures in latent space. In particular, the situations with weak (blue) and strong (red) wind correspond to suspicious low-dimensional latent structures.

8.3 Challenges in Dimensionality Reduction

Large non-sparse, but structural data sets are difficult challenges in dimensionality reduction. How can algorithms find structures in data sets with high-dimensional non-sparse data, which may lie in many submanifolds with different distributions and parameterizations. In such a situation, the decision, which information can be neglected, is a difficult task. Efficient processing of large data sets belongs to the most important challenges. To cope with a huge amount of patterns, the development of parallel dimensionality reduction algorithms is an interesting challenge. The distribution of machine learning algorithms to many processors is an appealing approach, as often scaling w.r.t. the number of patterns means to repeat the same steps. These mechanism are *embarrassingly parallelizable*. When an algorithm can divide tasks into subtasks, taking advantage of multi-core machines is a reasonable undertaking, e.g., by computing subtasks on different processors and combining smaller solution parts to the final solution. The search for codebook vectors and neighborhood queries can be parallelized.

A further interesting challenge is time series prediction, which becomes more and more important in practical applications, e.g., in energy systems. For the integration of renewable energy sources, a precise forecast is a difficult task. The volatileness of wind and solar energy renders the precise forecast difficult. In [65], we have introduced a model that formulates time series prediction as regression problem. The approach is a spatio-temporal time series formulation based on SVR and employs data from the US National Renewable Energy Laboratory [89]. When many wind turbines and parks take part in the prediction process, time series analysis becomes a high-dimensional data mining problem. The question arises, how to reduce the dimensionality of time series data, but at the same time maintain the important intrinsic structure for precise predictions.

A

Test Problems

In this work, we evaluate and compare the proposed methods on benchmark test problems. The data sets are summarized in the following.

A.1 Astronomy

Our experimental study is based on data from the Sloan Digital Sky Survey (SDSS) [2]. In [6], the EFIGI catalog of 4,458 nearby galaxies is presented. We concentrate on a subset of 100 galaxies of types -6 to -4 and 7 to 11. The images are 40×40 RGB images. The data has been preprocessed: all entries below a threshold $\psi = 0.1$ have been set to 0. Figure A.1 shows a selection of eight galaxies from the EFIGI data set. The galaxy data set is employed in the experimental analysis of Chapter 5.

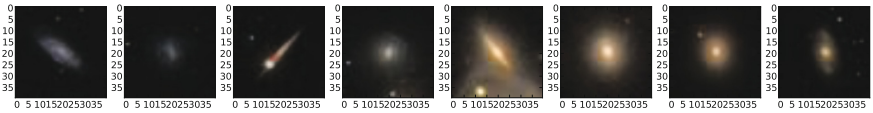


Fig. A.1 Visualization of eight sample galaxies from the EFIGI data set

A.2 Boston

The Boston housing data set stems from 506 census tracts of Boston in 1970. It consists of $N = 506$ patterns with $d = 13$ features (positive real values), e.g., *proportion of owner-occupied units built prior to 1940* and *weighted distances to five Boston employment centers*. The original data has been published by Harrison and Rubinfeld [38]. The data set was taken from the *StatLib* library, which is maintained at Carnegie Mellon University.

A.3 Digits

The *Digits* data set [46] comprises handwritten digits and is often employed as reference problem related to the recognition of handwritten characters and digits. Figure A.2 shows a collection of images from the *Digits* data set. The collection shows all ten digits, while most experimental analyses concentrate on a label subset.

Fig. A.2 Visualization of a collection of images from the *Digits* data set

A.4 NIALM Data

The NIALM data sets have been recorded in collaboration with the OFFIS[1], Oldenburg. The *install* data set consists of 120 patterns that have manually been recorded and labeled. The *field study* data set consists of patterns that have been recorded in a household test environment. Table A.1 shows the appliances that are part of the test data sets *install* and of the *field study*.

Table A.1 List of 15 appliances of the *install* and the *field study* data set

#	appliances	#	appliances	#	appliances
1	shelf light	6	table light, bedroom	11	ceiling lamp, bathroom
2	fridge	7	table light, TV	12	ceiling lamp, living room
3	bedside lamp	8	table light, door	13	ceiling lamp, corridor
4	desk lamp	9	kettle	14	ceiling lamp, bedroom
5	TV	10	mirror lamps	15	air conditioning

A.5 ISOMAP-Faces

The ISOMAP-*Faces* data set has been used in the experimental study of the original ISOMAP article [103]. It consists of $N = 698$ images of faces of a statue with varying *poses* and *lights*. The data set can be downloaded from `http://isomap.stanford.edu/`. Figure A.3 shows the first 45 images of the data set.

[1] Oldenburger Forschungs- und Entwicklungsinstitut für Informatik.

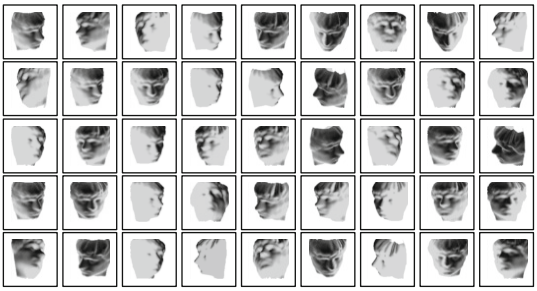


Fig. A.3 Visualization of a collection of images from the *Faces* data set

A.6 S-Structure

The 3-dimensional S data set consists of 500 points in the version 3D-S without hole, see Part (a) of Figure A.4. The counterpart 3D-S$_h$ with hole consists of approximately 350 points.

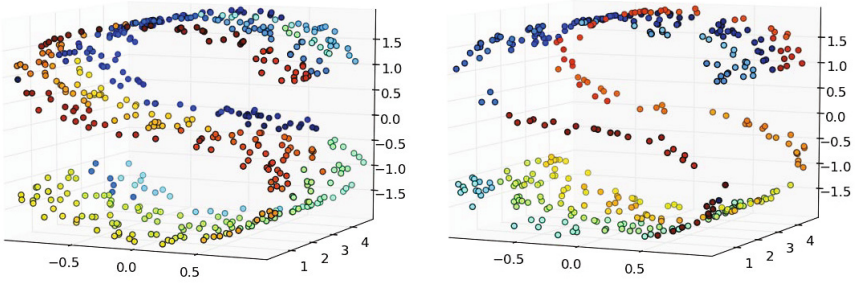


Fig. A.4 3-dimensional S: (a) the 3D-S data set and (b) the 3D-S$_h$ data set with hole

References

1. Vicon motion capture system, http://www.vicon.com/
2. SDSS 2012, sloan digital sky survey (2012), http://www.sdss.org
3. Abraham, A., Grosan, C., Ramos, V. (eds.): Swarm Intelligence in Data Mining. SCI, vol. 34. Springer (2006)
4. Aronszajn, N.: Theory of reproducing kernels. Transactions of the American Mathematical Society 68(3), 337–404 (1950)
5. Bailey, T., Jain, A.: A note on distance-weighted k-nearest neighbor rules. IEEE Transaction on Systems, Man and Cybernetics 8(4), 311–313 (1978)
6. Baillard, A., Bertin, E., de Lapparent, V., Fouqué, P., Arnouts, S., Mellier, Y., Pelló, R., Leborgne, J.-F., Prugniel, P., Markarov, D., Makarova, L., McCracken, H.J., Bijaoui, A., Tasca, L.: The EFIGI catalogue of 4458 nearby galaxies with detailed morphology 532, A74 1103.5734 (2011)
7. Baranski, M., Voss, J.: Genetic algorithm for pattern detection in NIALM systems. IEEE Transaction on Systems, Man and Cybernetics 4, 3462–3468 (2004)
8. Baruque, B., Corchado, E.: Fusion Methods for Unsupervised Learning Ensembles. SCI, vol. 322. Springer, Heidelberg (2011)
9. Bentley, J.L.: Multidimensional binary search trees used for associative searching. Communications of the ACM 18(9), 509–517 (1975)
10. Beyer, H.-G.: An alternative explanation for the manner in which genetic algorithms operate. BioSystems 41, 1–15 (1997)
11. Beyer, H.-G., Schwefel, H.-P.: Evolution strategies - A comprehensive introduction. Natural Computing 1, 3–52 (2002)
12. Bishop, C.M.: Pattern Recognition and Machine Learning (Information Science and Statistics). Springer (2007)
13. Bradley, P.S., Mangasarian, O.L.: k-plane clustering. Journal of Global Optimization 16, 23–32 (2000)
14. Breiman, L.: Bagging predictors. Machine Learning 24(2), 123–140 (1996)
15. Carreira-Perpiñán, M.Á., Lu, Z.: Parametric dimensionality reduction by unsupervised regression. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1895–1902 (2010)
16. Chang, H., Yeung, D.: Robust locally linear embedding. Pattern Recognition 39, 1053–1065 (2006)

17. Chang, H.-H., Yang, H.-T., Lin, C.-L.: Load identification in neural networks for a non-intrusive monitoring of industrial electrical loads. In: Shen, W., Yong, J., Yang, Y., Barthès, J.-P.A., Luo, J. (eds.) CSCWD 2007. LNCS, vol. 5236, pp. 664–674. Springer, Heidelberg (2008)
18. Chechik, G., Heitz, G., Elidan, G., Abbeel, P., Koller, D.: Max-margin classification of data with absent features. Journal of Machine Learning Research 9, 1–21 (2008)
19. Comon, P.: Independent component analysis, a new concept? Signal Processing 36(3), 287–314 (1994)
20. Costeira, J.P., Kanade, T.: A multibody factorization method for independently moving objects. International Journal of Computer Vision 29(3), 159–179 (1998)
21. Cover, T., Hart, P.: Nearest neighbor pattern classification 13, 21–27 (1967)
22. Dick, U., Haider, P., Scheffer, T.: Learning from incomplete data with infinite imputations. In: International Conference on Machine Learning (ICML), pp. 232–239 (2008)
23. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik 8, 269–271 (1959)
24. Dorigo, M., Birattari, M.: Ant colony optimization. In: Encyclopedia of Machine Learning, pp. 36–39 (2010)
25. Dunn, J.C.: A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters. Journal of Cybernetics 3(3), 32–57 (1973)
26. Elhamifar, E., Vidal, R.: Sparse subspace clustering. In: IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2790–2797 (2009)
27. Elhamifar, E., Vidal, R.: Sparse manifold clustering and embedding. In: Advances in Neural Information Processing Systems (NIPS), pp. 55–63 (2011)
28. Fix, E., Hodges, J.: Discriminatory analysis, nonparametric discrimination: Consistency properties. 4 (1951)
29. Freund, Y., Schapire, R.E.: A decision-theoretic generalization of on-line learning and an application to boosting. In: European Conference on Computational Learning Theory (EuroCOLT), pp. 23–37 (1995)
30. Friedman, J.H., Tukey, J.W.: A projection pursuit algorithm for exploratory data analysis. IEEE Transactions on Computers C-23(9), 881–890 (1974)
31. Gear, C.W.: Multibody grouping from motion images. International Journal of Computer Vision 29(2), 133–150 (1998)
32. Ghahramani, Z., Jordan, M.I.: Supervised learning from incomplete data via an EM approach. In: Advances in Neural Information Processing Systems (NIPS), pp. 120–127 (1993)
33. Gieseke, F., Polsterer, K.L., Thom, A., Zinn, P., Bomanns, D., Dettmar, R.-J., Kramer, O., Vahrenhold, J.: Detecting quasars in large-scale astronomical surveys. In: International Conference on Machine Learning and Applications (ICMLA), pp. 352–357 (2010)
34. Goh, A., Vidal, R.: Segmenting motions of different types by unsupervised manifold clustering. In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR (2007)
35. Goldberg, D.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Longman Publishing, Boston (1989)
36. Guyon, I., Elisseeff, A.: An introduction to variable and feature selection. Journal of Machine Learning Research 3, 1157–1182 (2003)

37. Härdle, W., Simar, L.: Applied Multivariate Statistical Analysis. Springer, Heidelberg (2007)
38. Harrison, D., Rubinfeld, D.: Hedonic prices and the demand for clean air. Journal on Environmental Economics an Management 5, 81–102 (1978)
39. Hart, G.W.: Nonintrusive appliance load monitoring. Proceedings of the IEEE 80(12), 1870–1891 (1992)
40. Hastie, T., Tibshirani, R., Friedman, J.: The Elements of Statistical Learning. Springer, Berlin (2009)
41. Herrmann, L., Ultsch, A.: The architecture of ant-based clustering to improve topographic mapping. In: Dorigo, M., Birattari, M., Blum, C., Clerc, M., Stützle, T., Winfield, A.F.T. (eds.) ANTS 2008. LNCS, vol. 5217, pp. 379–386. Springer, Heidelberg (2008)
42. Holland, J.H.: Hidden Order: How Adaptation Builds Complexity. Addison-Wesley, Reading (1995)
43. Hubble, E.P.: Extra-galactic nebulae. Astrophysical Journal (64), 321–369 (1926)
44. Huber, P.J.: Robust Statistics. Wiley, New York (1981)
45. Huber, P.J.: Projection pursuit. Annals of Statistics 13(2), 435–475 (1985)
46. Hull, J.: A database for handwritten text recognition research. IEEE Transactions on Pattern Analysis and Machine Intelligence 16(5), 550–554 (1994)
47. Jolliffe, I.: Principal component analysis. Springer series in statistics. Springer, New York (1986)
48. Kao, Y., Cheng, K.: An ACO-based clustering algorithm. In: Dorigo, M., Gambardella, L.M., Birattari, M., Martinoli, A., Poli, R., Stützle, T. (eds.) ANTS 2006. LNCS, vol. 4150, pp. 340–347. Springer, Heidelberg (2006)
49. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of IEEE International Conference on Neural Networks, pp. 1942–1948 (1995)
50. Kitchin, C.: Galaxies in Turmoil - The Active and Starburst Galaxies and the Black Holes That Drive Them. Springer, New York (2007)
51. Kiviluoto, K.: Topology preservation in self-organizing maps. In: IEEE International Conference on Neural Networks (ICNN), pp. 294–299 (1996)
52. Klanke, S.: Learning Manifolds with the Parametrized Self-Organizing Map and Unsupervised Kernel Regression. PhD thesis, University of Bielefeld (2007)
53. Klanke, S., Ritter, H.: Variants of unsupervised kernel regression: General cost functions. Neurocomputing 70(7-9), 1289–1303 (2007)
54. Kohonen, T.: Self-Organizing Maps. Springer (2001)
55. Kramer, O.: Dimensionality reduction by unsupervised nearest neighbor regression. In: International Conference on Machine Learning and Applications (ICMLA), pp. 275–278. IEEE (2011)
56. Kramer, O.: On machine symbol grounding and optimization. International Journal of Cognitive Informatics and Natural Intelligence (IJCINI) 5(3), 73–85 (2011)
57. Kramer, O.: On evolutionary approaches to unsupervised nearest neighbor regression. In: Di Chio, C., et al. (eds.) EvoApplications 2012. LNCS, vol. 7248, pp. 346–355. Springer, Heidelberg (2012)
58. Kramer, O.: On unsupervised nearest-neighbor regression and robust loss functions. In: International Conference on Agents and Artificial Intelligence (ICAART), pp. 164–170 (2012)

59. Kramer, O.: A particle swarm embedding algorithm for nonlinear dimensionality reduction. In: Dorigo, M., Birattari, M., Blum, C., Christensen, A.L., Engelbrecht, A.P., Groß, R., Stützle, T. (eds.) ANTS 2012. LNCS, vol. 7461, pp. 1–12. Springer, Heidelberg (2012)
60. Kramer, O.: Unsupervised nearest neighbors with kernels. In: Glimm, B., Krüger, A. (eds.) KI 2012. LNCS, vol. 7526, pp. 97–106. Springer, Heidelberg (2012)
61. Kramer, O.: Fast Submanifold Learning with Unsupervised Nearest Neighbors. In: Tomassini, M., Antonioni, A., Daolio, F., Buesser, P. (eds.) ICANNGA 2013. LNCS, vol. 7824, pp. 317–325. Springer, Heidelberg (2013)
62. Kramer, O., Gieseke, F.: A stochastic optimization approach for unsupervised kernel regression. In: International Conference on Genetic and Evolutionary Methods (GEM), pp. 520–525. CSREA Press (2011)
63. Kramer, O., Gieseke, F.: Evolutionary kernel density regression. Expert Systems and Applications 39(10), 9246–9254 (2012)
64. Kramer, O., Ciaurri, D.E., Koziel, S.: Derivative-Free Optimization. Springer, Berlin (2011)
65. Kramer, O., Gieseke, F.: Short-term wind energy forecasting using support vector regression. In: Corchado, E., Snášel, V., Sedano, J., Hassanien, A.E., Calvo, J.L., Ślęzak, D. (eds.) SOCO 2011. AISC, vol. 87, pp. 271–280. Springer, Heidelberg (2011)
66. Kramer, O., Gieseke, F.: Evolutionary kernel density regression. Expert Systems and Applications 39(10), 9246–9254 (2012)
67. Kramer, O., Satzger, B., Lässig, J.: Power prediction in smart grids with evolutionary local kernel regression. In: Graña Romay, M., Corchado, E., Garcia Sebastian, M.T. (eds.) HAIS 2010, Part I. LNCS, vol. 6076, pp. 262–269. Springer, Heidelberg (2010)
68. Kramer, O., Gieseke, F., Satzger, B.: Wind Energy Prediction, and Monitoring with Neural Computation. Neurocomputing Journal 109, 84–93 (2013)
69. Kramer, O., Wilken, O., Beenken, P., Hein, A., Hüwel, A., Klingenberg, T., Meinecke, C., Raabe, T., Sonnenschein, M.: On ensemble classifiers for nonintrusive appliance load monitoring. In: Corchado, E., Snášel, V., Abraham, A., Woźniak, M., Graña, M., Cho, S.-B. (eds.) HAIS 2012, Part III. LNCS, vol. 7208, pp. 322–331. Springer, Heidelberg (2012)
70. Kruskal, J.: Nonmetric multidimensional scaling: a numerical method. Psychometrika 29(2), 115–129 (1964)
71. Lawrence, N.D.: Probabilistic non-linear principal component analysis with gaussian process latent variable models. Journal of Machine Learning Research 6, 1783–1816 (2005)
72. Lee, J.A., Verleysen, M.: Nonlinear Dimensionality Reduction. Springer, Berlin (2007)
73. Lee, J.A., Verleysen, M.: Quality assessment of dimensionality reduction: Rank-based criteria. Neurocomputing 72(7-9), 1431–1443 (2009)
74. Lin, Y.-H., Tsai, M.-S.: Applications of hierarchical support vector machines for identifying load operation in nonintrusive load monitoring systems. In: Intelligent Control and Automation (WCICA), pp. 688–693 (2011)
75. Lueks, W., Mokbel, B., Biehl, M., Hammer, B.: How to evaluate dimensionality reduction? - improving the co-ranking matrix. CoRR abs/1110.3917 (2011)

76. Ma, Y., Derksen, H., Hong, W., Wright, J.: Segmentation of multivariate mixed data via lossy data coding and compression. IEEE Transaction on Pattern Analysis and Machine Intelligence 29(9), 1546–1562 (2007)
77. Martin, M., Maycock, J., Schmidt, F.P., Kramer, O.: Recognition of manual actions using vector quantization and dynamic time warping. In: Graña Romay, M., Corchado, E., Garcia Sebastian, M.T. (eds.) HAIS 2010, Part I. LNCS, vol. 6076, pp. 221–228. Springer, Heidelberg (2010)
78. Martinetz, T., Schulten, K.: A neural gas network learns topologies. In: Artificial Neural Networks, pp. 397–402. Elsevier (1991)
79. Meinicke, P.: Unsupervised Learning in a Generalized Regression Framework. PhD thesis, University of Bielefeld (2000)
80. Meinicke, P., Klanke, S., Memisevic, R., Ritter, H.: Principal surfaces from unsupervised kernel regression. IEEE Transaction on Pattern Analysis and Machine Intelligence 27(9), 1379–1391 (2005)
81. Mitchell, T.: Machine Learning. McGraw Hill (1997)
82. Nadaraya, E.: On estimating regression. Theory of Probability and Its Application 10, 186–190 (1964)
83. Nourashrafeddin, S., Arnold, D., Milios, E.E.: An evolutionary subspace clustering algorithm for high-dimensional data. In: Proceedings of the Annual Conference on Genetic and Evolutionary Computation (GECCO), pp. 1497–1498 (2012)
84. Omohundro, S.M.: Five balltree construction algorithms. Technical report, International Computer Science Institute, ICSI (1989)
85. O'Neill, M., Brabazon, A.: Self-organizing swarm (SOSwarm) for financial credit-risk assessment. In: IEEE Congress on Evolutionary Computation (CEC), pp. 3087–3093 (2008)
86. Ostermeier, A., Gawelczyk, A., Hansen, N.: A derandomized approach to self adaptation of evolution strategies. Evolutionary Computation 2(4), 369–380 (1994)
87. Patel, S.N., Robertson, T., Kientz, J.A., Reynolds, M.S., Abowd, G.D.: At the flick of a switch: Detecting and classifying unique electrical events on the residential power line (Nominated for the best paper award). In: Krumm, J., Abowd, G.D., Seneviratne, A., Strang, T. (eds.) UbiComp 2007. LNCS, vol. 4717, pp. 271–288. Springer, Heidelberg (2007)
88. Pearson, K.: On lines and planes of closest fit to systems of points in space. Philosophical Magazine 2(6), 559–572 (1901)
89. Potter, C.W., Lew, D., McCaa, J., Cheng, S., Eichelberger, S., Grimit, E.: Creating the dataset for the western wind and solar integration study (u.s.a.). In: International Workshop on Large Scale Integration of Wind Power and on Transmission Networks for Offshore Wind Farms (2008)
90. Rechenberg, I.: Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Frommann-Holzboog, Stuttgart (1973)
91. Riedmiller, M., Braun, H.: A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In: Proceedings of the IEEE International Conference on Neural Networks, pp. 586–591 (1993)
92. Ritter, H., Martinetz, T., Schulten, K.: Neural Computation and Self-Organizing Maps: An Introduction. Addison Wesley (1992)
93. Rojas, R.: Neural Networks - A Systematic Introduction. Springer (1996)
94. Roweis, S.T., Saul, L.K.: Nonlinear dimensionality reduction by locally linear embedding. Science 290, 2323–2326 (2000)

95. Rudolph, G.: Finite Markov chain results in evolutionary computation: A tour d'horizon. Fundamenta Informaticae 35(1-4), 67–89 (1998)
96. Rumelhart, D., Hintont, G., Williams, R.: Learning representations by back-propagating errors. Nature 323(6088), 533–536 (1986)
97. Schafer, J.L., Graham, J.W.: Missing data: Our view of the state of the art. Psychological Methods 7(2), 147–177 (2002)
98. Schölkopf, B., Smola, A.J.: Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press, Cambridge (2001)
99. Smola, A.J., Mika, S., Schölkopf, B., Williamson, R.C.: Regularized principal manifolds. Journal of Machine Learning Research 1, 179–209 (2001)
100. Steffen, J., Pardowitz, M., Ritter, H.: Using structured UKR manifolds for motion classification and segmentation. In: IEEE International Conference on Intelligent Robots and Systems (IROS), pp. 4785–4790 (2009)
101. Suykens, J.A.K., Vandewalle, J.: Least squares support vector machine classifiers. Neural Processing Letters 9(3), 293–300 (1999)
102. Tan, S., Mavrovouniotis, M.: Reducing data dimensionality through optimizing neural network inputs. AIChE Journal 41(6), 1471–1479 (1995)
103. Tenenbaum, J.B., Silva, V.D., Langford, J.C.: A global geometric framework for nonlinear dimensionality reduction. Science 290, 2319–2323 (2000)
104. Tipping, M.E., Bishop, C.M.: Mixtures of probabilistic principal component analysers. Neural Computation 11(2), 443–482 (1999)
105. Tseng, P.: Nearest q-flat to m points. Journal of Optimization Theory and Applications 105, 249–252 (2000)
106. Vahdat, A., Heywood, M.I., Zincir-Heywood, A.N.: Bottom-up evolutionary subspace clustering. In: IEEE Congress on Evolutionary Computation, pp. 1–8 (2010)
107. Vidal, R.: Subspace clustering. IEEE Signal Processing Magazine 28(2), 52–68 (2011)
108. Vidal, R., Ma, Y., Sastry, S.: Generalized principal component analysis (gpca). IEEE Transactions on Pattern Analysis and Machine Intelligence 27(12), 1945–1959 (2005)
109. von Luxburg, U.: A tutorial on spectral clustering. Statistics and Computing 17, 1–24 (2007)
110. Watson, G.: Smooth regression analysis. Sankhya Series A 26, 359–372 (1964)
111. Wendemuth, A.: Grundlagen der stochastischen Sprachverarbeitung. Oldenbourg (2004)
112. Williams, D., Liao, X., Xue, Y., Carin, L., Krishnapuram, B.: On classification with incomplete data. IEEE Transactions on Pattern Analysis and Machine Intelligence 29(3), 427–436 (2007)
113. Wolpert, D.H., Macready, W.G.: No free lunch theorems for search. Technical report, Santa Fe, NM (1995)
114. Yan, J., Pollefeys, M.: A general framework for motion segmentation: Independent, articulated, rigid, non-rigid, degenerate and non-degenerate. In: Leonardis, A., Bischof, H., Pinz, A. (eds.) ECCV 2006. LNCS, vol. 3954, pp. 94–106. Springer, Heidelberg (2006)
115. Zhang, T., Szlam, A., Wang, Y., Lerman, G.: Hybrid linear modeling via local best-fit flats. International Journal of Computer Vision, 1–24 (2012)
116. Zhu, X., Goldberg, A.B.: Introduction to Semi-Supervised Learning. Morgan and Claypool (2009)

Index